# ITK Sphinx Examples Documentation

***Release 5.3.0***

**Insight Software Consortium**

**Jun 16, 2021**

# CONTENTS

Contents:

CHAPTER

# ONE

# HOW TO BUILD, RUN, AND VISUALIZE

The Python examples do not need to be built. First, install Python from python.org, Anaconda, or a system package manager like Ubuntu Linux apt or macOS Homebrew.

Next, install the *itk* package:

```
python -m pip install --upgrade pip
python -m pip install itk
```

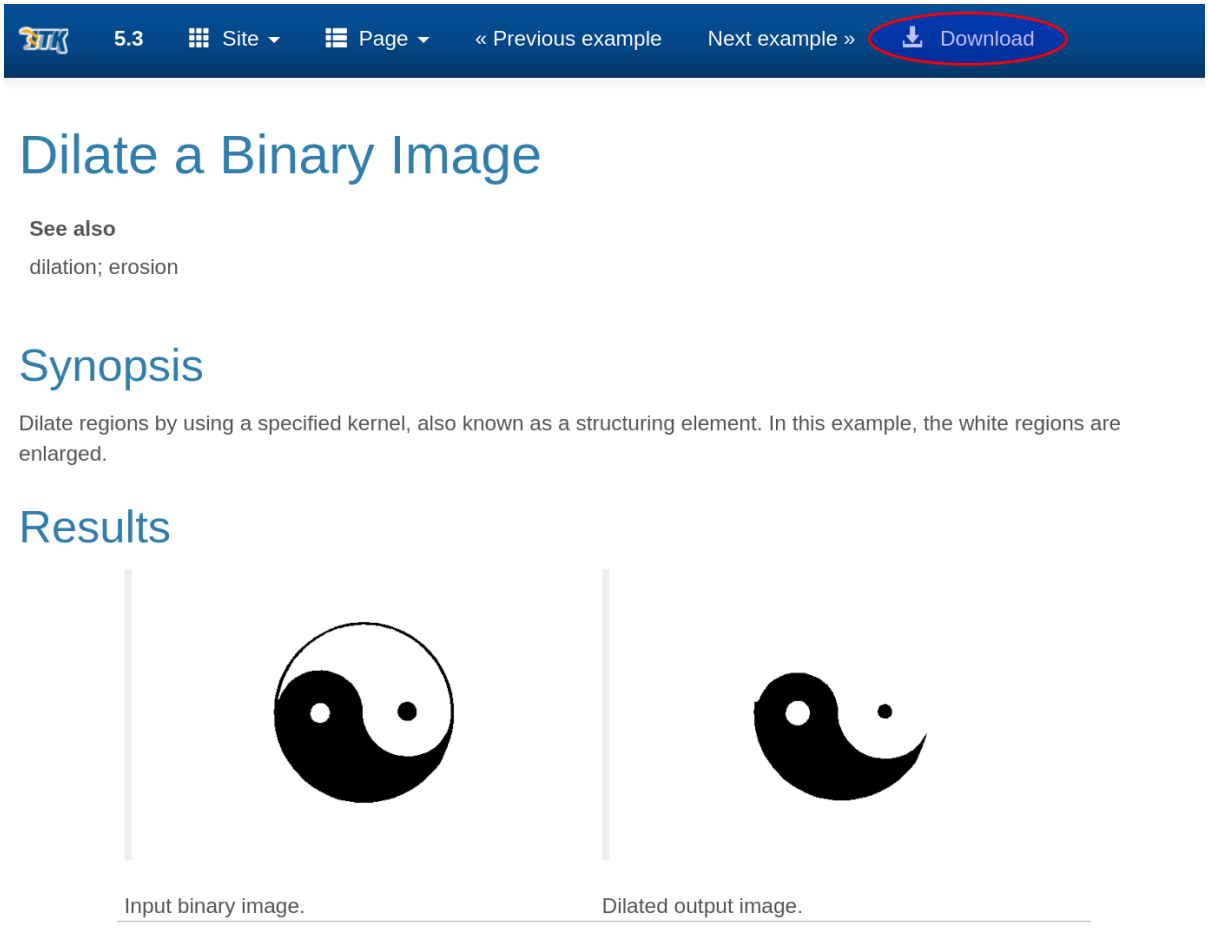



Fig. 1: Download an example by clicking the *Download* button.

Download the <*ExampleName*>.*zip* file from the link on the top of an example's webpage. Unpack the example. For

example,

```
unzip ExampleName.zip
```

Finally, run the example:

```
python ExampleName/Code.py
```

## 1.1 Build individual examples

Download the *<ExampleName>.zip* file from the link on the top of the example's webpage. Unpack the example:

```
unzip ExampleName.zip
```

Set convenience variables:

```
ITK_SOURCE=ExampleName
ITK_BUILD=ExampleName/build
```

Move to the build directory:

```
cd ${ITK_BUILD}
```

Run CMake (minimum version 3.10.2) to configure the project.

- If ITK Version 5.0.0 or above not installed but compiled on your system, you will need to specify the path to your ITK build:

  ```
  cmake -DITK_DIR=/home/luis/itk_build -S ${ITK_SOURCE} -B ${ITK_BUILD}
  ```

Build the project, and run the test:

```
cmake --build .
ctest -V
```

## 1.2 Build all examples and the documentation

Set convenience variables:

```
ITK_SOURCE=ITKEx
ITK_BUILD=ITKEx-build
```

Clone the repository:

```
git clone --recursive https://github.com/InsightSoftwareConsortium/ITKSphinxExamples.
↪git ${ITK_SOURCE}
```

Make a build directory:

```
mkdir -p ${ITK_BUILD}
cd ${ITK_BUILD}
```

Run CMake (minimum version 3.10.2) to configure the project.

- If ITK is not installed nor compiled, you can then make use of the superbuild functionality:

```
cmake -DBUILD_DOCUMENTATION:BOOL=ON -S ${ITK_SOURCE}/Superbuild/ -B ${ITK_BUILD}
```

- If ITK (Version 5.0.0 or above) is installed:

```
cmake -DBUILD_DOCUMENTATION:BOOL=ON -S ${ITK_SOURCE} -B ${ITK_BUILD}
```

- If ITK (Version 5.0.0 or above) is not installed but compiled on your system, you will need to specify the path to your ITK build:

```
ITK_COMPILED_DIR=/usr/opt/itk_build
cmake -DITK_DIR=${ITK_COMPILED_DIR} -DBUILD_DOCUMENTATION:BOOL=ON -S ${ITK_SOURCE_
↪DIR} -B ${ITK_BUILD_DIR}
```

The superbuild will download and build all required dependencies. If you are building the documentation and not using superbuild functionality, then you must have all the required dependencies installed, which are listed in the *README.rst* file located at the root of the source tree. If you just want to build the examples and not their documentation, set *BUILD_DOCUMENTATION* to *OFF* in your CMake configuration.

Build the project (this will generate the documentation and all examples):

```
cmake --build .
```

Run the tests with a superbuild:

```
cd ${ITK_BUILD_DIR}/ITKEX-build
ctest -V
```

Run the tests without a superbuild:

```
cd ${ITK_BUILD_DIR}
ctest -V
```

## 1.3 Run an example

After building the examples, you can run an example by using *cd* to move to the example's directory. Then, directly run the executable.

Alternatively, the *ctest* command line program that comes with CMake can be used to drive the examples as unit test. Running:

```
ctest
```

in the binary tree will run all unit tests found in the current directory and below.

```
ctest -R Binary
```

will run all tests whose name matches the regular expression *Binary*.

```
ctest -V
```

will run *ctest* in verbose mode, which prints out the command executed and all of the resulting text output.

## 1.4 Visualize the results

ITK is a library limited in scope to image analysis, and it purposely does not attempt to perform image visualization. Visualizing the results of analysis is possible with a number of third-party applications. Note that these packages are specially suited for medical images, which often have anisotropic spacing and can span three or more dimensions. All applications listed are open source and cross-platform.

**3DSlicer** 3DSlicer is an open-source software platform for the analysis and visualization of medical images and for research in image guided therapy. The platform provides functionality for segmentation, registration and three-dimensional visualization of multi-modal image data, as well as advanced image analysis algorithms for diffusion tensor imaging, functional magnetic resonance imaging and image-guided therapy. Standard image file formats are supported, and the application integrates interface capabilities to biomedical research software and image informatics frameworks.

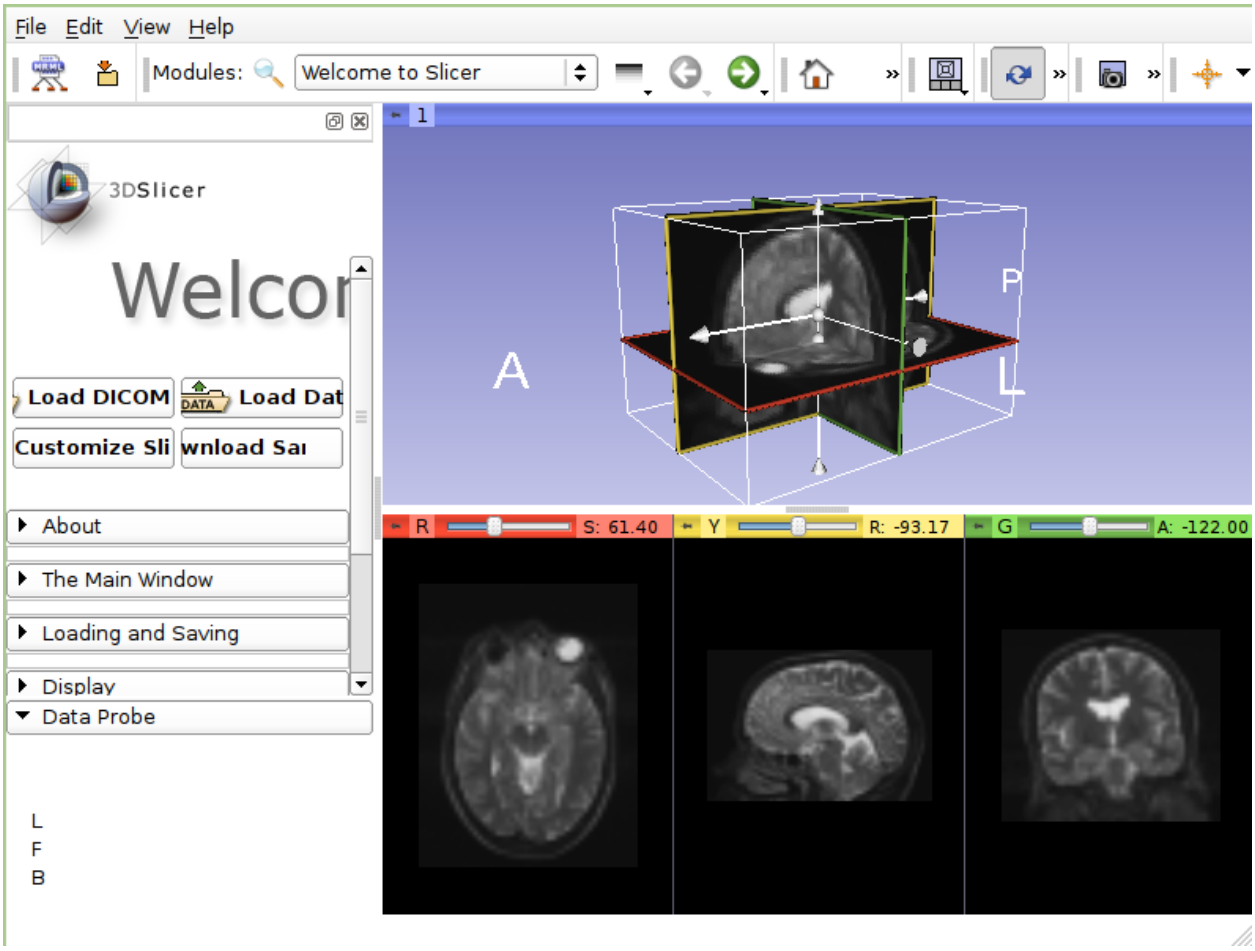



Fig. 2: 3DSlicer

**ImageViewer** An FLTK-based ImageViewer was extracted from the ITKApps repository. This simple yet effective slice-based viewer works on 2D and 3D images and supports probing of data values.

**ITK-SNAP** ITK-SNAP is segmentation application, but it is also a nice general resource for visualization of the results of analysis.

**MITK** MITK is a free open-source software system for development of interactive medical image processing software.


Fig. 3: ImageViewer

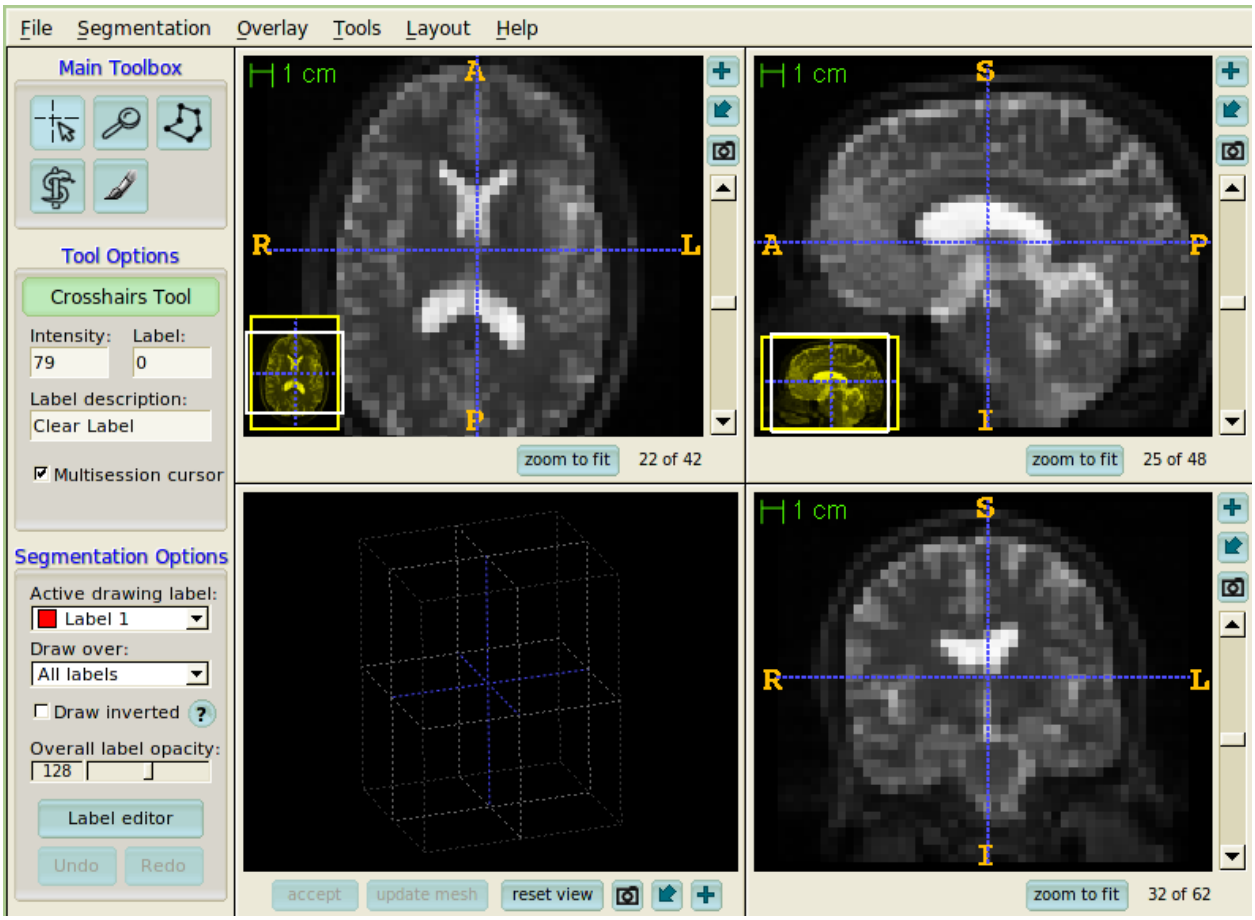



Fig. 4: ITK-SNAP

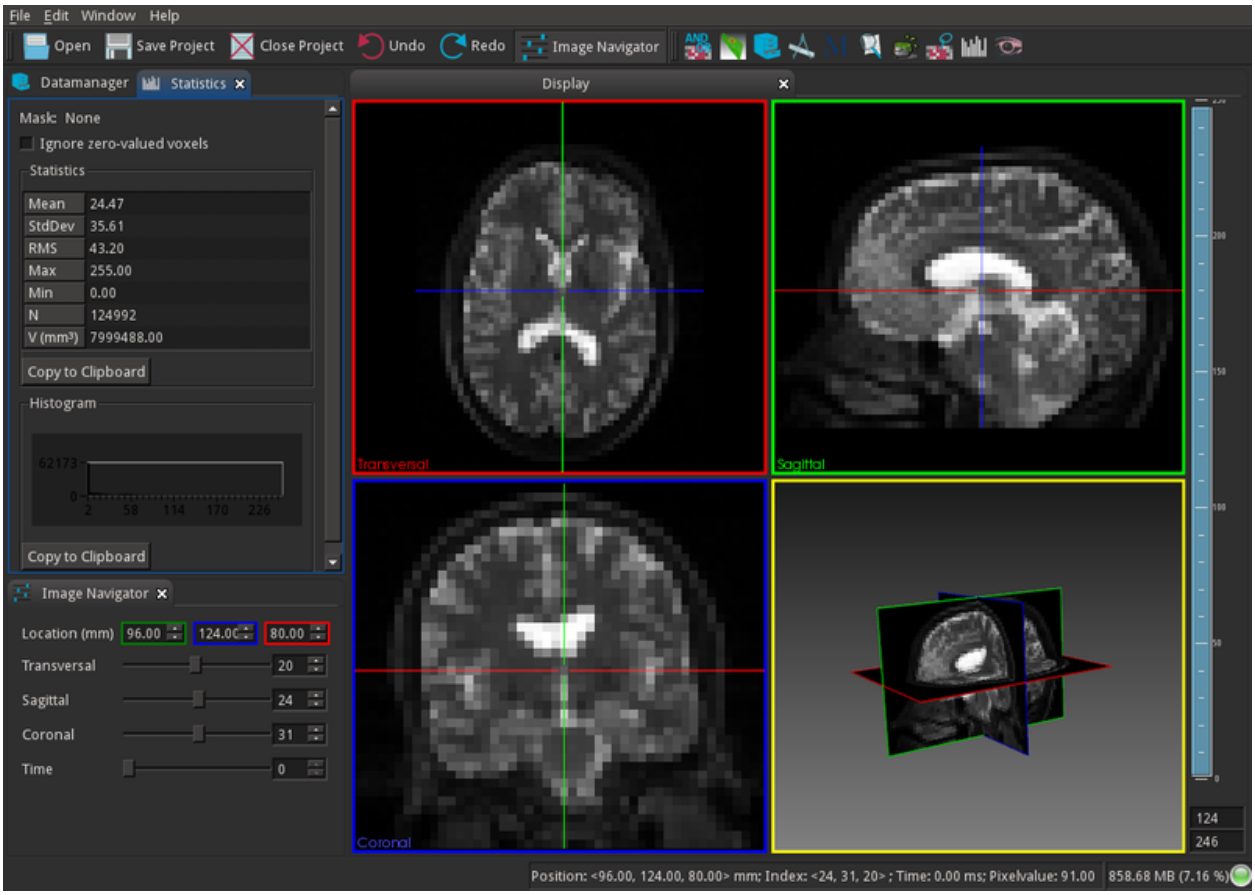

File Edit Window Help
Open
Save Project
Close Project
Undo
Redo
Image Navigator
Datamanager
Statistics
Display
Mask: None
Ignore zero-valued voxels
Statistics
Mean 24.47
StdDev 35.61
RMS 43.20
Max 255.00
Min 0.00
N 124992
V (mm³) 7999488.00
Copy to Clipboard
Histogram
62173
0
2 58 114 170 226
Copy to Clipboard
Image Navigator
Location (mm) 96.00 124.00 80.00
Transversal 20
Sagittal 24
Coronal 31
Time 0
Transversal
Sagittal
Coronal
124
246
Position: <96.00, 124.00, 80.00> mm; Index: <24, 31, 20> ; Time: 0.00 ms; Pixelvalue: 91.00
858.68 MB (7.16 %)

Fig. 5: MITK

**Paraview** Paraview is a full-featured scientific visualizion GUI written with Qt/VTK. It has extensive parallel processing capabilities.

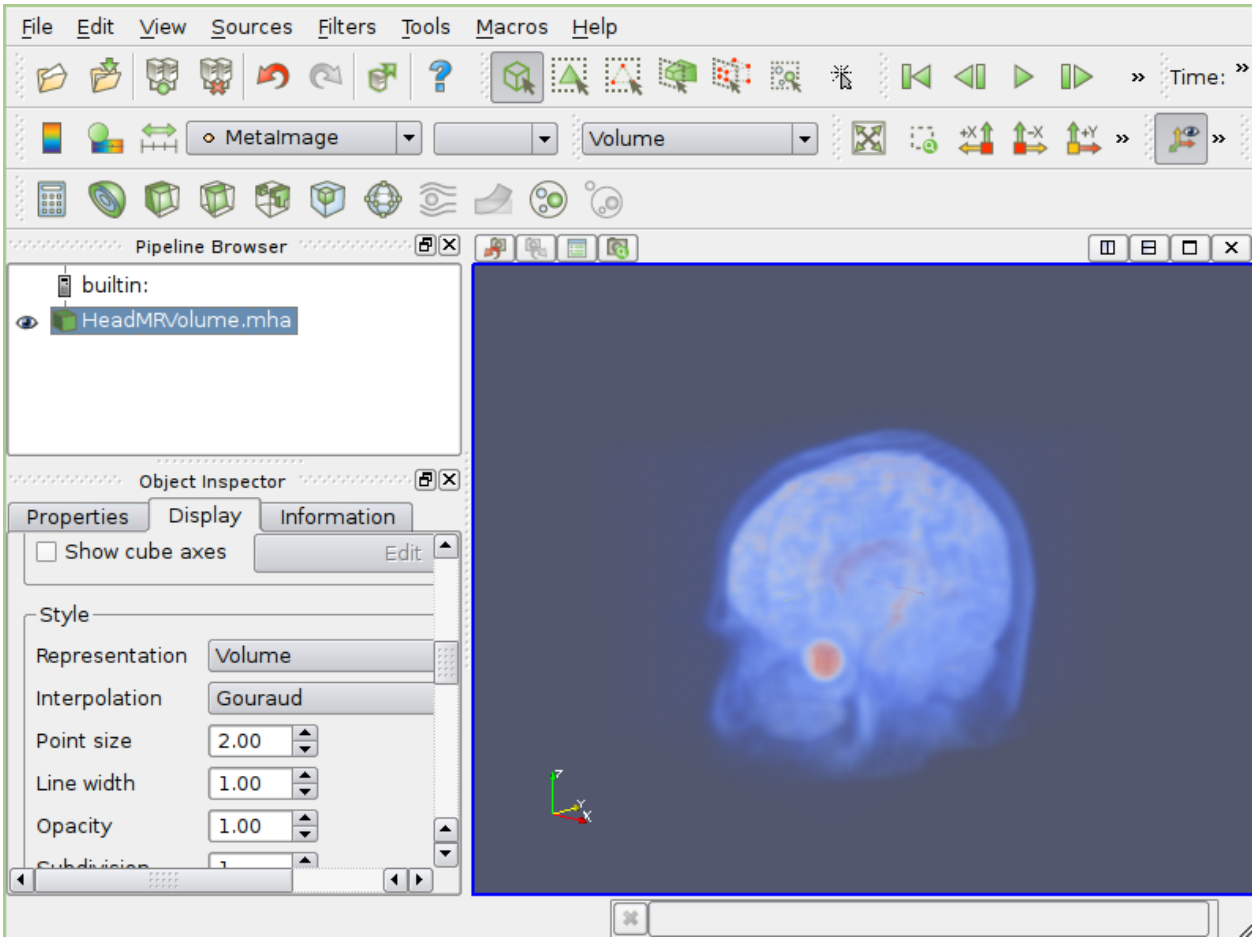



Fig. 6: Paraview

**VV** VV is an image viewer designed for fast and simple visualization of spatio-temporal images: 2D, 2D+t, 3D and 3D+t (or 4D) images.

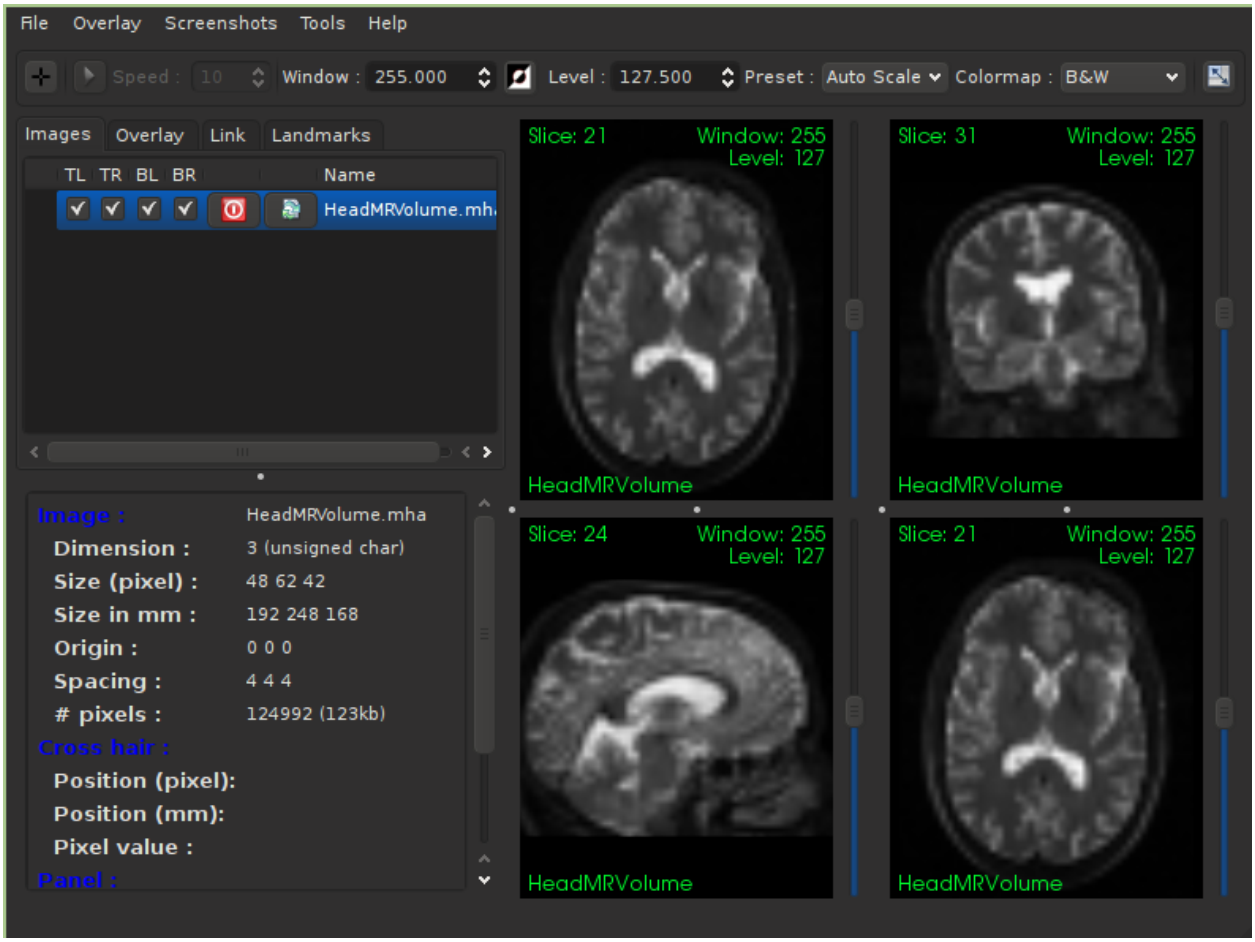

File
Overlay
Screenshots
Tools
Help
Speed : 10
Window : 255.000
Level : 127.500
Preset : Auto Scale
Colormap : B&W
Images
Overlay
Link
Landmarks
TL TR BL BR
Name
HeadMRVolume.mh
Slice: 21
Window: 255
Level: 127
HeadMRVolume
Slice: 31
Window: 255
Level: 127
HeadMRVolume
Image :
HeadMRVolume.mha
Dimension :
3 (unsigned char)
Size (pixel) :
48 62 42
Size in mm :
192 248 168
Origin :
0 0 0
Spacing :
4 4 4
# pixels :
124992 (123kb)
Cross hair :
Position (pixel):
Position (mm):
Pixel value :
Panel :
Slice: 24
Window: 255
Level: 127
HeadMRVolume
Slice: 21
Window: 255
Level: 127
HeadMRVolume

Fig. 7: VV

CHAPTER

TWO

# CONTRIBUTE

The ITK Sphinx Examples are stored in the Git distributed version control system, and the mainline repository can be found at ITK Sphinx Examples repository. Files in the repository can be edited via the typical *text editor / version control workflow* used when modifying source code.

Examples code can be written in the C++ or Python programming languages. Documentation for the examples is written in reStructuredText / Sphinx. Sphinx is a build system that uses the reStructuredText syntax with additional capabilies, such as multi-document projects and rendering of LaTeX-style equations. A good reference for the easy-to-learn reStructuredText syntax is the Quick reStructuredText guide.

There are a variety of ways to take part in the ITK Sphinx Examples project. All facets are important to the success of the project.

## 2.1 Contribute with Git

### 2.1.1 Clone the repository and setup for development

The entire source tree can be downloaded with Git:

```
git clone --recursive https://github.com/InsightSoftwareConsortium/ITKSphinxExamples.
↪git ITKEx
```

**Note:** Due to path length limitations on Windows, it is recommended to clone into a short directory like *ITKEx*.

After cloning the repository, it can be configured for development (confirm proper Git configuration, setup Git hooks, configure your GitHub account) with

```
cd ITKEx
./Utilities/SetupForDevelopment.sh
```

### 2.1.2 Modifying an existing example

First, create a local branch:

```
git checkout -b Modification
```

Modify existing example (source code and/or corresponding documentation). Add files which are missing

```
git add missing_files
```

Once you are done, commit your changes as follows:

```
git commit -a
```

To submit your changes to GitHub:

```
git review-push
```

### 2.1.3 Create a new example

To create a new example, first install Cookiecutter:

```
python -m pip install cookiecutter titlecase
```

and make use of the Python script located in the binary tree (*/path/to/ITKSphinxExamples-build/Utilities*):

```
cd /path/to/ITKSphinxExamples-build/Utilities
python ./CreateNewExample.py
```

The script will ask for an *example_name* and a *class_name*. The example_name would be *ComputeTumorIntensityVariation* for an example that answers the question "How do I compute tumor intensity variation?" The class_name indicates the itk class that is the focus of the example, such as *itk::Statistics::MaskedImageToHistogramFilter*.

*This script will generate *.cxx, *.py, and *.rst files to be modified.*

Note that the generated files are then located based on the group, module and class name to be demonstrated. For instance an example which would demonstrate the usage of itk::Image will be generated in `src/Core/Common/`.

Please do not add images directly to the repository. Instead, use the *content link system*.

## 2.2 How to write a new example

In this section we describe how to write an example by walking through the sections of an example's reStructuredText file. The mechanics of submitting an example via Git are *covered in other sections*.

### 2.2.1 General guidelines

All lines should be wrapped to 80 characters or less. This improves readability in text editors. It is generally ITK's policy to format code lines without wrapping up to 200 characters, but the 80 character limit keeps code readable in the PDF output.

Please remove all trailing whitespace. Use two newlines between section headings.

Follow ITK conventions for coding; do not use TABs, and use two spacing for indentation.

The characters used to define reStructuredText sections are as follows:

```
Title
=====

Section
-------

Subsection
..........
```

### 2.2.2 Directory placement

The examples live inside the *src* directory of the repository. They are organized by the dominant class demonstrated in the example. There are two directories of organization which mirror the directory layout in ITKv4, *Group* followed by *Module*. Therefore, place the example in the directory corresponding to the Module of one of the prominant classes used in the example.

For instance, an examples that demonstrates the *BinaryDilateImageFilter* would be placed in the *src/Filtering/BinaryMathematicalMorphology/* directory.

Note that when adding new examples, both *CMakeLists.txt* and *index.rst* files for the tree should be updated (to enforce the examples to be compiled, tested and to appear in the sphinx documentation). For this reason, we recommend to use the provided python script *CreateNewExample.py* see *Create a new example*

### 2.2.3 Title

The ITK Sphinx Examples are intended to be a set of cookbook recipes that concisely demonstrate how to achieve a single objective. In general, they are intended to provide an answer to the question, "How do I . . . ?" The title for an example can be derived by placing the example in the question format; we call this the Trebek Title Method.

For instance, an example that answers the question, *"How do I dilate a binary image?"* would be called *Dilate a binary image*. This determines the title in the reStructuredText file:

```
Dilate a binary image
=====================
```

and it also determines the name of the directory, file, and test associated with the example. These names come from a CamelCase conversion of the title. For instance, in this case, the corresponding CamelCase title is *DilateABinaryImage*. All files for the examples are placed in a directory called *DilateABinaryImage*, the C++ code is placed in *DilateABinaryImage.cxx*, Python code in *DilateABinaryImage.py*, reStructuredText documentation in *DilateABinaryImage.rst*, and the test will be named *DilateABinaryImageTest*.

### 2.2.4 Index entries

Index entries follow the title entry. They are entered with the Sphinx index directive; see that documentation for more information on how to specify index entries. Important classes for the example or terms associated with the example should be *single* entries. Term index entries should be lower case to distinguish them from classes or methods in the index. Terms with sub-terms or classes with important methods demonstrated should be used as *pair* entries. Use *seealso* instead of *see* to add a *see also* entry. For instance, with our dilate a binary image example:

```
.. index::
  single: BinaryBallStructuringElement
  single: BinaryDilateImageFilter
  pair: mathematical morphology; dilation
  pair: BinaryBallStructuringElement; SetRadius

.. seealso:: dilation; erosion
```

### 2.2.5 Synopsis

This section contains a short summary of the example.

### 2.2.6 Code

This section sources the relevant code for inclusion in the documentation with syntax highlighting care of Pygments. The literalinclude directive performs this function. This ensures that all the code that is documented is also tested. Each program should be included in a subsection corresponding to its language; one of *C++*, *Python*, or *Java*. For instance:

```
C++
...

.. literalinclude:: DilateABinaryImage.cxx
```

Follow ITK style guidelines when writing code. Keep the example as simple as possible, and follow the best practices layed out in the rest of the examples. Do basic argument checking and parsing. Wrap *Update()* calls in a *try / catch* for C++ code, etc.

### 2.2.7 Results

Include images or text output here that results from the example.

#### Images

If there was an input image, display it for reference. The images displayed here should be rendered in the PNG format for display either by directly outputing to PNG format or by rendering with your favorite *visualization application* and saving a screenshot. Display the image with the figure directive. Provide alt text with the *:alt:* option and a brief descriptive caption. For instance:

```
.. figure:: DilateABinaryImageOutputBaseline.png
  :scale: 50%
  :alt: Dilated output.

  Dilated output.
```

#### Text

Text output should be placed in a literal block by inserting two colons followed by indentation. For instance:

```
::

  Registration done !
  Number of iterations = 27
  Translation along X  = 25.0966
  Translation along Y  = 22.3275
  Optimal metric value = 4597.96
```

#### PolyData

For data structures rendered as a PolyData, such as meshes, a screenshot of the input and output rendering is insightful and motivating. The screenshot can be rendered with your favorite visualization, then included like the image renderings per above.

As a supplement to the renderings, an interactive 3D WebGL can be included in HTML output. This can be produced with the *–webgl* flag to the VTK Python script in the repository at *Utilities/Visualization/VTKPolyData.py*. In recent releases of Paraview, it can be produced by clicking *File*, *Export Scene*, *WEBGL files*. In the reStructuredText file, add:

```
.. raw:: html

  <div class="figure">
    <iframe src="InputMesh.html" width="200" height="225" seamless></iframe>
    <p class="caption">Interactive input mesh</p>
  </div>
```

### 2.2.8 Classes demonstrated

At the end of the example, provide quick reference and a link to the doxygen documentation for the important classes or structs used in the example. To do this, use the `breathelink` directive for C++ classes or `breathelinkstruct` directive for C++ structs, as follows:

```
.. breathelink:: itk::BinaryDilateImageFilter
.. breathelinkstruct:: itk::Index
```

Note that `breathelink` and `breathelinkstruct` are custom directives which make the use of:

- the doxygenclass directive or doxygenstruct directive provided by Breathe with the *:no-link:* option.
- the doxylink directive which provides a link to the full doxygen documentation with an external hyperlink target. Note that Doxygen URL's follow a predictable pattern.

## 2.3 Upload Binary Data

### 2.3.1 Motivation

Since every local Git repository contains a copy of the entire project history, it is important to avoid adding large binary files directly to the repository. Large binary files added and removed throughout a project's history will cause the repository to become bloated, take up too much disk space, require excessive time and bandwidth to download, etc.

A solution to this problem which has been adopted by this project is to store binary files, such as images, in a separate location outside the Git repository, then download the files at build time with CMake.

A "content link" file contains an identifying SHA512 hash. The content link is stored in the Git repository at the path where the file would exist, but with a ".sha512" extension appended to the file name. CMake will find these content link files at *build* time, download them from a list of server resources, and create symlinks or copies of the original files at the corresponding location in the *build tree*.

### 2.3.2 Prerequisites

The data.kitware.com server is an ITK community resource where any community member can upload binary data files. There are two methods available to upload data files:

1. The Girder web interface.
2. The *girder-cli* command line executable that comes with the girder-client Python package.

Before uploading data, please visit data.kitware.com and register for an account.

Once files have been uploaded to your account, they will be publicly available and accessible since data is content addressed. At release time, the release manager will upload and archive repository data references in the ITK collection and other redundant storage locations.

### 2.3.3 Upload Via the Web Interface

Next, proceed to *Download the Content Link*.

### 2.3.4 Upload Via Python Script

A Python script to upload files from the command line, *girder-cli*, is available with the girder-client python package. To install it:

```
python -m pip install girder-client
```

To upload files with the *girder-cli* script, we need to obtain an API key and a parent folder id from the web interface.

Use both the API key and the folder ID when calling *girder-cli*. For example,

```
girder-cli \
  --api-key 12345ALongSetOfCharactersAndNumbers \
  --api-url https://data.kitware.com/api/v1 \
  upload \
  58becaee8d777f0aefede556 \
  /tmp/cthead1.png
```

Next, proceed to *Download the Content Link*.

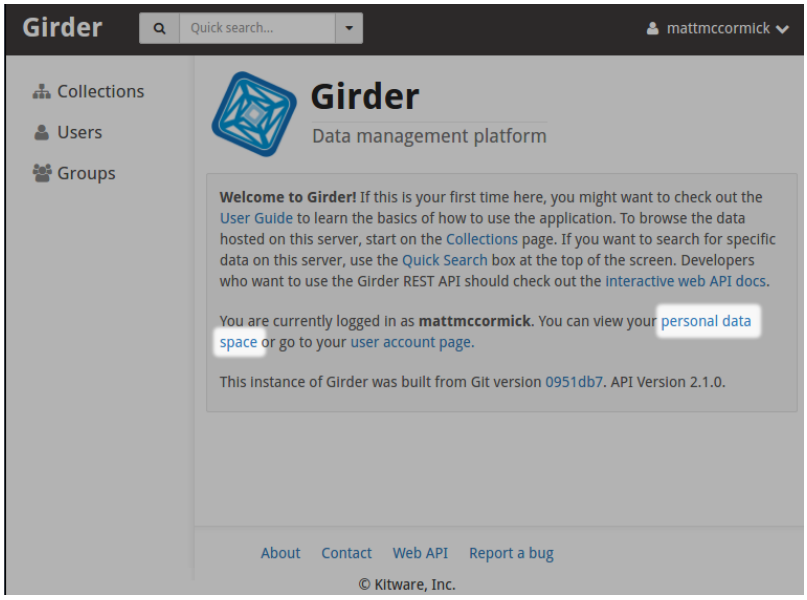



Fig. 1: After logging in, you will be presented with the welcome page. Click on the **personal data space** link.

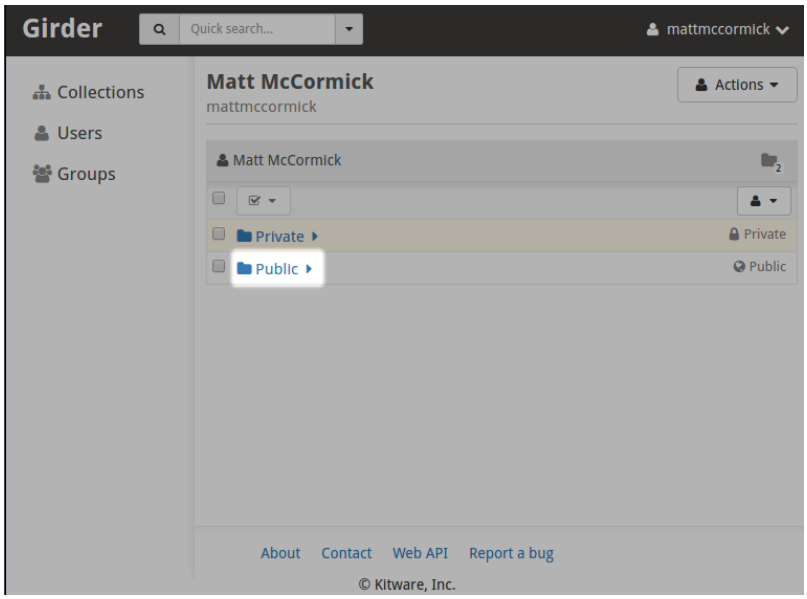



Fig. 2: Next, select the **Public** folder of your personal data space.

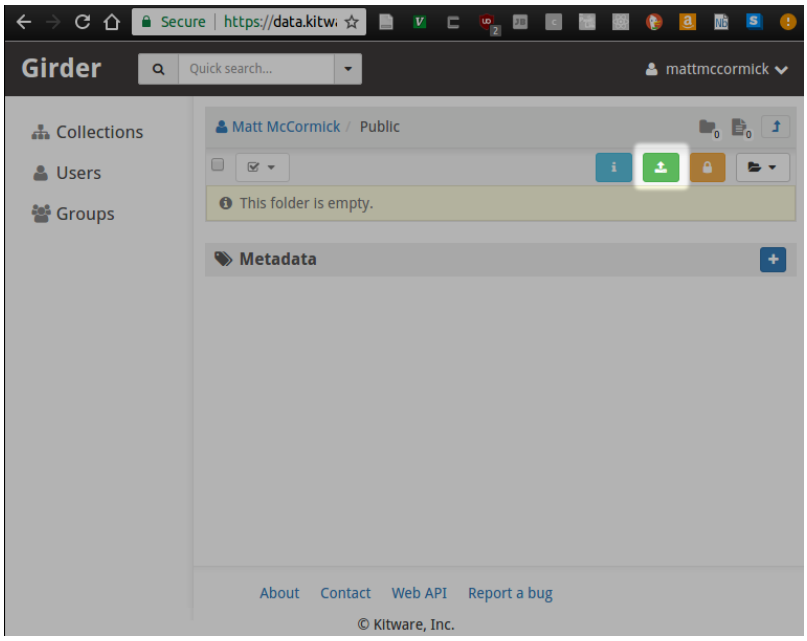



Fig. 3: Click the **green upload button**.

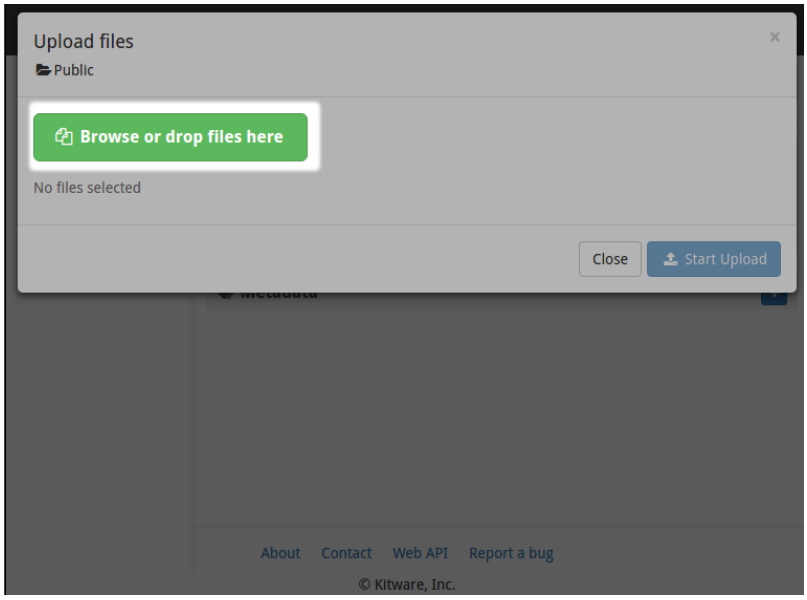



Fig. 4: Click the **Browse or drop files** to select the files to upload.

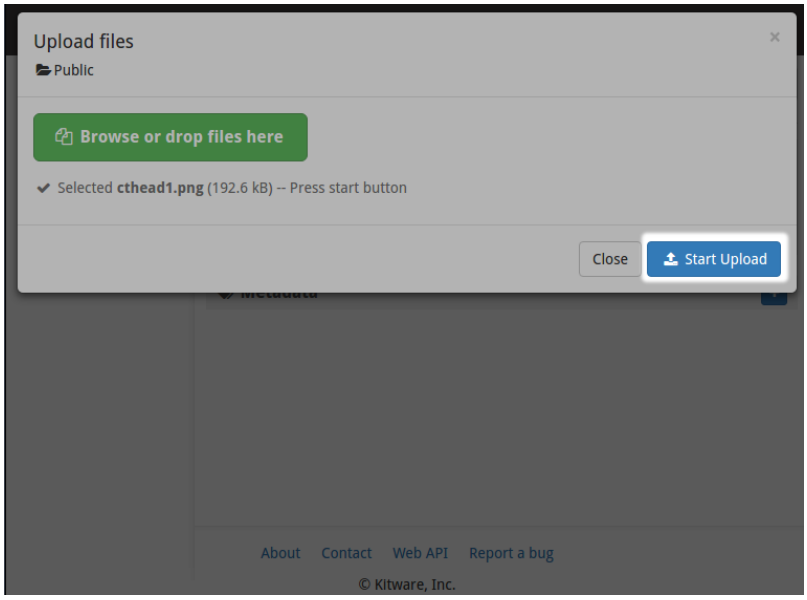



Fig. 5: Click **Start Upload** to upload the file to the server.

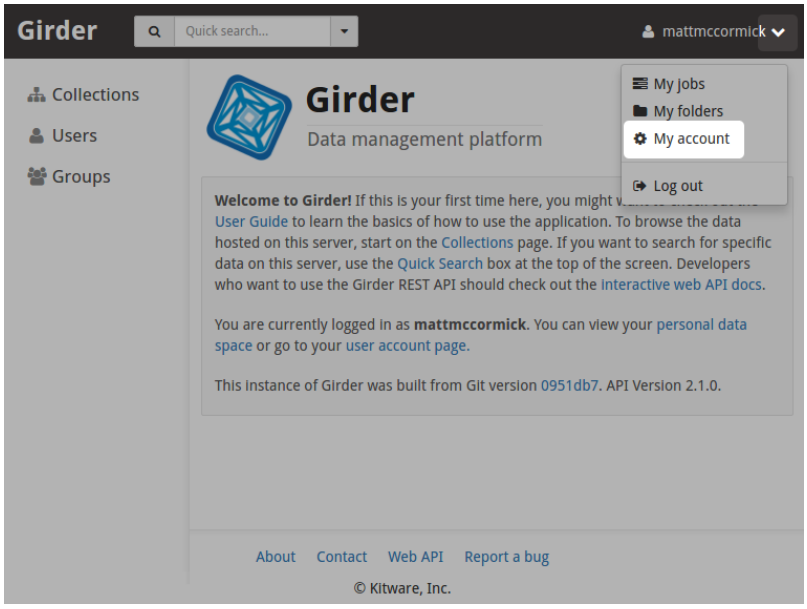



Fig. 6: After logging in, select **My account** from the user drop down.

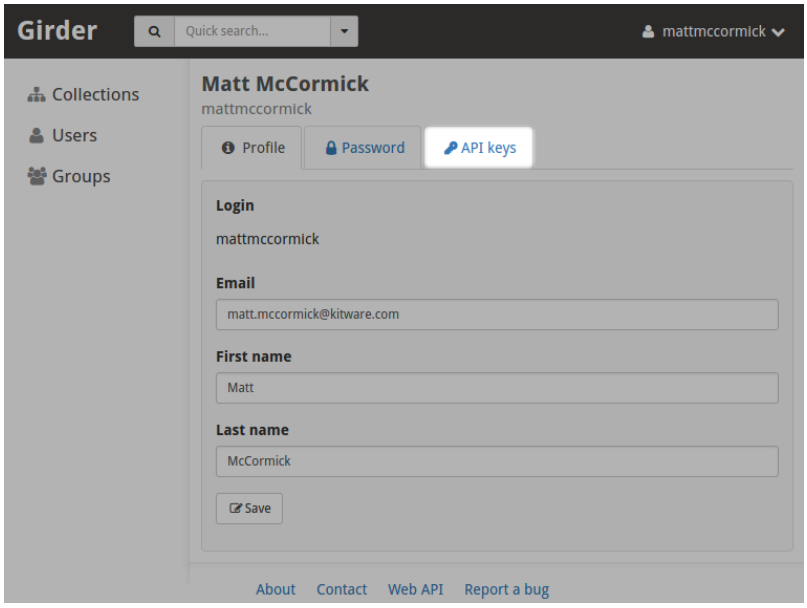



Fig. 7: Next, select the **API keys** tab.

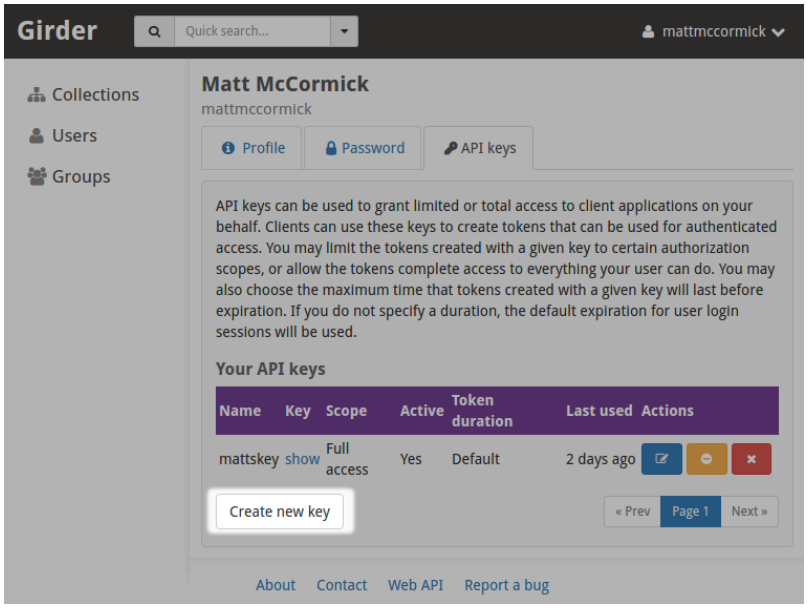



Fig. 8: Create a new API key if one is not available.

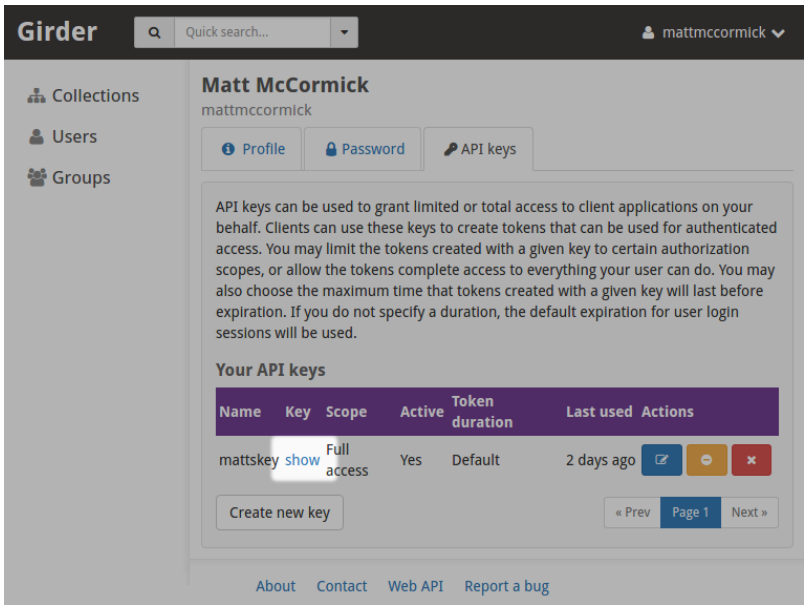



Fig. 9: The **show** link will show the key, which can be copied into the command line.

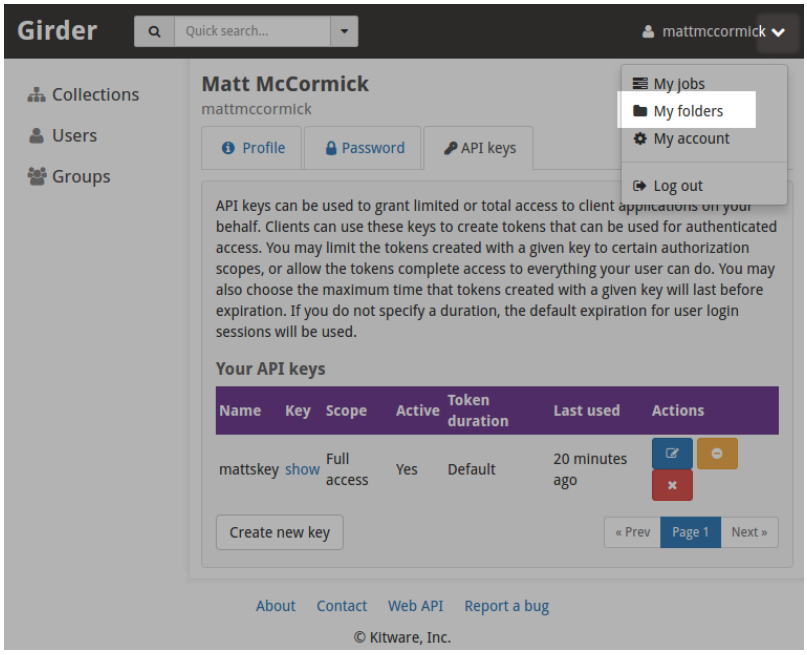



Fig. 10: Next, select **My Folders** from the user drop down.

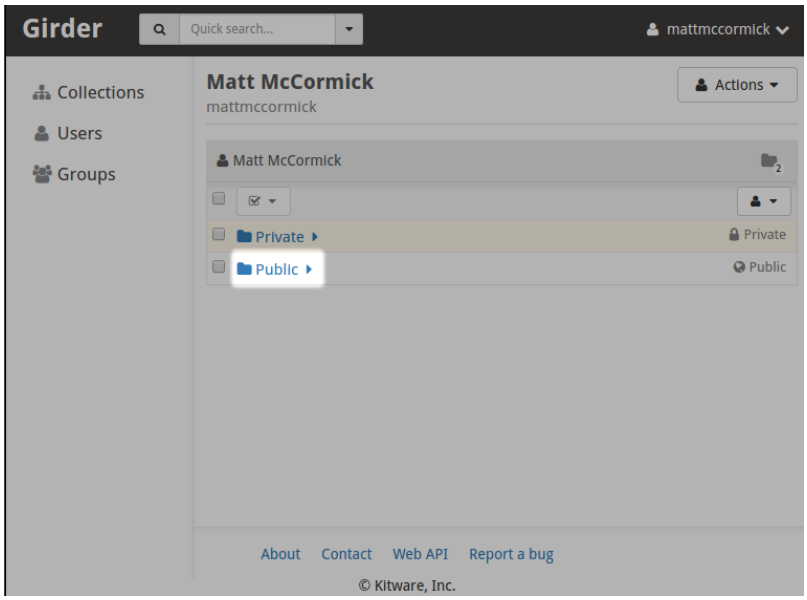



Fig. 11: Next, select the **Public** folder of your personal data space.

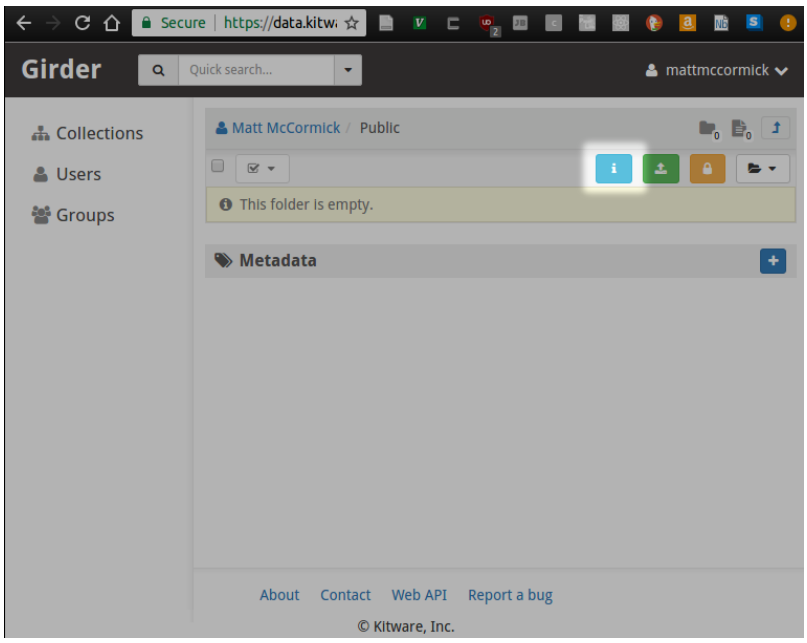



Fig. 12: Click the **i** button for information about the folder.

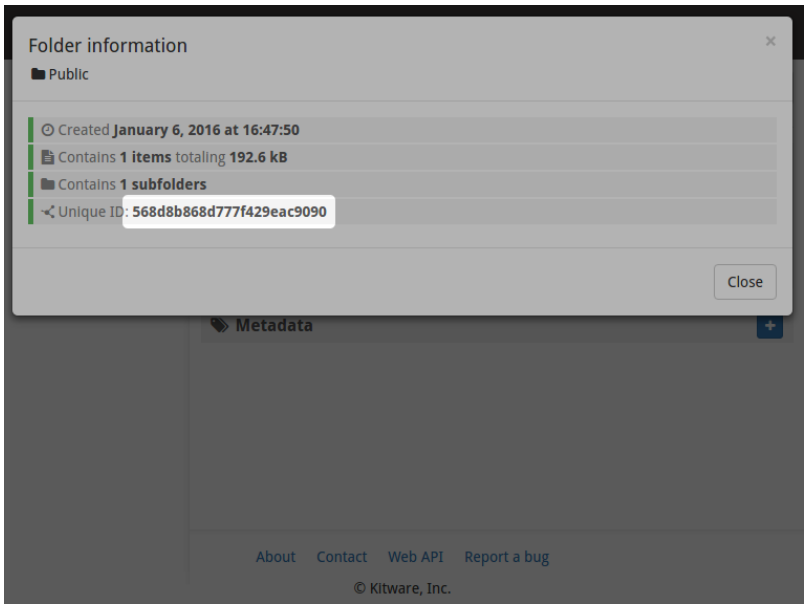



Fig. 13: The **Unique ID** can be copied into the command line.

### 2.3.5 Download the Content Link

Move the content link file to the source tree at the location where the actual file is desired in the build tree. Stage the new file to your commit:

```
git add -- path/to/file.sha512
```

## 2.4 Peer review with GitHub

GitHub code reviews are a method to vastly improve the quality of code, train new community members, and learn from other community members. In this section, we describe how to perform a pull request review.

### 2.4.1 Peer review duty

All community members are encouraged to perform reviews. If possible, all code that is merged should be reviewed and marked *Approved* by at least one other community member prior to merging. In order for this system to function smoothly, community members must perform more reviews than submit patches. Any community member can register for a GitHub account and perform a review.

Please review any patch of interest open for review on GitHub; it is not necessary for the author to request your review.

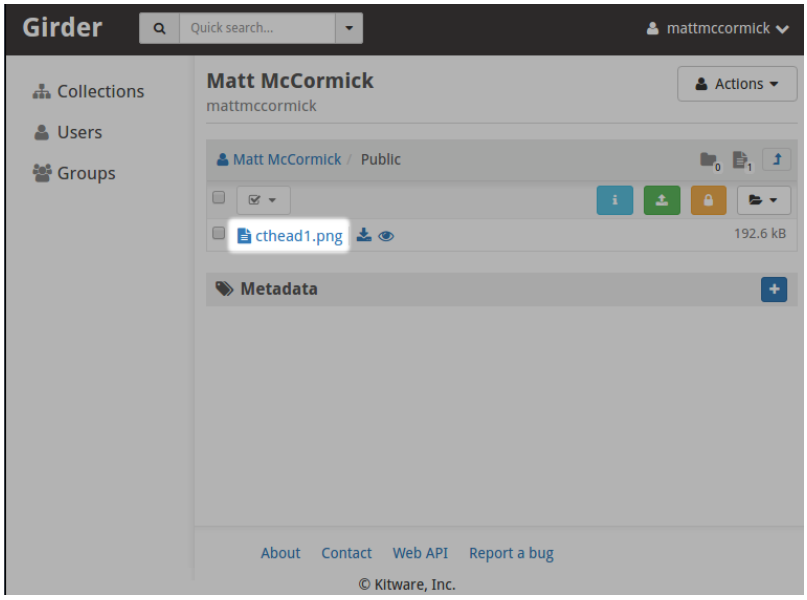



Fig. 14: Click on the file that has been uploaded.

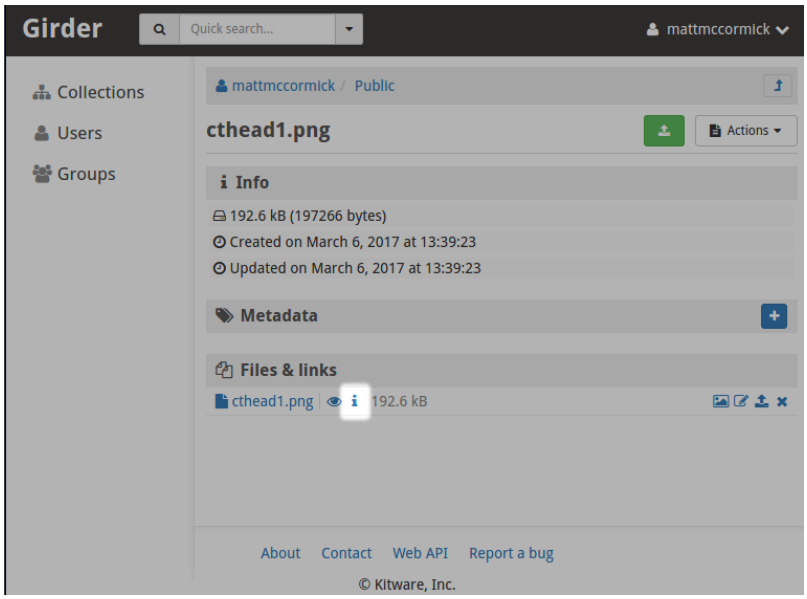



Fig. 15: Click on the **i** button for further information.

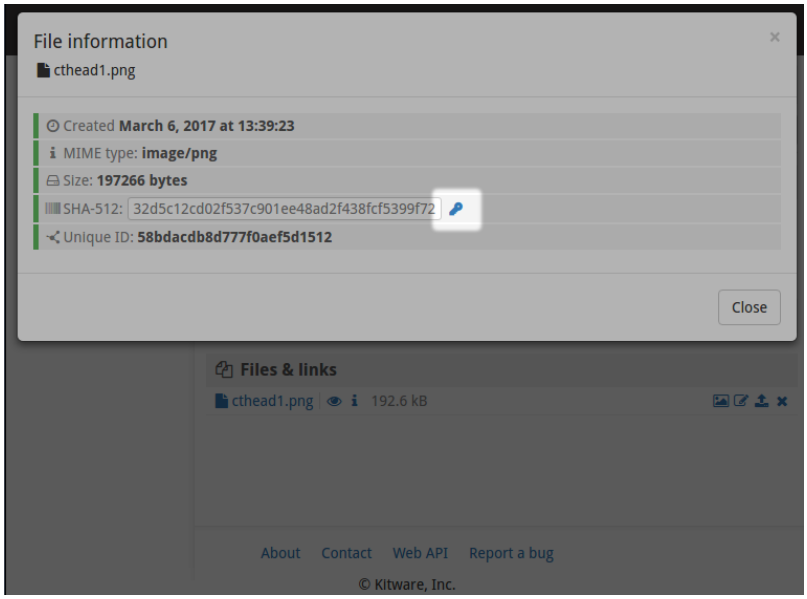



Fig. 16: Finally, click on the **Download key file** icon to download the key file.

### 2.4.2 Review comments

To start a review, open the pull request, then click on an individual commit. Or, view all files changed in the topic branch by clicking *Files changed*.

When reviewing files, inline comments can be entered by clicking on a the + button that pops up when hovering with a mouse to the right of the source code line numbers. When creating the first comment, select *Add a single comment*, which will send an email per comment, or select *Start a review*, which sends all inline comments along with the summary comment in a single email to the author.

After all inline comments have been made, click the *Review changes* button, optionally enter a summary comment, and optionally mark the pull request with *Approve* or *Request changes*. If inline comments were entered, please provide an indication in the overall review comment with a phrase like, "Please see inline comments."

### 2.4.3 Review ratings

***Approve*** Everything looks good – ready to merge.

***Comment*** The code was not examined or a neutral remark was made.

***Request changes*** Some issues were identified in the code that should be addressed.

## 2.5 Submit results to CDash

Quality control of the examples is maintained with a dashboard that collects configuration, build, and testings results in a CDash dashboard. Examples are compiled nightly, executed and compared to baseline images to ensure these examples remain valid on all supported platforms and to ensure changes in the development of ITK do not adversely effect these examples.

### 2.5.1 Experimental submission

To submit test results after *building all the examples*, simply run:

```
ctest -D Experimental
```

at the top of the build tree.

### 2.5.2 Nightly submission

You can volunteer to submit nightly testing results that ensure repository activity during the day will allow breakage on your system to go unnoticed.

#### Step 1: Setup the build directory

You will need a directory with sufficient disk space to hold the project source code, build tree, and testing scripts. This can be placed anywhere on your system where you have write access:

```
mkdir ~/dashboards
cd ~/dashboards
```

Copy the testing support script into your dashboard directory.

```
cp ~/src/ITKSphinxExamples/Utilities/Dashboard/itkexamples_common.cmake .
```

#### Step 2: Setup a script describing your system

We will write a CTest script that will update the repository, configure, build, and test the project, and then upload the results to the CDash server. Most of work in this script will be performed by *itkexamples_common.cmake*, but we need to configure the settings unique to our system.

Create a file, *itkexamples_dashboard.cmake*, to hold these settings:

```
cp ~/src/ITKSphinxExamples/Utilities/Dashboard/itkexamples_dashboard.cmake.example␣
↪itkexamples_dashboard.cmake
```

You *must* edit this file to describe your local system.

```
# An identifier for this computer that shows up on the dashboard.
set(CTEST_SITE "uberbox")

# A description of the build that identifies it on the dashboard.
# Usually it should contain the platform and a compiler description.
# Please end the name with "Examples".
set(CTEST_BUILD_NAME "Linux GCC Examples")

# This will set the CMAKE_BUILD_TYPE.
set(CTEST_BUILD_CONFIGURATION Debug)

# The Generator for CMake.
set(CTEST_CMAKE_GENERATOR "Unix Makefiles")

# This is the directory where the source and build trees will be placed.
get_filename_component(CTEST_DASHBOARD_ROOT "${CTEST_SCRIPT_DIRECTORY}/dashboard_tests
↪" ABSOLUTE)
```

(continues on next page)

(continued from previous page)

```
# You can set initial variables for the CMakeCache.
#set(dashboard_cache
#"
#ITK_DIR:PATH=/var/coolstuff/ITK_build
#BUILD_DOCUMENTATION:BOOL=ON
#"
#)

# Finally, include the rest of the script that will download, build, and submit the
# results.  For more configuration variables, see the comments at the top of
# this script.
#set(dashboard_superbuild 1)
include(${CTEST_SCRIPT_DIRECTORY}/itkexamples_common.cmake)
```

### Step 3: Configure your system to run the script on a daily basis

#### Linux or Mac

First, test the script to ensure that it runs correctly:

```
.. code-block:: bash
```

ctest -S ./itkexamples_dashboard.cmake -V -A

Did everything work? Do you see your build on the ITKSphinxExamples dashboard? If so, continue to setup a cronjob that will run the script daily. Edit your crontab:

```
.. code-block:: bash
```

crontab -e

The following line will run the script every day at 3 AM:

```
.. code-block:: bash
```

0 3 * * * ctest -S /home/luis/dashboards/itkexamples_dashboard.cmake -A &> /dev/null

#### Windows

In windows, the scheduler can be accessed by entering the *Schedule tasks* program in the Control Panel.

It may be useful to put all the nightly dashboards in a batch script and run the batch script daily. When creating the new task, the Action associate with the task is to *Start a program*. This script could be placed at:

```
.. code-block:: bash
```

C:Dashboardsrun_dashboard.bat

where *run_dashboard.bat* contains:

```
.. code-block:: bash
```

call ctest -S C:Dashboardsitkexamples_dashboard.cmake

To schedule your Windows box to reboot nightly, set up a task that runs:

```
.. code-block:: bash
```

    C:WindowsSystem32shutdown.exe /r /t 60 /f

If you're automatically rebooting your computer every night, you also need to make sure the computer automatically logs in as the dashboard user when it boots back up. To do this

1. Enter *control userpasswords2* in the *Start*, *Run...* menu entry or a *cmd.exe* shell.
2. Select the account of the user you want to automatically login.
3. Uncheck the box labelled *Users must enter a user name and password to use this computer*
4. Click OK.

Thank you for contributing to the quality of the ITK Sphinx Examples!

CHAPTER

THREE

# EXAMPLES

## 3.1 Bridge

### 3.1.1 VtkGlue

#### Convert an itk::Image to vtkImageData

#### Synopsis

Convert an itk::Image to vtkImageData in a pipeline. This transfers the image buffer data along with image size, pixel spacing, and origin. Since *vtkImageData* does not yet support an orientation matrix, the direction cosines are lost. This requires *Module_ITKVtkGlue* to be turned on in ITK's CMake configuration.

#### Results

Output:

```
vtkImageData (0x51828b0)
  Debug: Off
  Modified Time: 240
  Reference Count: 2
  Registered Events: (none)
  Information: 0x35d7810
  Data Released: False
  Global Release Data: Off
  UpdateTime: 241
  Field Data:
    Debug: Off
    Modified Time: 186
    Reference Count: 1
    Registered Events: (none)
    Number Of Arrays: 0
    Number Of Components: 0
    Number Of Tuples: 0
  Number Of Points: 39277
  Number Of Cells: 38880
  Cell Data:
    Debug: Off
    Modified Time: 194
    Reference Count: 1
    Registered Events: (none)
```

(continues on next page)

(continued from previous page)

```
    Number Of Arrays: 0
    Number Of Components: 0
    Number Of Tuples: 0
    Copy Tuple Flags: ( 1 1 1 1 1 0 1 1 )
    Interpolate Flags: ( 1 1 1 1 1 0 0 1 )
    Pass Through Flags: ( 1 1 1 1 1 1 1 1 )
    Scalars: (none)
    Vectors: (none)
    Normals: (none)
    TCoords: (none)
    Tensors: (none)
    GlobalIds: (none)
    PedigreeIds: (none)
    EdgeFlag: (none)
  Point Data:
    Debug: Off
    Modified Time: 240
    Reference Count: 1
    Registered Events: (none)
    Number Of Arrays: 1
    Array 0 name = scalars
    Number Of Components: 1
    Number Of Tuples: 39277
    Copy Tuple Flags: ( 1 1 1 1 1 0 1 1 )
    Interpolate Flags: ( 1 1 1 1 1 0 0 1 )
    Pass Through Flags: ( 1 1 1 1 1 1 1 1 )
    Scalars:
      Debug: Off
      Modified Time: 240
      Reference Count: 1
      Registered Events: (none)
      Name: scalars
      Data type: unsigned char
      Size: 39277
      MaxId: 39276
      NumberOfComponents: 1
      Information: 0
      Name: scalars
      Number Of Components: 1
      Number Of Tuples: 39277
      Size: 39277
      MaxId: 39276
      LookupTable: (none)
      Array: 0x51a7150
    Vectors: (none)
    Normals: (none)
    TCoords: (none)
    Tensors: (none)
    GlobalIds: (none)
    PedigreeIds: (none)
    EdgeFlag: (none)
  Bounds:
    Xmin,Xmax: (0, 180)
    Ymin,Ymax: (0, 216)
    Zmin,Zmax: (0, 0)
  Compute Time: 254
  Spacing: (1, 1, 1)
```

(continues on next page)

(continued from previous page)

```
  Origin: (0, 0, 0)
  Dimensions: (181, 217, 1)
  Increments: (0, 0, 0)
  Extent: (0, 180, 0, 216, 0, 0)
```

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("5.2.0"):
    print("ITK 5.2.0 is required.")
    sys.exit(1)

parser = argparse.ArgumentParser(description="Convert An itk Image to vtk Image Data.
↪")
parser.add_argument("input_image")
args = parser.parse_args()

inputImage = itk.imread(args.input_image)

vtkImage = itk.image_to_vtk_image(inputImage)

print(vtkImage)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageToVTKImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];

  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::ImageToVTKImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  try
  {
    filter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  vtkImageData * myvtkImageData = filter->GetOutput();
  myvtkImageData->Print(std::cout);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ImageToVTKImageFilter** : **public** itk::*ProcessObject*

Converts an ITK image into a VTK image and plugs a itk data pipeline to a VTK datapipeline.

This class puts together an itkVTKImageExporter and a vtkImageImporter. It takes care of the details related to the connection of ITK and VTK pipelines. The User will perceive this filter as an adaptor to which an itk::Image can be plugged as input and a vtkImage is produced as output.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert an itk::Image to vtkImageData in a pipeline
- Convert RGB vtkImageData to an itk::Image

See itk::ImageToVTKImageFilter for additional documentation.

### Convert an RGB itk::Image to vtkImageData

### Synopsis

Convert an itk::Image with RGB pixels to vtkImageData. This transfers the image buffer data along with image size, pixel spacing, and origin. Since *vtkImageData* does not yet support an orientation matrix, the direction cosines are lost. This requires *Module_ITKVtkGlue* to be turned on in ITK's CMake configuration.

### Results

Output:

```
vtkImageData (0x35dc400)
  Debug: Off
  Modified Time: 242
  Reference Count: 2
  Registered Events: (none)
  Information: 0x35dc590
  Data Released: False
  Global Release Data: Off
  UpdateTime: 243
  Field Data:
    Debug: Off
    Modified Time: 187
    Reference Count: 1
    Registered Events: (none)
    Number Of Arrays: 0
    Number Of Components: 0
    Number Of Tuples: 0
  Number Of Points: 20574
  Number Of Cells: 20286
  Cell Data:
    Debug: Off
    Modified Time: 195
    Reference Count: 1
    Registered Events: (none)
    Number Of Arrays: 0
    Number Of Components: 0
    Number Of Tuples: 0
    Copy Tuple Flags: ( 1 1 1 1 1 0 1 1 )
    Interpolate Flags: ( 1 1 1 1 1 0 0 1 )
    Pass Through Flags: ( 1 1 1 1 1 1 1 1 )
    Scalars: (none)
    Vectors: (none)
    Normals: (none)
    TCoords: (none)
    Tensors: (none)
    GlobalIds: (none)
    PedigreeIds: (none)
    EdgeFlag: (none)
  Point Data:
    Debug: Off
    Modified Time: 242
    Reference Count: 1
    Registered Events: (none)
    Number Of Arrays: 1
```

(continues on next page)

(continued from previous page)

```
    Array 0 name = scalars
    Number Of Components: 3
    Number Of Tuples: 20574
    Copy Tuple Flags: ( 1 1 1 1 1 0 1 1 )
    Interpolate Flags: ( 1 1 1 1 1 0 0 1 )
    Pass Through Flags: ( 1 1 1 1 1 1 1 1 )
    Scalars:
      Debug: Off
      Modified Time: 242
      Reference Count: 1
      Registered Events: (none)
      Name: scalars
      Data type: unsigned char
      Size: 61722
      MaxId: 61721
      NumberOfComponents: 3
      Information: 0
      Name: scalars
      Number Of Components: 3
      Number Of Tuples: 20574
      Size: 61722
      MaxId: 61721
      LookupTable: (none)
      Array: 0x35c3430
    Vectors: (none)
    Normals: (none)
    TCoords: (none)
    Tensors: (none)
    GlobalIds: (none)
    PedigreeIds: (none)
    EdgeFlag: (none)
  Bounds:
    Xmin,Xmax: (0, 126)
    Ymin,Ymax: (0, 161)
    Zmin,Zmax: (0, 0)
  Compute Time: 256
  Spacing: (1, 1, 1)
  Origin: (0, 0, 0)
```

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("5.2.0"):
    print("ITK 5.2.0 is required.")
    sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
parser = argparse.ArgumentParser(
    description="Convert An RBG itk Image to vtk Image Data."
)
parser.add_argument("input_image")
args = parser.parse_args()

inputImage = itk.imread(args.input_image)

vtkImage = itk.image_to_vtk_image(inputImage)

print(vtkImage)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageToVTKImageFilter.h"
#include "itkRGBPixel.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];

  constexpr unsigned int Dimension = 2;

  using PixelComponentType = unsigned char;
  using PixelType = itk::RGBPixel<PixelComponentType>;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::ImageToVTKImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  try
  {
    filter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  vtkImageData * myvtkImageData = filter->GetOutput();
  myvtkImageData->Print(std::cout);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ImageToVTKImageFilter** : **public** itk::*ProcessObject*

Converts an ITK image into a VTK image and plugs a itk data pipeline to a VTK datapipeline.

This class puts together an itkVTKImageExporter and a vtkImageImporter. It takes care of the details related to the connection of ITK and VTK pipelines. The User will perceive this filter as an adaptor to which an itk::Image can be plugged as input and a vtkImage is produced as output.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert an itk::Image to vtkImageData in a pipeline
- Convert RGB vtkImageData to an itk::Image

See itk::ImageToVTKImageFilter for additional documentation.

### Convert RGB vtkImageData to an itk::Image

### Synopsis

Convert RGB vtkImageData to an itk::Image in a processing pipeline. This transfers the image buffer data along with image size, pixel spacing, and origin. Since *vtkImageData* does not yet support an orientation matrix, the direction cosines are lost. This requires *Module_ITKVtkGlue* to be turned on in ITK's CMake configuration.

### Results

Output:

```
Image (0x2dc70e0)
  RTTI typeinfo:   itk::Image<itk::RGBPixel<unsigned char>, 2u>
  Reference Count: 2
  Modified Time: 28
  Debug: Off
  Object Name:
  Observers:
    none
  Source: (0x3281a40)
  Source output name: Primary
```

(continues on next page)

(continued from previous page)

```
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 23
  UpdateMTime: 29
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [127, 162]
  BufferedRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [127, 162]
  RequestedRegion:
    Dimension: 2
    Index: [0, 0]
    Size: [127, 162]
  Spacing: [1, 1]
  Origin: [0, 0]
  Direction:
1 0
0 1

  IndexToPointMatrix:
1 0
0 1

  PointToIndexMatrix:
1 0
0 1

  Inverse Direction:
1 0
0 1

  PixelContainer:
    ImportImageContainer (0x3282c00)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, itk::RGBPixel
↪<unsigned char> >
      Reference Count: 1
      Modified Time: 27
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 0x3196130
      Container manages memory: false
      Size: 20574
      Capacity: 20574
```

### Code

### Python

```python
#!/usr/bin/env python

import itk
import vtk
import argparse

parser = argparse.ArgumentParser(
    description="Convert RBG vtk Image Data To An itk Image."
)
parser.add_argument("input_image")
args = parser.parse_args()

Dimension = 2
PixelComponentType = itk.UC
PixelType = itk.RGBPixel[PixelComponentType]
ImageType = itk.Image[PixelType, Dimension]

reader = vtk.vtkPNGReader()
reader.SetFileName(args.input_image)
reader.SetDataScalarTypeToUnsignedChar()

vtkToItkFilter = itk.VTKImageToImageFilter[ImageType].New()
vtkToItkFilter.SetInput(reader.GetOutput())

reader.Update()
vtkToItkFilter.Update()
myitkImage = vtkToItkFilter.GetOutput()
print(myitkImage)
```

### C++

```cpp
#include "vtkSmartPointer.h"
#include "vtkPNGReader.h"
#include "itkVTKImageToImageFilter.h"
#include "itkRGBPixel.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Dimension = 2;

  using PixelComponentType = unsigned char;
  using PixelType = itk::RGBPixel<PixelComponentType>;
  using ImageType = itk::Image<PixelType, Dimension>;

  vtkSmartPointer<vtkPNGReader> reader = vtkSmartPointer<vtkPNGReader>::New();
  reader->SetFileName(inputFileName);
  reader->SetDataScalarTypeToUnsignedChar();

  using FilterType = itk::VTKImageToImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  try
  {
    reader->Update();
    filter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  ImageType::ConstPointer myitkImage = filter->GetOutput();
  myitkImage->Print(std::cout);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**>
**class VTKImageToImageFilter** : **public** itk::VTKImageImport<*TOutputImage*>

Converts a VTK image into an ITK image and plugs a VTK data pipeline to an ITK datapipeline.

This class puts together an itk::VTKImageImport and a vtk::ImageExport. It takes care of the details related to the connection of ITK and VTK pipelines. The User will perceive this filter as an adaptor to which a vtkImage-Data can be plugged as input and an itk::Image is produced as output.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- VTK Image To ITK Image

See itk::VTKImageToImageFilter for additional documentation.

### Convert vtkImageData to an itk::Image

### Synopsis

Convert vtkImageData to an itk::Image in a processing pipeline. This transfers the image buffer data along with image size, pixel spacing, and origin. Since *vtkImageData* does not yet support an orientation matrix, the direction cosines are lost. This requires *Module_ITKVtkGlue* to be turned on in ITK's CMake configuration.

### Results

Output:

```
Image (0x3b9d810)
  RTTI typeinfo:   itk::Image<unsigned char, 3u>
  Reference Count: 2
  Modified Time: 62
  Debug: Off
  Object Name:
  Observers:
    none
  Source: (0x47e5460)
  Source output name: Primary
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 57
  UpdateMTime: 63
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [181, 217, 1]
  BufferedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [181, 217, 1]
  RequestedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [181, 217, 1]
  Spacing: [1, 1, 1]
  Origin: [0, 0, 0]
  Direction:
1 0 0
0 1 0
0 0 1

  IndexToPointMatrix:
1 0 0
0 1 0
0 0 1

  PointToIndexMatrix:
1 0 0
0 1 0
0 0 1
```

(continues on next page)

(continued from previous page)

```
  Inverse Direction:
1 0 0
0 1 0
0 0 1

  PixelContainer:
    ImportImageContainer (0x4875620)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned char>
      Reference Count: 1
      Modified Time: 61
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 0x334e260
      Container manages memory: false
      Size: 39277
      Capacity: 39277
```

### Code

### Python

```
#!/usr/bin/env python

import itk
import vtk
import argparse

parser = argparse.ArgumentParser(description="Convert vtk Image Data To An itk Image.
↪")
parser.add_argument("input_image")
args = parser.parse_args()

Dimension = 2
PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

reader = vtk.vtkPNGReader()
reader.SetFileName(args.input_image)
reader.SetDataScalarTypeToUnsignedChar()

magnitude = vtk.vtkImageMagnitude()
magnitude.SetInputConnection(reader.GetOutputPort())
magnitude.Update()

vtkToItkFilter = itk.VTKImageToImageFilter[ImageType].New()
vtkToItkFilter.SetInput(magnitude.GetOutput())

vtkToItkFilter.Update()
myitkImage = vtkToItkFilter.GetOutput()
print(myitkImage)
```

### C++

```
#include "vtkSmartPointer.h"
#include "vtkPNGReader.h"
#include "vtkImageMagnitude.h"
#include "itkVTKImageToImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  vtkSmartPointer<vtkPNGReader> reader = vtkSmartPointer<vtkPNGReader>::New();
  reader->SetFileName(inputFileName);
  reader->SetDataScalarTypeToUnsignedChar();

  vtkSmartPointer<vtkImageMagnitude> magnitude = vtkSmartPointer<vtkImageMagnitude>::
↪New();
  magnitude->SetInputConnection(reader->GetOutputPort());
  magnitude->Update();

  using FilterType = itk::VTKImageToImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(magnitude->GetOutput());

  try
  {
    filter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  ImageType::ConstPointer myitkImage = filter->GetOutput();
  myitkImage->Print(std::cout);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**>
**class VTKImageToImageFilter** : **public** itk::VTKImageImport<*TOutputImage*>
Converts a VTK image into an ITK image and plugs a VTK data pipeline to an ITK datapipeline.

This class puts together an itk::VTKImageImport and a vtk::ImageExport. It takes care of the details related to the connection of ITK and VTK pipelines. The User will perceive this filter as an adaptor to which a vtkImage-Data can be plugged as input and an itk::Image is produced as output.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- VTK Image To ITK Image

See itk::VTKImageToImageFilter for additional documentation.

### Visualize an Evolving Dense 2D Level Set as Elevation Map

### Synopsis

Visualize an evolving dense level-set function 2D rendered as an elevation map.

### Results

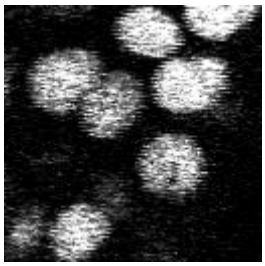


Fig. 1: Input image

### Code

### C++

```
#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkLevelSetIterationUpdateCommand.h"
#include "itkLevelSetContainer.h"
#include "itkLevelSetEquationChanAndVeseInternalTerm.h"
#include "itkLevelSetEquationChanAndVeseExternalTerm.h"
#include "itkLevelSetEquationContainer.h"
#include "itkLevelSetEquationTermContainer.h"
#include "itkLevelSetEvolution.h"
```

(continues on next page)

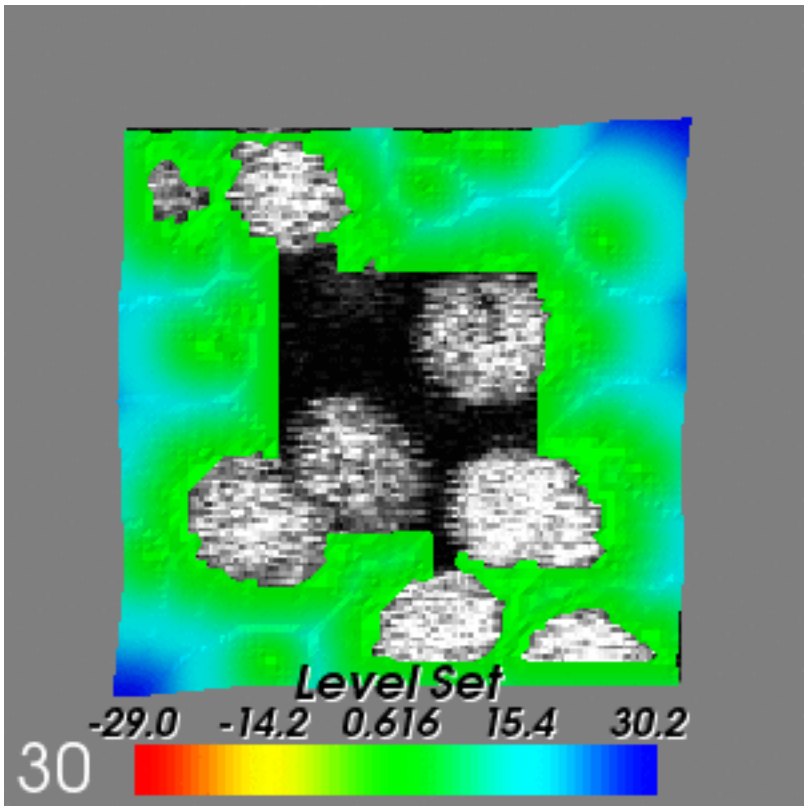



Fig. 2: Evolving level-sets

(continued from previous page)

```
#include "itkLevelSetEvolutionNumberOfIterationsStoppingCriterion.h"
#include "itkLevelSetDenseImage.h"
#include "itkVTKVisualize2DLevelSetAsElevationMap.h"
#include "itkSinRegularizedHeavisideStepFunction.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "1- Input Image" << std::endl;
    std::cerr << "2- Number of Iterations" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  InputImageType::Pointer input = reader->GetOutput();

  int numberOfIterations = std::stoi(argv[2]);

  using LevelSetPixelType = float;
  using LevelSetImageType = itk::Image<LevelSetPixelType, Dimension>;
  using LevelSetType = itk::LevelSetDenseImage<LevelSetImageType>;

  using LevelSetOutputType = LevelSetType::OutputType;
  using LevelSetRealType = LevelSetType::OutputRealType;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using BinaryImageType = itk::Image<LevelSetOutputType, Dimension>;
  BinaryImageType::Pointer binary = BinaryImageType::New();
  binary->SetRegions(input->GetLargestPossibleRegion());
  binary->CopyInformation(input);
  binary->Allocate();
  binary->FillBuffer(itk::NumericTraits<LevelSetOutputType>::Zero);

  BinaryImageType::RegionType region;
  BinaryImageType::IndexType  index;
  BinaryImageType::SizeType   size;

  index.Fill(5);
  size.Fill(120);

  region.SetIndex(index);
```

(continues on next page)

(continued from previous page)

```
  region.SetSize(size);

  using InputIteratorType = itk::ImageRegionIteratorWithIndex<BinaryImageType>;
  InputIteratorType iIt(binary, region);
  iIt.GoToBegin();
  while (!iIt.IsAtEnd())
  {
    iIt.Set(itk::NumericTraits<LevelSetOutputType>::One);
    ++iIt;
  }

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<BinaryImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(binary);
  adaptor->Initialize();
  LevelSetType::Pointer levelSet = adaptor->GetLevelSet();

  // The Heaviside function
  using HeavisideFunctionType = itk::SinRegularizedHeavisideStepFunction
↪<LevelSetRealType, LevelSetRealType>;
  HeavisideFunctionType::Pointer heaviside = HeavisideFunctionType::New();
  heaviside->SetEpsilon(1.5);

  // Create the level set container
  using LevelSetContainerType = itk::LevelSetContainer<itk::IdentifierType,␣
↪LevelSetType>;
  LevelSetContainerType::Pointer levelSetContainer = LevelSetContainerType::New();
  levelSetContainer->SetHeaviside(heaviside);
  levelSetContainer->AddLevelSet(0, levelSet);

  // Create the terms.
  //
  // // Chan and Vese internal term
  using ChanAndVeseInternalTermType =
    itk::LevelSetEquationChanAndVeseInternalTerm<InputImageType,␣
↪LevelSetContainerType>;
  ChanAndVeseInternalTermType::Pointer cvInternalTerm = ChanAndVeseInternalTermType::
↪New();
  cvInternalTerm->SetInput(input);
  cvInternalTerm->SetCoefficient(0.5);

  // // Chan and Vese external term
  using ChanAndVeseExternalTermType =
    itk::LevelSetEquationChanAndVeseExternalTerm<InputImageType,␣
↪LevelSetContainerType>;
  ChanAndVeseExternalTermType::Pointer cvExternalTerm = ChanAndVeseExternalTermType::
↪New();
  cvExternalTerm->SetInput(input);

  // Create term container (equation rhs)
  using TermContainerType = itk::LevelSetEquationTermContainer<InputImageType,␣
↪LevelSetContainerType>;
  TermContainerType::Pointer termContainer = TermContainerType::New();
  termContainer->SetLevelSetContainer(levelSetContainer);
```

(continues on next page)

(continued from previous page)

```
  termContainer->SetInput(input);
  termContainer->AddTerm(0, cvInternalTerm);
  termContainer->AddTerm(1, cvExternalTerm);

  // Create equation container
  using EquationContainerType = itk::LevelSetEquationContainer<TermContainerType>;
  EquationContainerType::Pointer equationContainer = EquationContainerType::New();
  equationContainer->SetLevelSetContainer(levelSetContainer);
  equationContainer->AddEquation(0, termContainer);

  // Create stopping criteria
  using StoppingCriterionType = itk::
↪LevelSetEvolutionNumberOfIterationsStoppingCriterion<LevelSetContainerType>;
  StoppingCriterionType::Pointer criterion = StoppingCriterionType::New();
  criterion->SetNumberOfIterations(numberOfIterations);

  // Create the visualizer
  using VisualizationType = itk::VTKVisualize2DLevelSetAsElevationMap<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);
  visualizer->SetScreenCapture(true);

  // Create evolution class
  using LevelSetEvolutionType = itk::LevelSetEvolution<EquationContainerType,␣
↪LevelSetType>;
  LevelSetEvolutionType::Pointer evolution = LevelSetEvolutionType::New();
  evolution->SetEquationContainer(equationContainer);
  evolution->SetStoppingCriterion(criterion);
  evolution->SetLevelSetContainer(levelSetContainer);

  using IterationUpdateCommandType = itk::LevelSetIterationUpdateCommand
↪<LevelSetEvolutionType, VisualizationType>;
  IterationUpdateCommandType::Pointer iterationUpdateCommand =␣
↪IterationUpdateCommandType::New();
  iterationUpdateCommand->SetFilterToUpdate(visualizer);
  iterationUpdateCommand->SetUpdatePeriod(5);

  evolution->AddObserver(itk::IterationEvent(), iterationUpdateCommand);

  evolution->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### Visualize an Evolving Dense 2D Level-Set Zero-Set

### Synopsis

Visualize the evolving zero-set of a dense level-set function 2D rendered.

### Results

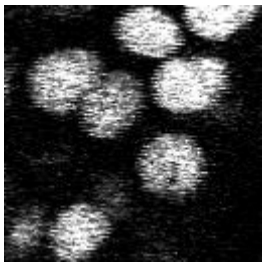


Fig. 3: Input image

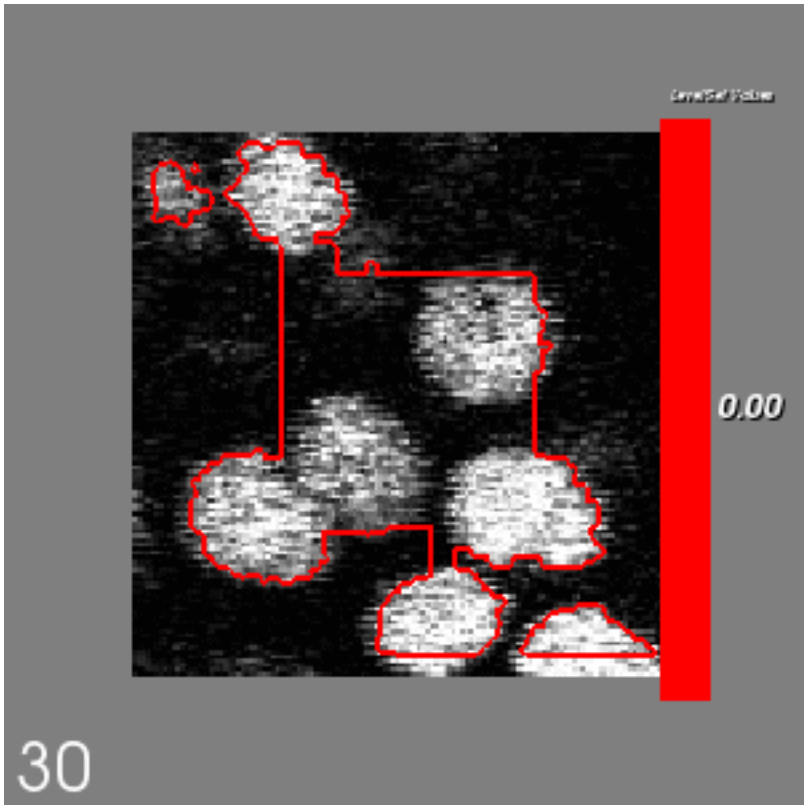



Fig. 4: Evolving level-sets

### Code

### C++

```
#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkLevelSetIterationUpdateCommand.h"
#include "itkLevelSetContainer.h"
#include "itkLevelSetEquationChanAndVeseInternalTerm.h"
#include "itkLevelSetEquationChanAndVeseExternalTerm.h"
#include "itkLevelSetEquationContainer.h"
#include "itkLevelSetEquationTermContainer.h"
#include "itkLevelSetEvolution.h"
#include "itkLevelSetEvolutionNumberOfIterationsStoppingCriterion.h"
#include "itkLevelSetDenseImage.h"
#include "itkVTKVisualizeImageLevelSetIsoValues.h"
#include "itkSinRegularizedHeavisideStepFunction.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "1- Input Image" << std::endl;
    std::cerr << "2- Number of Iterations" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  InputImageType::Pointer input = reader->GetOutput();

  int numberOfIterations = std::stoi(argv[2]);

  using LevelSetPixelType = float;
  using LevelSetImageType = itk::Image<LevelSetPixelType, Dimension>;
  using LevelSetType = itk::LevelSetDenseImage<LevelSetImageType>;

  using LevelSetOutputType = LevelSetType::OutputType;
  using LevelSetRealType = LevelSetType::OutputRealType;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using BinaryImageType = itk::Image<LevelSetOutputType, Dimension>;
  BinaryImageType::Pointer binary = BinaryImageType::New();
```

(continues on next page)

(continued from previous page)

```
  binary->SetRegions(input->GetLargestPossibleRegion());
  binary->CopyInformation(input);
  binary->Allocate();
  binary->FillBuffer(itk::NumericTraits<LevelSetOutputType>::Zero);

  BinaryImageType::RegionType region;
  BinaryImageType::IndexType  index;
  BinaryImageType::SizeType   size;

  index.Fill(5);
  size.Fill(120);

  region.SetIndex(index);
  region.SetSize(size);

  using InputIteratorType = itk::ImageRegionIteratorWithIndex<BinaryImageType>;
  InputIteratorType iIt(binary, region);
  iIt.GoToBegin();
  while (!iIt.IsAtEnd())
  {
    iIt.Set(itk::NumericTraits<LevelSetOutputType>::One);
    ++iIt;
  }

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<BinaryImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(binary);
  adaptor->Initialize();
  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // The Heaviside function
  using HeavisideFunctionType = itk::SinRegularizedHeavisideStepFunction
↪<LevelSetRealType, LevelSetRealType>;
  HeavisideFunctionType::Pointer heaviside = HeavisideFunctionType::New();
  heaviside->SetEpsilon(1.5);

  // Create the level set container
  using LevelSetContainerType = itk::LevelSetContainer<itk::IdentifierType,␣
↪LevelSetType>;
  LevelSetContainerType::Pointer levelSetContainer = LevelSetContainerType::New();
  levelSetContainer->SetHeaviside(heaviside);
  levelSetContainer->AddLevelSet(0, levelSet);

  // Create the terms.
  //
  // // Chan and Vese internal term
  using ChanAndVeseInternalTermType =
    itk::LevelSetEquationChanAndVeseInternalTerm<InputImageType,␣
↪LevelSetContainerType>;
  ChanAndVeseInternalTermType::Pointer cvInternalTerm = ChanAndVeseInternalTermType::
↪New();
  cvInternalTerm->SetInput(input);
  cvInternalTerm->SetCoefficient(0.5);
```

(continues on next page)

(continued from previous page)

```
  // // Chan and Vese external term
  using ChanAndVeseExternalTermType =
    itk::LevelSetEquationChanAndVeseExternalTerm<InputImageType,␣
↪LevelSetContainerType>;
  ChanAndVeseExternalTermType::Pointer cvExternalTerm = ChanAndVeseExternalTermType::
↪New();
  cvExternalTerm->SetInput(input);

  // Create term container (equation rhs)
  using TermContainerType = itk::LevelSetEquationTermContainer<InputImageType,␣
↪LevelSetContainerType>;
  TermContainerType::Pointer termContainer = TermContainerType::New();
  termContainer->SetLevelSetContainer(levelSetContainer);
  termContainer->SetInput(input);
  termContainer->AddTerm(0, cvInternalTerm);
  termContainer->AddTerm(1, cvExternalTerm);

  // Create equation container
  using EquationContainerType = itk::LevelSetEquationContainer<TermContainerType>;
  EquationContainerType::Pointer equationContainer = EquationContainerType::New();
  equationContainer->SetLevelSetContainer(levelSetContainer);
  equationContainer->AddEquation(0, termContainer);

  // Create stopping criteria
  using StoppingCriterionType = itk::
↪LevelSetEvolutionNumberOfIterationsStoppingCriterion<LevelSetContainerType>;
  StoppingCriterionType::Pointer criterion = StoppingCriterionType::New();
  criterion->SetNumberOfIterations(numberOfIterations);

  // Create the visualizer
  using VisualizationType = itk::VTKVisualizeImageLevelSetIsoValues<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);
  visualizer->SetScreenCapture(true);

  // Create evolution class
  using LevelSetEvolutionType = itk::LevelSetEvolution<EquationContainerType,␣
↪LevelSetType>;
  LevelSetEvolutionType::Pointer evolution = LevelSetEvolutionType::New();
  evolution->SetEquationContainer(equationContainer);
  evolution->SetStoppingCriterion(criterion);
  evolution->SetLevelSetContainer(levelSetContainer);

  using IterationUpdateCommandType = itk::LevelSetIterationUpdateCommand
↪<LevelSetEvolutionType, VisualizationType>;
  IterationUpdateCommandType::Pointer iterationUpdateCommand =␣
↪IterationUpdateCommandType::New();
  iterationUpdateCommand->SetFilterToUpdate(visualizer);
  iterationUpdateCommand->SetUpdatePeriod(5);

  evolution->AddObserver(itk::IterationEvent(), iterationUpdateCommand);

  evolution->Update();

  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

### Visualize a Static Dense 2D Level Set as Elevation Map

### Synopsis

Visualize a static dense level-set function 2D rendered as an elevation map. From the input image, first an otsu thresholding technique is used to get a binary mask, which is then converted to a dense level-set function.

### Results

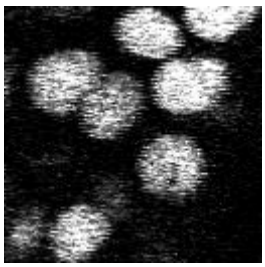


Fig. 5: Input image

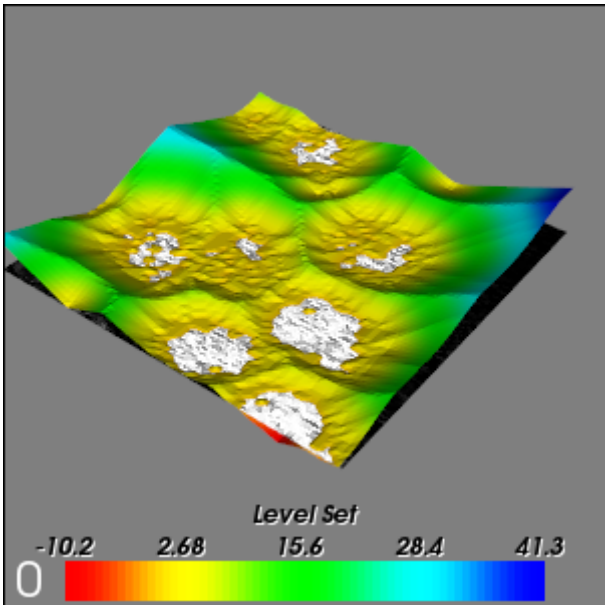



Fig. 6: Static level-sets

### Code

### C++

```
#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkLevelSetIterationUpdateCommand.h"
#include "itkLevelSetContainer.h"
#include "itkLevelSetEquationChanAndVeseInternalTerm.h"
#include "itkLevelSetEquationChanAndVeseExternalTerm.h"
#include "itkLevelSetEquationContainer.h"
#include "itkLevelSetEquationTermContainer.h"
#include "itkLevelSetEvolution.h"
#include "itkLevelSetEvolutionNumberOfIterationsStoppingCriterion.h"
#include "itkLevelSetDenseImage.h"
#include "itkVTKVisualize2DLevelSetAsElevationMap.h"
#include "itkSinRegularizedHeavisideStepFunction.h"

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "vtkRenderWindowInteractor.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "<Input Image> <Interactive (0 or 1)>" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  InputImageType::Pointer input = reader->GetOutput();

  using LevelSetPixelType = float;
  using LevelSetImageType = itk::Image<LevelSetPixelType, Dimension>;
  using LevelSetType = itk::LevelSetDenseImage<LevelSetImageType>;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using OtsuFilterType = itk::OtsuMultipleThresholdsImageFilter<InputImageType,␣
↪LevelSetImageType>;
  OtsuFilterType::Pointer otsu = OtsuFilterType::New();
  otsu->SetInput(input);
  otsu->SetNumberOfHistogramBins(256);
```

(continues on next page)

(continued from previous page)

```
  otsu->SetNumberOfThresholds(1);

  using RescaleType = itk::RescaleIntensityImageFilter<LevelSetImageType,␣
↪LevelSetImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(otsu->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(1);

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<LevelSetImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(rescaler->GetOutput());
  adaptor->Initialize();

  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // Create the visualizer
  using VisualizationType = itk::VTKVisualize2DLevelSetAsElevationMap<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(visualizer->GetRenderWindow());

  try
  {
    visualizer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  bool interactive = (std::stoi(argv[2]) != 0);
  if (interactive)
  {
    renderWindowInteractor->Start();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### Visualize a Static Dense 2D Level-Set Zero-Set

### Synopsis

Visualize a static dense level-set function 2D's zero set. From the input image, first an otsu thresholding technique is used to get a binary mask, which is then converted to a dense level-set function.

### Results

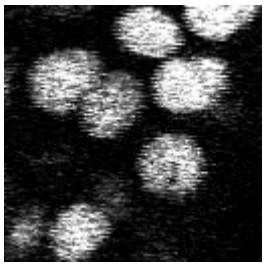


Fig. 7: Input image

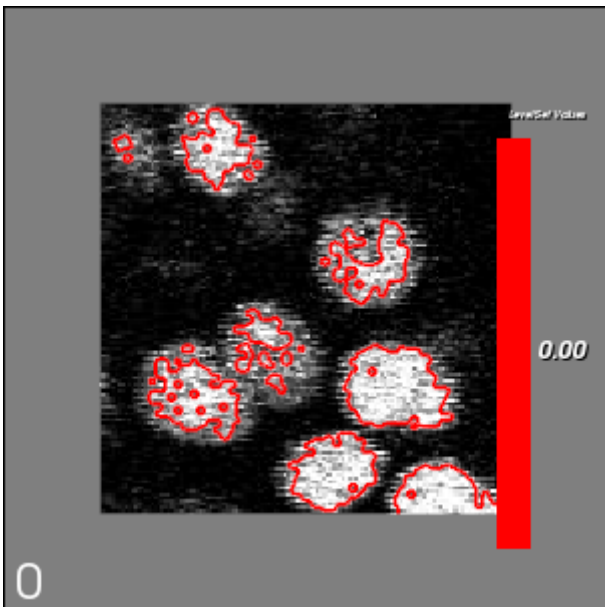



Fig. 8: Static level-sets

### Code

### C++

```
#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkLevelSetIterationUpdateCommand.h"
#include "itkLevelSetContainer.h"
#include "itkLevelSetEquationChanAndVeseInternalTerm.h"
#include "itkLevelSetEquationChanAndVeseExternalTerm.h"
#include "itkLevelSetEquationContainer.h"
#include "itkLevelSetEquationTermContainer.h"
#include "itkLevelSetEvolution.h"
#include "itkLevelSetEvolutionNumberOfIterationsStoppingCriterion.h"
#include "itkLevelSetDenseImage.h"
#include "itkVTKVisualizeImageLevelSetIsoValues.h"
#include "itkSinRegularizedHeavisideStepFunction.h"

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "vtkRenderWindowInteractor.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "<Input Image> <Interactive (0 or 1)>" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  InputImageType::Pointer input = reader->GetOutput();

  using LevelSetPixelType = float;
  using LevelSetImageType = itk::Image<LevelSetPixelType, Dimension>;
  using LevelSetType = itk::LevelSetDenseImage<LevelSetImageType>;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using OtsuFilterType = itk::OtsuMultipleThresholdsImageFilter<InputImageType,␣
↪LevelSetImageType>;
  OtsuFilterType::Pointer otsu = OtsuFilterType::New();
  otsu->SetInput(input);
  otsu->SetNumberOfHistogramBins(256);
```

(continues on next page)

(continued from previous page)

```
  otsu->SetNumberOfThresholds(1);

  using RescaleType = itk::RescaleIntensityImageFilter<LevelSetImageType,␣
↪LevelSetImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(otsu->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(1);

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<LevelSetImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(rescaler->GetOutput());
  adaptor->Initialize();

  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // Create the visualizer
  using VisualizationType = itk::VTKVisualizeImageLevelSetIsoValues<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(visualizer->GetRenderWindow());

  try
  {
    visualizer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  bool interactive = (std::stoi(argv[2]) != 0);
  if (interactive)
  {
    renderWindowInteractor->Start();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### Visualize a Static Sparse Malcolm 2D Level-Set Layers

### Synopsis

Visualize a static sparse Malcolm level-set function 2D's layers. From the input image, first an otsu thresholding technique is used to get a binary mask, which is then converted to a sparse level-set function.

Note that Malcolm's representation is composed of a single layer (value = {0});

### Results

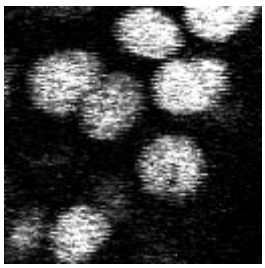


Fig. 9: Input image

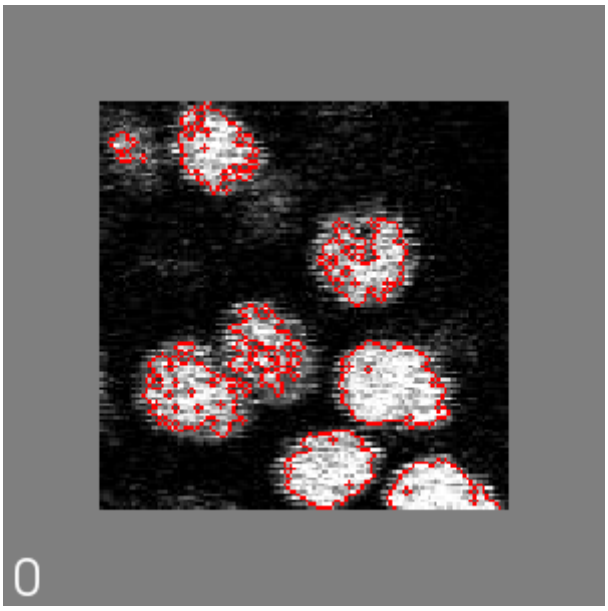



Fig. 10: Static level-sets

### Code

### C++

```
#include "itkVTKVisualize2DSparseLevelSetLayers.h"

#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkMalcolmSparseLevelSetImage.h"

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "vtkRenderWindowInteractor.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "<Input Image> <Interactive (0 or 1)>" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  InputImageType::Pointer input = reader->GetOutput();

  using LevelSetType = itk::MalcolmSparseLevelSetImage<Dimension>;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using OtsuFilterType = itk::OtsuMultipleThresholdsImageFilter<InputImageType,␣
↪InputImageType>;
  OtsuFilterType::Pointer otsu = OtsuFilterType::New();
  otsu->SetInput(input);
  otsu->SetNumberOfHistogramBins(256);
  otsu->SetNumberOfThresholds(1);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType, InputImageType>
↪;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(otsu->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(1);

  // convert a binary mask to a level-set function
```

(continues on next page)

(continued from previous page)

```
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<InputImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(rescaler->GetOutput());
  adaptor->Initialize();

  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // Create the visualizer
  using VisualizationType = itk::VTKVisualize2DSparseLevelSetLayers<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);
  visualizer->SetScreenCapture(true);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(visualizer->GetRenderWindow());

  try
  {
    visualizer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  bool interactive = (std::stoi(argv[2]) != 0);
  if (interactive)
  {
    renderWindowInteractor->Start();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### Visualize a Static Sparse Shi 2D Level-Set Layers

### Synopsis

Visualize a static sparse Shi level-set function 2D's layers. From the input image, first an otsu thresholding technique is used to get a binary mask, which is then converted to a sparse level-set function.

Note that Shi's representation is composed of 4 layers (values = {-3, -1, +1, +3})

### Results

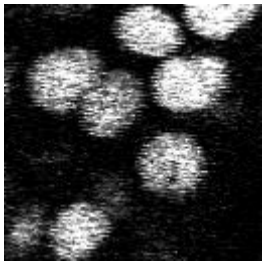


Fig. 11: Input image

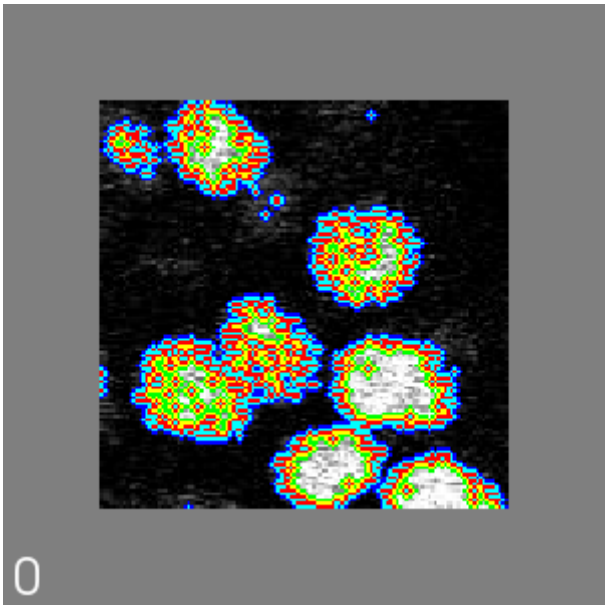



Fig. 12: Static level-sets

### Code

### C++

```
#include "itkVTKVisualize2DSparseLevelSetLayers.h"

#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkShiSparseLevelSetImage.h"

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "vtkRenderWindowInteractor.h"

int
main(int argc, char * argv[])
```

(continues on next page)

(continued from previous page)

```
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "<Input Image> <Interactive (0 or 1)>" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  InputImageType::Pointer input = reader->GetOutput();

  using LevelSetType = itk::ShiSparseLevelSetImage<Dimension>;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using OtsuFilterType = itk::OtsuMultipleThresholdsImageFilter<InputImageType,␣
↪InputImageType>;
  OtsuFilterType::Pointer otsu = OtsuFilterType::New();
  otsu->SetInput(input);
  otsu->SetNumberOfHistogramBins(256);
  otsu->SetNumberOfThresholds(1);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType, InputImageType>
↪;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(otsu->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(1);

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<InputImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(rescaler->GetOutput());
  adaptor->Initialize();

  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // Create the visualizer
  using VisualizationType = itk::VTKVisualize2DSparseLevelSetLayers<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);
```

(continues on next page)

(continued from previous page)

```
  visualizer->SetScreenCapture(true);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(visualizer->GetRenderWindow());

  try
  {
    visualizer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  bool interactive = (std::stoi(argv[2]) != 0);
  if (interactive)
  {
    renderWindowInteractor->Start();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### Visualize a Static Sparse Whitaker 2D Level-Set Layers

### Synopsis

Visualize a static sparse Whitaker level-set function 2D's layers. From the input image, first an otsu thresholding technique is used to get a binary mask, which is then converted to a sparse level-set function.

Note that Whitaker's representation is composed of 5 layers where values are real.

### Results

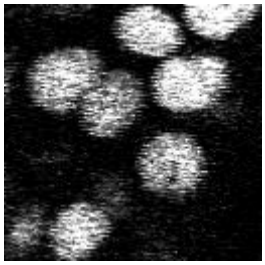


Fig. 13: Input image

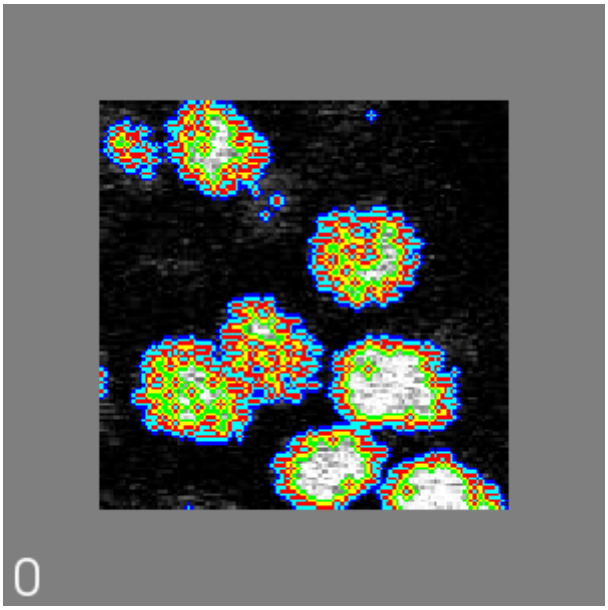



Fig. 14: Static level-sets

### Code

### C++

```
#include "itkVTKVisualize2DSparseLevelSetLayers.h"

#include "itkBinaryImageToLevelSetImageAdaptor.h"
#include "itkImageFileReader.h"
#include "itkWhitakerSparseLevelSetImage.h"

#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "vtkRenderWindowInteractor.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Missing Arguments" << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << "<Input Image> <Interactive (0 or 1)>" << std::endl;
    return EXIT_FAILURE;
  }

  // Image Dimension
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  // Read input image (to be processed).
```

(continues on next page)

(continued from previous page)

```
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  InputImageType::Pointer input = reader->GetOutput();

  using LevelSetPixelType = float;
  using LevelSetType = itk::WhitakerSparseLevelSetImage<LevelSetPixelType, Dimension>;

  // Generate a binary mask that will be used as initialization for the level
  // set evolution.
  using OtsuFilterType = itk::OtsuMultipleThresholdsImageFilter<InputImageType,␣
↪InputImageType>;
  OtsuFilterType::Pointer otsu = OtsuFilterType::New();
  otsu->SetInput(input);
  otsu->SetNumberOfHistogramBins(256);
  otsu->SetNumberOfThresholds(1);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType, InputImageType>
↪;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(otsu->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(1);

  // convert a binary mask to a level-set function
  using BinaryImageToLevelSetType = itk::BinaryImageToLevelSetImageAdaptor
↪<InputImageType, LevelSetType>;

  BinaryImageToLevelSetType::Pointer adaptor = BinaryImageToLevelSetType::New();
  adaptor->SetInputImage(rescaler->GetOutput());
  adaptor->Initialize();

  LevelSetType::Pointer levelSet = adaptor->GetModifiableLevelSet();

  // Create the visualizer
  using VisualizationType = itk::VTKVisualize2DSparseLevelSetLayers<InputImageType,␣
↪LevelSetType>;
  VisualizationType::Pointer visualizer = VisualizationType::New();
  visualizer->SetInputImage(input);
  visualizer->SetLevelSet(levelSet);
  visualizer->SetScreenCapture(true);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  renderWindowInteractor->SetRenderWindow(visualizer->GetRenderWindow());

  try
  {
    visualizer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  bool interactive = (std::stoi(argv[2]) != 0);
  if (interactive)
  {
    renderWindowInteractor->Start();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

### VTK Image to ITK Image

### Synopsis

Convert a VTK image to an ITK image.

### Results

Fig. 15: Input Image

Fig. 16: Output In QuickView

### Code

### C++

```
#include <itkImage.h>

#include <itkVTKImageToImageFilter.h>

#include "vtkSmartPointer.h"
#include "vtkPNGReader.h"
#include <vtkImageLuminance.h>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  vtkSmartPointer<vtkPNGReader> reader = vtkSmartPointer<vtkPNGReader>::New();
  reader->SetFileName(argv[1]);
  // reader->SetNumberOfScalarComponents(1); //doesn't seem to work - use␣
↪ImageLuminance instead
  reader->Update();


  // Must convert image to grayscale because itkVTKImageToImageFilter only accepts␣
↪single channel images
  vtkSmartPointer<vtkImageLuminance> luminanceFilter = vtkSmartPointer
↪<vtkImageLuminance>::New();
  luminanceFilter->SetInputConnection(reader->GetOutputPort());
  luminanceFilter->Update();

  using ImageType = itk::Image<unsigned char, 2>;

  using VTKImageToImageType = itk::VTKImageToImageFilter<ImageType>;

  VTKImageToImageType::Pointer vtkImageToImageFilter = VTKImageToImageType::New();
  vtkImageToImageFilter->SetInput(luminanceFilter->GetOutput());
  // vtkImageToImageFilter->SetInput(reader->GetOutput());
  vtkImageToImageFilter->Update();

  ImageType::Pointer image = ImageType::New();
  image->Graft(vtkImageToImageFilter->GetOutput()); // Need to do this because␣
↪QuickView can't accept const

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image.GetPointer()); // Need to do this because QuickView can't␣
↪accept smart pointers
  viewer.Visualize();
```

(continues on next page)

(continued from previous page)

```
#endif

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TOutputImage**>
**class VTKImageToImageFilter** : **public** itk::VTKImageImport<*TOutputImage*>
Converts a VTK image into an ITK image and plugs a VTK data pipeline to an ITK datapipeline.

This class puts together an itk::VTKImageImport and a vtk::ImageExport. It takes care of the details related to the connection of ITK and VTK pipelines. The User will perceive this filter as an adaptor to which a vtkImage-Data can be plugged as input and an itk::Image is produced as output.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- VTK Image To ITK Image

See itk::VTKImageToImageFilter for additional documentation.

## 3.2 Core

### 3.2.1 Common

#### Add Noise To Binary Image

#### Synopsis

Add noise to a binary image.

#### Results

Output:

```
Number of random samples: 105062
```

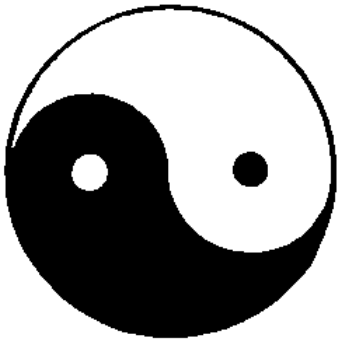


Fig. 17: Yinyang.png

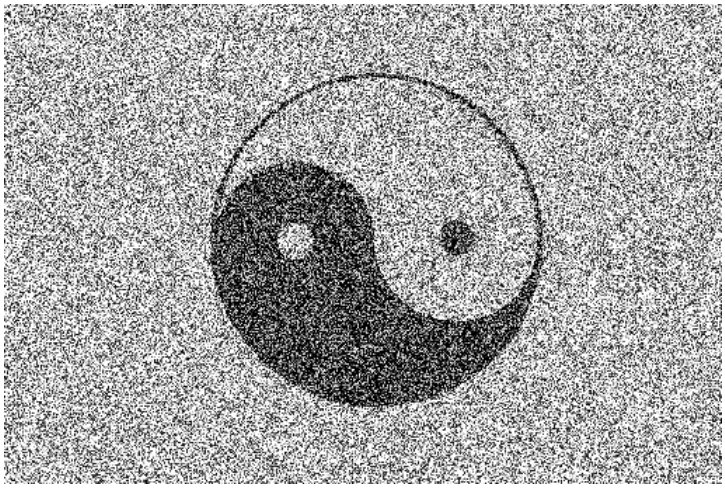


Fig. 18: Output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRandomNonRepeatingIteratorWithIndex.h"
#include "itkImageFileWriter.h"
#include "itkMersenneTwisterRandomVariateGenerator.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: " << argv[0] << " inputFile outputFile [percent]" << std::
↪endl;
    return EXIT_FAILURE;
  }
  double percent = .1;
  if (argc > 3)
  {
    percent = std::stod(argv[3]);
    if (percent >= 1.0)
    {
      percent /= 100.0;
    }
  }
  using ImageType = itk::Image<unsigned char, 2>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  using IteratorType = itk::ImageRandomNonRepeatingIteratorWithIndex<ImageType>;
  using WriterType = itk::ImageFileWriter<ImageType>;

  // Read the binary file
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  // At x% of the pixels, add a uniform random value between 0 and 255
  IteratorType it(reader->GetOutput(), reader->GetOutput()->
↪GetLargestPossibleRegion());
  it.SetNumberOfSamples(reader->GetOutput()->GetLargestPossibleRegion().
↪GetNumberOfPixels() * percent);
  std::cout << "Number of random samples: " << it.GetNumberOfSamples() << std::endl;
  using GeneratorType = itk::Statistics::MersenneTwisterRandomVariateGenerator;
  GeneratorType::Pointer random = GeneratorType::New();

  it.GoToBegin();
  while (!it.IsAtEnd())
  {
    it.Set(random->GetUniformVariate(0, 255));
    ++it;
  }

  // Write the file
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(reader->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRandomNonRepeatingIteratorWithIndex** : **public** itk::*ImageRandomNonRepeatingConstIteratorWithInde*

A multi-dimensional image iterator that visits image pixels within a region in a random order, without repeating.

This class was contributed by Rupert Brooks, McGill Centre for Intelligent Machines, Montreal, Canada. It is heavily based on the ImageRandomIterator class.

This iterator is a subclass of itk::ImageRandomNonRepeatingConstIteratorWithIndex that adds write-access functionality. Please see itk::ImageRandomNonRepeatingConstIteratorWithIndex for more information.

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**Author** Rupert Brooks, McGill Centre for Intelligent Machines. Canada

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageRandomNonRepeatingConstIteratorWithIndex

**See** ImageRandomNonRepeatingIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Noise To Binary Image

See itk::ImageRandomNonRepeatingIteratorWithIndex for additional documentation.

### Add Offset To Index

### Synopsis

Add an offset to a pixel index

### Results

**Output:**

```
index: [5, 5]
offset: [1, 1]
index + offset: [6, 6]
```

```
index: [5, 5]
offset: [-1, 1]
index + offset: [4, 6]
```

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.9.0"):
    print("ITK 4.9.0 is required.")
    sys.exit(1)

Dimension = 2

index = itk.Index[Dimension]()
index.Fill(5)

offset = itk.Offset[Dimension]()
offset.Fill(1)

newIndex = index + offset

print("index: " + str([int(index[0]), int(index[1])]))
print("offset: " + str([int(offset[0]), int(offset[1])]))
print("index + offset: " + str([int(newIndex[0]), int(newIndex[1])]))
print("")

offset[0] = -1
newIndex = index + offset

print("index: " + str([int(index[0]), int(index[1])]))
print("offset: " + str([int(offset[0]), int(offset[1])]))
print("index + offset: " + str([int(newIndex[0]), int(newIndex[1])]))
print("")
```

### C++

```
#include "itkIndex.h"
#include "itkOffset.h"

#include <iostream>

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;

  itk::Index<Dimension> index;
  index.Fill(5);

  itk::Offset<Dimension> offset;
```

(continues on next page)

(continued from previous page)

```
  offset.Fill(1);

  std::cout << "index: " << index << std::endl;
  std::cout << "offset: " << offset << std::endl;
  std::cout << "index + offset: " << index + offset << std::endl;
  std::cout << std::endl;

  offset[0] = -1;

  std::cout << "index: " << index << std::endl;
  std::cout << "offset: " << offset << std::endl;
  std::cout << "index + offset: " << index + offset << std::endl;
  std::cout << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VDimension** = 2>
**struct Offset**

Represent a n-dimensional offset between two n-dimensional indexes of n-dimensional image.

Offset is a templated class to represent a multi-dimensional offset, i.e. (i,j,k,...). Offset is templated over the dimension of the space. ITK assumes the first element of a size (bounds) is the fastest moving index.

For efficiency, Offset does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Offset is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Offset<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Offset<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**See** Index

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Offset To Index

See itk::Offset for additional documentation.

template<unsigned int **VDimension** = 2>
**struct Index**

Represent a n-dimensional index in a n-dimensional image.

Index is a templated class to represent a multi-dimensional index, i.e. (i,j,k,...). Index is templated over the dimension of the index. ITK assumes the first element of an index is the fastest moving index.

For efficiency sake, Index does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Index is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Index<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Index<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Distance between two indices
- Create A Index

See itk::Index for additional documentation.

### Apply a Filter Only to a Specified Region of an Image

### Synopsis

The key of using SetRequestedRegion to tell a filter to only operate on a specified ImageRegion is to call SetRequestedRegion on the filters GetOutput(). That is, to tell the DerivativeImageFilter to only operate on a small region, you must do

```
derivativeFilter->GetOutput()->SetRequestedRegion(smallRegion);
derivativeFilter->Update();
```

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk

Dimension = 2
PixelType = itk.F
ImageType = itk.Image[PixelType, Dimension]

smallSize = itk.Size[Dimension]()
smallSize.Fill(10)
```

(continues on next page)

(continued from previous page)

```
index = itk.Index[Dimension]()
index.Fill(0)

region = itk.ImageRegion[Dimension]()
region.SetIndex(index)
region.SetSize(smallSize)

bigSize = itk.Size[Dimension]()
bigSize.Fill(10)

RandomSourceType = itk.RandomImageSource[ImageType]
randomImageSource = RandomSourceType.New()
randomImageSource.SetNumberOfWorkUnits(1)  # to produce non-random results
randomImageSource.SetSize(bigSize)
randomImageSource.GetOutput().SetRequestedRegion(region)

print("Created random image.")

DerivativeImageFilterType = itk.DerivativeImageFilter[ImageType, ImageType]

derivativeFilter = DerivativeImageFilterType.New()
derivativeFilter.SetInput(randomImageSource.GetOutput())
derivativeFilter.SetDirection(0)  # "x" axis
derivativeFilter.GetOutput().SetRequestedRegion(region)
derivativeFilter.Update()

print("Computed derivative.")
```

### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkDerivativeImageFilter.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = float;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::SizeType smallSize;
  smallSize.Fill(10);

  ImageType::IndexType index;
  index.Fill(0);

  ImageType::RegionType region(index, smallSize);

  ImageType::SizeType bigSize;
  bigSize.Fill(10000);

  using RandomSourceType = itk::RandomImageSource<ImageType>;
```

(continues on next page)

(continued from previous page)

```
  RandomSourceType::Pointer randomImageSource = RandomSourceType::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce non-random results
  randomImageSource->SetSize(bigSize);
  randomImageSource->GetOutput()->SetRequestedRegion(smallSize);

  std::cout << "Created random image." << std::endl;

  using DerivativeImageFilterType = itk::DerivativeImageFilter<ImageType, ImageType>;

  DerivativeImageFilterType::Pointer derivativeFilter = DerivativeImageFilterType::
↪New();
  derivativeFilter->SetInput(randomImageSource->GetOutput());
  derivativeFilter->SetDirection(0); // "x" axis
  derivativeFilter->GetOutput()->SetRequestedRegion(smallSize);
  derivativeFilter->Update();

  std::cout << "Computed derivative." << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image

- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Apply Custom Operation to Each Pixel in Image

### Synopsis

Apply a custom operation to each pixel in an image.

### Results

Output:

```
pixel (1,1) was = [1,0]
pixel (1,1) now = [-4.37144e-08, 1]
```

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkVector.h"
#include "itkVariableLengthVector.h"
#include "itkRigid2DTransform.h"
#include "itkUnaryFunctorImageFilter.h"
#include "itkMath.h"
```

(continues on next page)

(continued from previous page)

```
template <class TInput, class TOutput>
class RotateVectors
{
public:
  RotateVectors() = default;
  ~RotateVectors() = default;
  bool
  operator!=(const RotateVectors &) const
  {
    return false;
  }
  bool
  operator==(const RotateVectors & other) const
  {
    return !(*this != other);
  }
  inline TOutput
  operator()(const TInput & A) const
  {
    using VectorType = itk::Vector<float, 2>;
    VectorType v;
    v[0] = A[0];
    v[1] = A[1];

    using TransformType = itk::Rigid2DTransform<float>;

    TransformType::Pointer transform = TransformType::New();
    transform->SetAngle(itk::Math::pi / 2.0);

    VectorType outputV = transform->TransformVector(v);
    TOutput    transformedVector;
    transformedVector.SetSize(2);
    transformedVector[0] = outputV[0];
    transformedVector[1] = outputV[1];

    return transformedVector;
  }
};

int
main(int, char *[])
{
  using ImageType = itk::VectorImage<float, 2>;

  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 2;
  size[1] = 3;

  region.SetSize(size);
  region.SetIndex(start);

  ImageType::Pointer image = ImageType::New();
```

(continues on next page)

(continued from previous page)

```
  image->SetRegions(region);
  image->SetVectorLength(2);
  image->Allocate();

  ImageType::IndexType pixelIndex;
  pixelIndex[0] = 1;
  pixelIndex[1] = 1;

  using VectorType = itk::VariableLengthVector<float>;
  VectorType v;
  v.SetSize(2);
  v[0] = 1;
  v[1] = 0;

  image->SetPixel(pixelIndex, v);

  using FilterType =
    itk::UnaryFunctorImageFilter<ImageType, ImageType, RotateVectors<ImageType::
↪PixelType, ImageType::PixelType>>;

  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->Update();

  ImageType::PixelType inputPixelValue = image->GetPixel(pixelIndex);
  ImageType::PixelType outputPixelValue = filter->GetOutput()->GetPixel(pixelIndex);

  std::cout << "pixel (1,1) was = " << inputPixelValue << std::endl;
  std::cout << "pixel (1,1) now = " << outputPixelValue << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TFunction**>
**class UnaryFunctorImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Implements pixel-wise generic operation on one image.

This class is parameterized over the type of the input image and the type of the output image. It is also parameterized by the operation to be applied, using a Functor style.

UnaryFunctorImageFilter allows the output dimension of the filter to be larger than the input dimension. Thus subclasses of the UnaryFunctorImageFilter (like the CastImageFilter) can be used to promote a 2D image to a 3D image, etc.

**See** UnaryGeneratorImageFilter

**See** BinaryFunctorImageFilter TernaryFunctorImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Custom Operation To Each Pixel In Image

See itk::UnaryFunctorImageFilter for additional documentation.

### Bounding Box of a Point Set

### Synopsis

Get the bounding box of a PointSet

### Results

**Output::** bounds: [0, 0.1, 0, 0.1, 0, 0] center: [0.05, 0.05, 0] diagonal length squared: 0.02

### Code

### Python

```
#!/usr/bin/env python

import itk
import platform

Dimension = 3
CoordType = itk.ctype("float")
# Windows requires unsigned long long for 64-bit identifiers
if platform.system() == "Windows":
    ElementIdentifierType = itk.ctype("unsigned long long")
else:
    ElementIdentifierType = itk.ctype("unsigned long")


PointSetType = itk.PointSet[CoordType, Dimension]

pointSet = PointSetType.New()
points = pointSet.GetPoints()

# Create points
p0 = itk.Point[CoordType, Dimension]()
p1 = itk.Point[CoordType, Dimension]()
p2 = itk.Point[CoordType, Dimension]()

p0[0] = 0.0
p0[1] = 0.0
p0[2] = 0.0
p1[0] = 0.1
p1[1] = 0.0
p1[2] = 0.0
p2[0] = 0.0
p2[1] = 0.1
p2[2] = 0.0

points.InsertElement(0, p0)
points.InsertElement(1, p1)
points.InsertElement(2, p2)
```

(continues on next page)

(continued from previous page)

```
VecContType = itk.VectorContainer[
    ElementIdentifierType, itk.Point[CoordType, Dimension]
]
BoundingBoxType = itk.BoundingBox[
    ElementIdentifierType, Dimension, CoordType, VecContType
]

boundingBox = BoundingBoxType.New()
boundingBox.SetPoints(points)
boundingBox.ComputeBoundingBox()

print("bounds: " + str(boundingBox.GetBounds()))
print("center: " + str(boundingBox.GetCenter()))
print("diagonal length squared: " + str(boundingBox.GetDiagonalLength2()))
```

### C++

```
#include "itkPoint.h"
#include "itkPointSet.h"
#include "itkBoundingBox.h"

int
main(int, char *[])
{
  using CoordType = float;
  constexpr unsigned int Dimension = 3;

  using PointSetType = itk::PointSet<CoordType, Dimension>;

  using PointIdentifier = PointSetType::PointIdentifier;
  using PointType = PointSetType::PointType;
  using PointsContainerPointer = PointSetType::PointsContainerPointer;

  PointSetType::Pointer  pointSet = PointSetType::New();
  PointsContainerPointer points = pointSet->GetPoints();

  // Create points
  PointType p0, p1, p2;

  p0[0] = 0.0;
  p0[1] = 0.0;
  p0[2] = 0.0;
  p1[0] = 0.1;
  p1[1] = 0.0;
  p1[2] = 0.0;
  p2[0] = 0.0;
  p2[1] = 0.1;
  p2[2] = 0.0;

  points->InsertElement(0, p0);
  points->InsertElement(1, p1);
  points->InsertElement(2, p2);
```

(continues on next page)

(continued from previous page)

```
  using BoundingBoxType = itk::BoundingBox<PointIdentifier, Dimension, CoordType>;

  BoundingBoxType::Pointer boundingBox = BoundingBoxType::New();
  boundingBox->SetPoints(points);
  boundingBox->ComputeBoundingBox();

  std::cout << "bounds: " << boundingBox->GetBounds() << std::endl;
  std::cout << "center: " << boundingBox->GetCenter() << std::endl;
  std::cout << "diagonal length squared: " << boundingBox->GetDiagonalLength2() <<␣
↪std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class PointSet** : **public** itk::DataObject

A superclass of the N-dimensional mesh structure; supports point (geometric coordinate and attribute) definition.

PointSet is a superclass of the N-dimensional mesh structure (itk::Mesh). It provides the portion of the mesh definition for geometric coordinates (and associated attribute or pixel information). The defined API provides operations on points but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier to access the points. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

PointSet has two template parameters. The first is the pixel type, or the type of data stored (optionally) with the points. The second is the "MeshTraits" structure controlling type information characterizing the point set. Most users will be happy with the defaults, and will not have to worry about this second argument.

Template parameters for PointSet:

TPixelType = The type stored as data for the point.

TMeshTraits = Type information structure for the point set.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a PointSet
- Read a PointSet
- Write a PointSet
- Bounding Box Of A Point Set

Subclassed by itk::Mesh< TPixelType, VDimension, TMeshTraits >

See itk::PointSet for additional documentation.

template<typename **TPointIdentifier** = IdentifierType, unsigned int **VPointDimension** = 3, typename **TCoordRep** = float, t

**class BoundingBox** : **public** itk::*Object*

Represent and compute information about bounding boxes.

BoundingBox is a supporting class that represents, computes, and caches information about bounding boxes. The bounding box can be computed from several sources, including manual specification and computation from an input points container.

This is a templated, n-dimensional version of the bounding box. Bounding boxes are represented by n pairs of (min,max) pairs, where min is the minimum coordinate value and max is the maximum coordinate value for coordinate axis i.

Template parameters for BoundingBox:

**Template Parameters**

- `TPointIdentifier`: The type used to access a particular point (i.e., a point's id)
- `TCoordRep`: Numerical type with which to represent each coordinate value.
- `VPointDimension`: Geometric dimension of space.

See itk::BoundingBox for additional documentation.

### Bresenham Line

### Synopsis

Get the points on a Bresenham line between two points.

### Results

Output:

```
[0, 0]
[1, 1]
[2, 2]
[3, 3]
[0, 0]
[1, 1]
[2, 2]
[3, 3]
[4, 4]
[5, 5]
[6, 6]
```

### Code

### C++

```
#include "itkBresenhamLine.h"
#include "itkVector.h"
#include "itkOffset.h"
#include "itkPoint.h"
```

(continues on next page)

(continued from previous page)

```
#include <iostream>

static void
Vector();
static void
Line();

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  Vector();
  Line();

  return EXIT_SUCCESS;
}

void
Vector()
{

  itk::BresenhamLine<2> line;

  itk::Vector<float, 2> v;
  v[0] = 1;
  v[1] = 1;
  std::vector<itk::Offset<2>> offsets = line.BuildLine(v, 4);

  for (auto offset : offsets)
  {
    std::cout << offset << std::endl;
  }
}

void
Line()
{

  itk::BresenhamLine<2> line;
  itk::Index<2>         pixel0;
  pixel0[0] = 0;
  pixel0[1] = 0;

  itk::Index<2> pixel1;
  pixel1[0] = 5;
  pixel1[1] = 5;

  std::vector<itk::Index<2>> pixels = line.BuildLine(pixel0, pixel1);

  for (auto pixel : pixels)
  {
    std::cout << pixel << std::endl;
  }
}
```

### Classes demonstrated

template<unsigned int **VDimension**>
**class BresenhamLine**

See itk::BresenhamLine for additional documentation.

### Build a Hello World Program

### Synopsis

This demonstrates building a simple Hello World ITK program. We instantiate an image and print it to stdout.

### Results

Output:

```
ITK Hello World!
Image (0x19ca6d0)
  RTTI typeinfo:   itk::Image<unsigned short, 3u>
  Reference Count: 2
  Modified Time: 1
  Debug: Off
  Observers:
    none
  Source: (none)
  Source output name: (none)
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 0
  UpdateMTime: 0
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  BufferedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  RequestedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  Spacing: [1, 1, 1]
  Origin: [0, 0, 0]
  Direction:
1 0 0
0 1 0
0 0 1

  IndexToPointMatrix:
  1 0 0
0 1 0
```

(continues on next page)

(continued from previous page)

```
0 0 1

  PointToIndexMatrix:
  1 0 0
0 1 0
0 0 1

  Inverse Direction:
  1 0 0
0 1 0
0 0 1

  PixelContainer:
    ImportImageContainer (0x19ca990)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned short>
      Reference Count: 1
      Modified Time: 2
      Debug: Off
      Observers:
        none
      Pointer: 0
      Container manages memory: true
      Size: 0
      Capacity: 0
```

### Code

### C++

```
#include "itkImage.h"

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned short, 3>;
  ImageType::Pointer image = ImageType::New();

  std::cout << "ITK Hello World!" << std::endl;
  std::cout << image << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```python
#!/usr/bin/env python

import itk

ImageType = itk.Image[itk.UL, 3]
image = ImageType.New()

print("ITK Hello World!")
print(image)
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, . . .

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function

- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Cast Vector Image to Another Type

### Synopsis

Cast a VectorImage to another type of VectorImage.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkCastImageFilter.h"

int
main(int /*argc*/, char * /*argv*/[])
{
  typedef itk::VectorImage<unsigned char, 2> UnsignedCharVectorImageType;
  typedef itk::VectorImage<float, 2>          FloatVectorImageType;

  FloatVectorImageType::Pointer image = FloatVectorImageType::New();

  typedef itk::CastImageFilter<FloatVectorImageType, UnsignedCharVectorImageType>␣
↪CastImageFilterType;
  CastImageFilterType::Pointer vectorCastImageFilter = CastImageFilterType::New();
  vectorCastImageFilter->SetInput(image);
  vectorCastImageFilter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 3>
**class VectorImage** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional vector image class.

This class differs from Image in that it is intended to represent multiple images. Each pixel represents *k* measurements, each of datatype *TPixel*. The memory organization of the resulting image is as follows: ... Pi0 Pi1 Pi2 Pi3 P(i+1)0 P(i+1)1 P(i+1)2 P(i+1)3 P(i+2)0 ... where Pi0 represents the 0th measurement of the pixel at index i.

Conceptually, a `VectorImage< TPixel, 3 >` is the same as a `Image< VariableLengthVector< TPixel >, 3 >`. The difference lies in the memory organization. The latter results in a fragmented organization with each location in the Image holding a pointer to an `VariableLengthVector` holding the actual pixel. The former stores the *k* pixels instead of a pointer reference, which apart from avoiding fragmentation of memory also avoids storing a 8 bytes of pointer reference for each pixel. The parameter *k* can be set using `SetVectorLength`.

The API of the class is such that it returns a pixeltype VariableLengthVector< TPixel > when queried, with the data internally pointing to the buffer. (the container does not manage the memory). Similarly SetPixel calls can be made with VariableLengthVector< TPixel >.

The API of this class is similar to Image.

**Caveats:** When using Iterators on this image, you cannot use the it.Value(). You must use Set/Get() methods instead.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**See** DefaultVectorPixelAccessor

**See** DefaultVectorPixelAccessorFunctor

**See** VectorImageToImagePixelAccessor

**See** VectorImageToImageAdaptor

**See** Image

**See** ImportImageContainer

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cast Vector Image To Another Type
- Create Vector Image
- Neighborhood Iterator On Vector Image

See itk::VectorImage for additional documentation.

### Check if Module Is Present

### Synopsis

Check if a specific module is present.

### Code

### C++

```
#include "itkImage.h"

#include <iostream>

int
main(int /*argc*/, char * /*argv*/[])
{

  return 0;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>
Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image

- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Compute Time Between Points

### Synopsis

Compute the time between points in code. The *Report()* method is available in ITK 4.9.0 or later.

### Results

```
Mean: 0.00756311416626
Total: 0.00756311416626
Mean: 0.00671458244324
Total: 0.0134291648865
System:              clay
Processor:           Unknown P6 family
    Cache:           15360
    Clock:           2194.87
    Cores:           12 cpus x 12 Cores = 144
    Virtual Memory:  Total: 0 Available: 0
    Physical Memory: Total:64294 Available: 36172
OSName:              Linux
    Release:         4.1.4-calculate
    Version:         #2 SMP PREEMPT Sun Aug 9 17:03:44 EDT 2015
    Platform:        x86_64
    Operating System is 64 bit
ITK Version: 4.9.0
```

(continues on next page)

(continued from previous page)

```
Name Of Probe(Time)           Iteration      Total(s)       Min(s)          Mean(s)
↪     Max(s)          Std(s)
                              2              0.0134292      0.00586605      0.00671458
↪     0.00756311      0.00120001
```

### Code

### C++

```
#include "itkTimeProbe.h"
#include "itkNumericTraits.h"

#include <iostream>
#include <string>

void
LongFunction()
{
  for (int i = 0; i < itk::NumericTraits<int>::max() / 100; i++)
  {
    double a = 0;
    (void)a;
  }
}

int
main(int, char *[])
{
  itk::TimeProbe clock;

  clock.Start();
  LongFunction();

  clock.Stop();
  std::cout << "Mean: " << clock.GetMean() << std::endl;
  std::cout << "Total: " << clock.GetTotal() << std::endl;

  clock.Start();
  LongFunction();

  clock.Stop();
  std::cout << "Mean: " << clock.GetMean() << std::endl;
  std::cout << "Total: " << clock.GetTotal() << std::endl;

  clock.Report();

  return EXIT_SUCCESS;
}
```

### Python

```python
#!/usr/bin/env python

import itk


def LongFunction():
    # CPython loops are much slower than C++,
    # so a smaller range is used in this case.
    for i in range(int(1e5)):
        a = 0.0  # noqa: F841


clock = itk.TimeProbe()

clock.Start()
LongFunction()

clock.Stop()
print("Mean: " + str(clock.GetMean()))
print("Total: " + str(clock.GetTotal()))

clock.Start()
LongFunction()

clock.Stop()
print("Mean: " + str(clock.GetMean()))
print("Total: " + str(clock.GetTotal()))

clock.Report()
```

### Classes demonstrated

**class TimeProbe** : **public** itk::ResourceProbe<RealTimeClock::TimeStampType, RealTimeClock::TimeStampType>
Computes the time passed between two points in code.

This class allows the user to trace the time passed between the execution of two pieces of code. It can be started and stopped in order to evaluate the execution over multiple passes. The values of time are taken from the RealTimeClock.

**See** RealTimeClock

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Time Between Points

See itk::TimeProbe for additional documentation.

### Concept Checking Is Floating Point

#### Synopsis

Check at compilation time that a given type is floating point.

#### Results

#### Code

#### C++

```
#include "itkImage.h"
#include "itkConceptChecking.h"

template <typename TImage>
void
IsPixelTypeFloatingPoint(const TImage * const)
{
  itkConceptMacro(nameOfCheck, (itk::Concept::IsFloatingPoint<typename TImage::
↪PixelType>));
}

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using FloatImageType = itk::Image<float, Dimension>;
  FloatImageType::Pointer f = FloatImageType::New();
  IsPixelTypeFloatingPoint(f.GetPointer());

  using DoubleImageType = itk::Image<double, Dimension>;
  DoubleImageType::Pointer d = DoubleImageType::New();
  IsPixelTypeFloatingPoint(d.GetPointer());

  return EXIT_SUCCESS;
}
```

### Concept Checking Is Same Dimension

#### Synopsis

Check at compilation time that 2 dimensions are the same.

### Results

### Code

#### C++

```
#include "itkImage.h"
#include "itkConceptChecking.h"

template <typename TImage, unsigned int VDimension>
void
CheckIfDimensionIsTheSame(const TImage * const)
{
  itkConceptMacro(nameOfCheck, (itk::Concept::SameDimension<TImage::ImageDimension,␣
↪VDimension>));
}
int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
  ImageType::Pointer image = ImageType::New();

  CheckIfDimensionIsTheSame<ImageType, 2>(image.GetPointer());

  using ImageType2 = itk::Image<PixelType, Dimension>;

  CheckIfDimensionIsTheSame<ImageType, ImageType2::ImageDimension>(image.
↪GetPointer());

  return EXIT_SUCCESS;
}
```

## Concept Checking Is Same Type

### Synopsis

Check at compilation time that 2 given types are the same.

### Results

### Code

#### C++

```
#include "itkImage.h"
#include "itkConceptChecking.h"

template <typename TImage, class TValue>
void
```

(continues on next page)

(continued from previous page)

```
CheckIfPixelTypeIsTheSameAs(const TImage * const)
{
  itkConceptMacro(nameOfCheck, (itk::Concept::SameType<typename TImage::PixelType,␣
↪TValue>));
}
int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
  ImageType::Pointer image = ImageType::New();

  CheckIfPixelTypeIsTheSameAs<ImageType, unsigned char>(image.GetPointer());

  using ImageType2 = itk::Image<PixelType, Dimension>;

  CheckIfPixelTypeIsTheSameAs<ImageType, ImageType2::PixelType>(image.GetPointer());

  return EXIT_SUCCESS;
}
```

### Convert Array to Image

### Synopsis

Convert a C-style array to an itkImage.

### Results

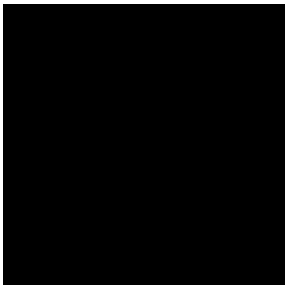


Fig. 19: Output Image.

### Code

### C++

```
#include "itkImage.h"
#include "itkImportImageFilter.h"

#include "itkImageFileWriter.h"

int
main(int, char *[])
{
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 3;
  using ImageType = itk::Image<PixelType, Dimension>;
  using ImportFilterType = itk::ImportImageFilter<PixelType, Dimension>;

  ImportFilterType::Pointer importFilter = ImportFilterType::New();

  ImportFilterType::SizeType size;

  size[0] = 200; // size along X
  size[1] = 200; // size along Y
  size[2] = 200; // size along Z

  ImportFilterType::IndexType start;
  start.Fill(0);

  ImportFilterType::RegionType region;
  region.SetIndex(start);
  region.SetSize(size);

  importFilter->SetRegion(region);

  double origin[Dimension];
  origin[0] = 0.0; // X coordinate
  origin[1] = 0.0; // Y coordinate
  origin[2] = 0.0; // Z coordinate

  importFilter->SetOrigin(origin);

  double spacing[Dimension];
  spacing[0] = 1.0; // along X direction
  spacing[1] = 1.0; // along Y direction
  spacing[2] = 1.0; // along Z direction

  importFilter->SetSpacing(spacing);

  const unsigned int numberOfPixels = size[0] * size[1] * size[2];
  auto *             localBuffer = new PixelType[numberOfPixels];

  constexpr double radius = 80.0;

  const double radius2 = radius * radius;
  PixelType *  it = localBuffer;

  for (unsigned int z = 0; z < size[2]; z++)
```

(continues on next page)

(continued from previous page)

```
  {
    const double dz = static_cast<double>(z) - static_cast<double>(size[2]) / 2.0;
    for (unsigned int y = 0; y < size[1]; y++)
    {
      const double dy = static_cast<double>(y) - static_cast<double>(size[1]) / 2.0;
      for (unsigned int x = 0; x < size[0]; x++)
      {
        const double dx = static_cast<double>(x) - static_cast<double>(size[0]) / 2.0;
        const double d2 = dx * dx + dy * dy + dz * dz;
        *it++ = (d2 < radius2) ? 255 : 0;
      }
    }
  }

  const bool importImageFilterWillOwnTheBuffer = true;
  importFilter->SetImportPointer(localBuffer, numberOfPixels,␣
↪importImageFilterWillOwnTheBuffer);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName("test.png");

  writer->SetInput(importFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class ImportImageFilter** : **public** itk::*ImageSource*<*Image*<*TPixel*, *VImageDimension*>>

Import data from a standard C array into an itk::Image.

ImportImageFilter provides a mechanism for importing data into an itk::Image. ImportImageFilter is an image source, so it behaves like any other pipeline object.

This class is templated over the pixel type and the image dimension of the output image.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert Array To Image

See itk::ImportImageFilter for additional documentation.

### Covariant Vector Dot Product

### Synopsis

Dot product of CovariantVectors

### Results

Output:

```
u :[-1, 1, -1]
v :[1, 2, 3]
DotProduct( u, v ) = -2
u - DotProduct( u, v ) * v = [1, 5, 5]
```

### Code

### C++

```
#include "itkCovariantVector.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using VectorType = itk::CovariantVector<CoordType, Dimension>;

  VectorType u;
  u[0] = -1.;
  u[1] = 1.;
  u[2] = -1.;

  VectorType v;
  v[0] = 1.;
  v[1] = 2.;
  v[2] = 3.;

  std::cout << "u :" << u << std::endl;
  std::cout << "v :" << v << std::endl;
  std::cout << "DotProduct( u, v ) = " << u * v << std::endl;
  std::cout << "u - ( u * v ) * v = " << u - (u * v) * v << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class CovariantVector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional covariant vector.

CovariantVector is a templated class that holds a single vector (i.e., an array of values). CovariantVector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

CovariantVector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarCovariantVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

CovariantVector is the type that should be used for representing normals to surfaces and gradients of functions. AffineTransform transform covariant vectors different than vectors.

**See** Image

**See** Mesh

**See** Point

**See** Vector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a CovariantVector
- Covariant Vector Norm
- Covariant Vector Dot Product

See itk::CovariantVector for additional documentation.

### Covariant Vector Norm

### Synopsis

Compute the norm of a CovariantVector and normalize it.

### Results

Output:

```
v: [1, 2, 3]
vnorm: 3.74166
vnorm2: 14
v: [0.267261, 0.534522, 0.801784]
u: [0.267261, 0.534522, 0.801784]
```

### Code

#### C++

```
#include "itkCovariantVector.h"

int
main(int, char *[])
{
  using VectorType = itk::CovariantVector<double, 3>;
  VectorType v;
  v[0] = 1.0;
  v[1] = 2.0;
  v[2] = 3.0;

  std::cout << "v: " << v << std::endl;

  // norm
  VectorType::RealValueType vnorm = v.GetNorm();
  std::cout << "vnorm: " << vnorm << std::endl;

  VectorType::RealValueType vnorm2 = v.GetSquaredNorm();
  std::cout << "vnorm2: " << vnorm2 << std::endl;

  VectorType u = v;

  // normalization
  v.Normalize();
  std::cout << "v: " << v << std::endl;

  // another way to normalize
  if (vnorm != 0.)
  {
    for (unsigned int i = 0; i < u.GetNumberOfComponents(); i++)
    {
      u[i] /= vnorm;
    }
  }

  std::cout << "u: " << u << std::endl;

  if ((u - v).GetNorm() != 0.)
  {
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class CovariantVector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional covariant vector.

CovariantVector is a templated class that holds a single vector (i.e., an array of values). CovariantVector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

CovariantVector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarCovariantVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

CovariantVector is the type that should be used for representing normals to surfaces and gradients of functions. AffineTransform transform covariant vectors different than vectors.

**See** Image

**See** Mesh

**See** Point

**See** Vector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a CovariantVector
- Covariant Vector Norm
- Covariant Vector Dot Product

See itk::CovariantVector for additional documentation.

### Create a Backward Difference Operator

### Synopsis

Create a backward difference operator

### Results

Output:

```
Size: [3, 3]
Neighborhood:
    Radius:[1, 1]
    Size:[3, 3]
    DataBuffer:NeighborhoodAllocator { this = 0x7fffb3cae9f8, begin = 0x1dd0e50,␣
↪size=9 }

[-1, -1] 0
[0, -1] 0
[1, -1] 0
[-1, 0] -1
[0, 0] 1
[1, 0] 0
[-1, 1] 0
[0, 1] 0
[1, 1] 0
```

### Code

### C++

```
#include <itkBackwardDifferenceOperator.h>

int
main(int, char *[])
{
  using PixelType = float;
  constexpr unsigned int Dimension = 2;

  using BackwardDifferenceOperatorType = itk::BackwardDifferenceOperator<PixelType,␣
↪Dimension>;
  BackwardDifferenceOperatorType backwardDifferenceOperator;

  // Create the operator for the X axis derivative
  backwardDifferenceOperator.SetDirection(0);

  itk::Size<Dimension> radius;
  radius.Fill(1);

  backwardDifferenceOperator.CreateToRadius(radius);

  std::cout << "Size: " << backwardDifferenceOperator.GetSize() << std::endl;
```

(continues on next page)

(continued from previous page)

```
  std::cout << backwardDifferenceOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << backwardDifferenceOperator.GetOffset(i) << " " <<␣
↪backwardDifferenceOperator.GetElement(i)
              << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **TDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class BackwardDifferenceOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *TDimension*, *TAllocator*>

Operator whose inner product with a neighborhood returns a "half" derivative at the center of the neighborhood.

BackwardDifferenceOperator uses backward differences i.e. $F(x) - F(x-1)$ to calculate a "half" derivative useful, among other things, in solving differential equations. It is a directional NeighborhoodOperator that should be applied to a Neighborhood using the inner product.

**Note** BackwardDifferenceOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Backward Difference Operator

See itk::BackwardDifferenceOperator for additional documentation.

### Create a CovariantVector

### Synopsis

Create a CovariantVector

### Results

Output:

```
v: [1, 2, 3]
```

### Code

### C++

```
#include "itkCovariantVector.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using VectorType = itk::CovariantVector<CoordType, Dimension>;
  VectorType v;
  v[0] = 1.0;
  v[1] = 2.0;
  v[2] = 3.0;

  std::cout << "v: " << v << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk

Dimension = 3
CoordType = itk.D

VectorType = itk.CovariantVector[CoordType, Dimension]

v = VectorType()
v[0] = 1.0
v[1] = 2.0
v[2] = 3.0
print("v: " + str(v))
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class CovariantVector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional covariant vector.

CovariantVector is a templated class that holds a single vector (i.e., an array of values). CovariantVector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

CovariantVector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarCovariantVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

CovariantVector is the type that should be used for representing normals to surfaces and gradients of functions. AffineTransform transform covariant vectors different than vectors.

**See** Image

**See** Mesh

**See** Point

**See** Vector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a CovariantVector
- Covariant Vector Norm
- Covariant Vector Dot Product

See itk::CovariantVector for additional documentation.

### Create a Fixed Array

### Synopsis

Create a FixedArray

### Results

**Output::** [0, 1]

### Code

### C++

```
#include <iostream>
#include "itkFixedArray.h"

int
main(int, char *[])
{
  itk::FixedArray<double, 2> array;
  array[0] = 0;
  array[1] = 1;

  std::cout << array << std::endl;
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk

array = itk.FixedArray[itk.D, 2]()
array[0] = 0
array[1] = 1

print(array)
```

### Classes demonstrated

template<typename **TValue**, unsigned int **VLength** = 3>
**class FixedArray**

Simulate a standard C array with copy semantics.

Simulates a standard C array, except that copy semantics are used instead of reference semantics. Also, arrays of different sizes cannot be assigned to one another, and size information is known for function returns.

The length of the array is fixed at compile time. If you wish to specify the length of the array at run-time, use the class itk::Array. If you wish to change to change the length of the array at run-time, you're best off using std::vector<>.

**Template Parameters**

- `TValue`: Element type stored at each location in the array.
- `VLength`: = Length of the array.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Fixed Array

See itk::FixedArray for additional documentation.

### Create a Index

### Synopsis

Create a itk::Index, which represents a pixel location in an itk::Image.

### Results

**Output::** [0, 0] [1, 2]

### Code

### C++

```
#include "itkIndex.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;

  using IndexType = itk::Index<Dimension>;

  IndexType index;

  // Method 1
  // set both index[0] and index[1] to the same value (in this case, 0).
  index.Fill(0);
  std::cout << index << std::endl;

  // Method 2
  // set each component of the index individually.
  index[0] = 1;
  index[1] = 2;

  std::cout << index << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk

Dimension = 2

index = itk.Index[Dimension]()

# Method 1
```

(continues on next page)

(continued from previous page)

```
# set both index[0] and index[1] to the same value (in this case, 0).
index.Fill(0)
print(index)

# Method 2
# set each component of the index individually.
index[0] = 1
index[1] = 2
print(index)
```

### Classes demonstrated

template<unsigned int **VDimension** = 2>
**struct Index**

Represent a n-dimensional index in a n-dimensional image.

Index is a templated class to represent a multi-dimensional index, i.e. (i,j,k,...). Index is templated over the dimension of the index. ITK assumes the first element of an index is the fastest moving index.

For efficiency sake, Index does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Index is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Index<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Index<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Distance between two indices
- Create A Index

See itk::Index for additional documentation.

### Create a PointSet

### Synopsis

Create a PointSet

### Code

### C++

```
#include "itkPoint.h"
#include "itkPointSet.h"

int
main(int, char *[])
{
  using PixelType = float;
  constexpr unsigned int Dimension = 3;

  using PointSetType = itk::PointSet<PixelType, Dimension>;
  PointSetType::Pointer PointSet = PointSetType::New();

  using PointsContainerPointer = PointSetType::PointsContainerPointer;
  PointsContainerPointer points = PointSet->GetPoints();

  // Create points
  using PointType = PointSetType::PointType;
  PointType p0, p1, p2;

  p0[0] = 0.0;
  p0[1] = 0.0;
  p0[2] = 0.0;
  p1[0] = 0.1;
  p1[1] = 0.0;
  p1[2] = 0.0;
  p2[0] = 0.0;
  p2[1] = 0.1;
  p2[2] = 0.0;

  points->InsertElement(0, p0);
  points->InsertElement(1, p1);
  points->InsertElement(2, p2);

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk

PixelType = itk.F
Dimension = 3

PointSetType = itk.PointSet[PixelType, Dimension]
PointSet = PointSetType.New()

points = PointSet.GetPoints()
```

(continues on next page)

(continued from previous page)

```
# Create points
p0 = itk.Point[PixelType, Dimension]()
p1 = itk.Point[PixelType, Dimension]()
p2 = itk.Point[PixelType, Dimension]()

p0[0] = 0.0
p0[1] = 0.0
p0[2] = 0.0
p1[0] = 0.1
p1[1] = 0.0
p1[2] = 0.0
p2[0] = 0.0
p2[1] = 0.1
p2[2] = 0.0

points.InsertElement(0, p0)
points.InsertElement(1, p1)
points.InsertElement(2, p2)
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class PointSet** : **public** itk::DataObject

A superclass of the N-dimensional mesh structure; supports point (geometric coordinate and attribute) definition.

PointSet is a superclass of the N-dimensional mesh structure (itk::Mesh). It provides the portion of the mesh definition for geometric coordinates (and associated attribute or pixel information). The defined API provides operations on points but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier to access the points. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

PointSet has two template parameters. The first is the pixel type, or the type of data stored (optionally) with the points. The second is the "MeshTraits" structure controlling type information characterizing the point set. Most users will be happy with the defaults, and will not have to worry about this second argument.

Template parameters for PointSet:

TPixelType = The type stored as data for the point.

TMeshTraits = Type information structure for the point set.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a PointSet
- Read a PointSet
- Write a PointSet
- Bounding Box Of A Point Set

Subclassed by itk::Mesh< TPixelType, VDimension, TMeshTraits >

See itk::PointSet for additional documentation.

### Create a Size

### Synopsis

Create a itk::Size, which represents the size of a region in an itk::Image.

### Results

Output:

```
[0, 0]
[1, 2]
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

Dimension = 2

size = itk.Size[Dimension]()

# Method 1
#
# Set both components (size[0] and size[1]) to the same value
# (in this case, 0).
size.Fill(0)
print(size)

# Method 2
#
# Set each component separately.
size[0] = 1
size[1] = 2
print(size)
```

### C++

```cpp
#include <iostream>
#include "itkSize.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  itk::Size<Dimension>   size;

  // Method 1
  // set both components (size[0] and size[1]) to the same value
  // (in this case, 0).
  size.Fill(0);
  std::cout << size << std::endl;

  // Method 2
  // set each component separately.
  size[0] = 1;
  size[1] = 2;

  std::cout << size << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VDimension** = 2>
**struct Size**

Represent a n-dimensional size (bounds) of a n-dimensional image.

Size is a templated class to represent multi-dimensional array bounds, i.e. (I,J,K,...). Size is templated over teh dimension of the bounds. ITK assumes the first element of a size (bounds) is the fastest moving index.

For efficiency, Size does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Size is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Size<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Size<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**See** Index

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Size

See itk::Size for additional documentation.

### Create a Vector

### Synopsis

Create a Vector

### Results

Output:

```
v: [1, 2, 3]
```

### Code

### C++

```
#include "itkVector.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using VectorType = itk::Vector<CoordType, Dimension>;

  VectorType v;
  v[0] = 1.0;
  v[1] = 2.0;
  v[2] = 3.0;
  std::cout << "v: " << v << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk

Dimension = 3
CoordType = itk.D

VectorType = itk.Vector[CoordType, Dimension]

v = VectorType()
v[0] = 1.0
v[1] = 2.0
v[2] = 3.0
print("v: " + str(v))
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class Vector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional vector.

Vector is a templated class that holds a single vector (i.e., an array of values). Vector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

Vector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

**See** Image

**See** Mesh

**See** Point

**See** CovariantVector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a vector
- Dot product (inner product) of two vectors

See itk::Vector for additional documentation.

### Create an Image

### Synopsis

This example illustrates how to manually construct Image.

This example illustrates how to manually construct an *itk::Image* class. The following is the minimal code needed to instantiate, declare and create the Image class.

### Results

```
Image (0x1eb5f10)
  RTTI typeinfo:   itk::Image<unsigned char, 3u>
  Reference Count: 1
  Modified Time: 4
  Debug: Off
  Object Name:
  Observers:
    none
```

(continues on next page)

(continued from previous page)

```
  Source: (none)
  Source output name: (none)
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 0
  UpdateMTime: 0
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [200, 200, 200]
  BufferedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [200, 200, 200]
  RequestedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [200, 200, 200]
  Spacing: [1, 1, 1]
  Origin: [0, 0, 0]
  Direction:
1 0 0
0 1 0
0 0 1

  IndexToPointMatrix:
1 0 0
0 1 0
0 0 1

  PointToIndexMatrix:
1 0 0
0 1 0
0 0 1

  Inverse Direction:
1 0 0
0 1 0
0 0 1

  PixelContainer:
    ImportImageContainer (0x1a678a0)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned char>
      Reference Count: 1
      Modified Time: 5
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 0x7facf287d010
      Container manages memory: true
      Size: 8000000
      Capacity: 8000000
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

Dimension = 3
PixelType = itk.ctype("unsigned char")
ImageType = itk.Image[PixelType, Dimension]

image = ImageType.New()

start = itk.Index[Dimension]()
start[0] = 0  # first index on X
start[1] = 0  # first index on Y
start[2] = 0  # first index on Z

size = itk.Size[Dimension]()
size[0] = 200  # size along X
size[1] = 200  # size along Y
size[2] = 200  # size along Z

region = itk.ImageRegion[Dimension]()
region.SetSize(size)
region.SetIndex(start)

image.SetRegions(region)
image.Allocate()

print(image)
```

### C++

```cpp
#include "itkImage.h"

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned char, 3>;
  ImageType::Pointer image = ImageType::New();

  ImageType::IndexType start;
  start[0] = 0; // first index on X
  start[1] = 0; // first index on Y
  start[2] = 0; // first index on Z

  ImageType::SizeType size;
  size[0] = 200; // size along X
  size[1] = 200; // size along Y
  size[2] = 200; // size along Z

  ImageType::RegionType region;
```

(continues on next page)

(continued from previous page)

```
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  std::cout << image << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, . . .

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function

- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Create an Image Region

### Synopsis

This class holds an Index (the "bottom left" corner of a region) and a Size (the size of the region). These two items together completely describe a region.

### Results

**Output::** ImageRegion (0x7fff45dad5c0) Dimension: 2 Index: [1, 1] Size: [3, 3]

### Code

### C++

```
#include "itkImageRegion.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;

  using RegionType = itk::ImageRegion<Dimension>;
  RegionType::SizeType size;
  size.Fill(3);

  RegionType::IndexType index;
  index.Fill(1);

  RegionType region(index, size);

  std::cout << region << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```python
#!/usr/bin/env python

import itk

Dimension = 2

size = itk.Size[Dimension]()
size.Fill(3)

index = itk.Index[Dimension]()
index.Fill(1)

RegionType = itk.ImageRegion[Dimension]
region = RegionType()
region.SetIndex(index)
region.SetSize(size)

print(region)
```

### Classes demonstrated

template<unsigned int **VImageDimension**>
**class ImageRegion** : **public** itk::Region

An image region represents a structured region of data.

ImageRegion is an class that represents some structured portion or piece of an Image. The ImageRegion is represented with an index and a size in each of the n-dimensions of the image. (The index is the corner of the image, the size is the lengths of the image in each of the topological directions.)

**See** Region

**See** Index

**See** Size

**See** MeshRegion

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- An object which holds the index (start) and size of a region of an image
- Determine image region intersection
- Determine if a pixel is inside of a region
- Determine the overlap of two regions

See itk::ImageRegion for additional documentation.

### Create an Image of Vectors

### Synopsis

This example illustrates how to instantiate and use an *itk::Image* whose pixels are of *itk::Vector* type.

### Results

```
C++

[1.345, 6.841, 3.295]

Python

itkVectorF3 ([1.345, 6.841, 3.295])
```

### Code

### Python

```
#!/usr/bin/env python

import itk

Dimension = 3
ComponentType = itk.ctype("float")
PixelType = itk.Vector[ComponentType, Dimension]
ImageType = itk.Image[PixelType, Dimension]

image = ImageType.New()

start = itk.Index[Dimension]()
start[0] = 0
start[1] = 0
start[2] = 0

size = itk.Size[Dimension]()
size[0] = 200
size[1] = 200
size[2] = 200

region = itk.ImageRegion[Dimension]()
region.SetSize(size)
region.SetIndex(start)

image.SetRegions(region)
image.Allocate()

vectorValue = PixelType()
vectorValue[0] = 0
vectorValue[1] = 0
vectorValue[2] = 0
image.FillBuffer(vectorValue)
```

(continues on next page)

(continued from previous page)

```
pixelIndex = itk.Index[Dimension]()
pixelIndex[0] = 27
pixelIndex[1] = 29
pixelIndex[2] = 37

pixelValue = PixelType()
pixelValue[0] = 1.345
pixelValue[1] = 6.841
pixelValue[2] = 3.295

image.SetPixel(pixelIndex, pixelValue)

value = image.GetPixel(pixelIndex)

print(value)
```

### C++

```
#include "itkVector.h"
#include "itkImage.h"

int
main(int, char *[])
{
  using PixelType = itk::Vector<float, 3>;
  using ImageType = itk::Image<PixelType, 3>;

  // Then the image object can be created
  ImageType::Pointer image = ImageType::New();

  // The image region should be initialized
  const ImageType::IndexType start = { { 0, 0, 0 } };       // First index at {X,Y,Z}
  const ImageType::SizeType  size = { { 200, 200, 200 } }; // Size of {X,Y,Z}

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  // Pixel data is allocated
  image->SetRegions(region);
  image->Allocate();

  // The image buffer is initialized to a particular value
  ImageType::PixelType initialValue;

  // A vector can initialize all its components to the
  // same value by using the Fill() method.
  initialValue.Fill(0.0);

  // Now the image buffer can be initialized with this
  // vector value.
  image->FillBuffer(initialValue);
```

(continues on next page)

(continued from previous page)

```
  const ImageType::IndexType pixelIndex = { { 27, 29, 37 } }; // Position {X,Y,Z}


  ImageType::PixelType pixelValue;
  pixelValue[0] = 1.345; // x component
  pixelValue[1] = 6.841; // y component
  pixelValue[2] = 3.295; // x component

  image->SetPixel(pixelIndex, pixelValue);

  ImageType::PixelType value = image->GetPixel(pixelIndex);

  std::cout << value << std::endl;


  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class Vector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional vector.

Vector is a templated class that holds a single vector (i.e., an array of values). Vector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

Vector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

**See** Image

**See** Mesh

**See** Point

**See** CovariantVector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a vector
- Dot product (inner product) of two vectors

See itk::Vector for additional documentation.

template<typename **TPixel**, unsigned int **VImageDimension** = 2>

**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Create an RGB Image

### Synopsis

Create an RGB Image

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk

Dimension = 2
RGBPixelType = itk.RGBPixel[itk.ctype("unsigned char")]

image = itk.Image[RGBPixelType, Dimension].New()
```

### C++

```
#include "itkImage.h"
#include "itkRGBPixel.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using RGBPixelType = itk::RGBPixel<unsigned char>;

  using RGBImageType = itk::Image<RGBPixelType, Dimension>;
  RGBImageType::Pointer image = RGBImageType::New();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TComponent** = unsigned short>
**class RGBPixel** : **public** itk::*FixedArray*<*TComponent*, 3>

Represent Red, Green and Blue components for color images.

This class is templated over the representation used for each component.

The following syntax for assigning an index is allowed/suggested:

```
RGBPixel<float> pixel; pixel = 1.0f, 0.0f, .5f;
RGBPixel<char> pixelArray[2];
```

(continues on next page)

(continued from previous page)

```
pixelArray[0] = 255, 255, 255;
pixelArray[1] = 255, 255, 244;
```

Since RGBPixel is a subclass of Array, you can access its components as: pixel[0], pixel[1], pixel[2]

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create An RGB image

See itk::RGBPixel for additional documentation.

### Create Another

### Synopsis

Copy/duplicate a filter

### Results

Output:

```
2
```

### Code

### C++

```
#include "itkAbsImageFilter.h"
#include "itkImage.h"

template <class TImage>
void
CreateImage(typename TImage::Pointer image)
{
  using ImageType = TImage;
  typename ImageType::IndexType start;
  start.Fill(0);

  typename ImageType::SizeType size;
  size.Fill(2);

  typename ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(-2);
}
```

(continues on next page)

(continued from previous page)

```
int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = double;

  using ImageType = itk::Image<PixelType, Dimension>;

  using FilterType = itk::AbsImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  FilterType::Pointer filter2 = dynamic_cast<FilterType *>(filter->CreateAnother().
↪GetPointer());

  ImageType::Pointer image = ImageType::New();
  CreateImage<ImageType>(image);

  filter2->SetInput(image);
  filter2->Update();

  itk::Index<Dimension> index;
  index.Fill(0);

  std::cout << filter2->GetOutput()->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

**class Object** : **public** itk::*LightObject*

Base class for most ITK classes.

Object is the second-highest level base class for most itk objects. It extends the base object functionality of LightObject by implementing callbacks (via object/observer), debug flags/methods, and modification time tracking. Most ITK classes should be a subclass of Object due to the need to keep track of modified time.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Copy Another

Subclassed by itk::BinaryImageToLevelSetImageAdaptorBase< TInput, MalcolmSparseLevelSetImage< TInput::ImageDimension > >, itk::BinaryImageToLevelSetImageAdaptorBase< TInput, ShiSparseLevelSetImage< TInput::ImageDimension > >, itk::BinaryImageToLevelSetImageAdaptorBase< TInput, TOutput >, itk::BinaryImageToLevelSetImageAdaptorBase< TInput, WhitakerSparseLevelSetImage< TOutput, TInput::ImageDimension > >, itk::BinaryImageToLevelSetImageAdaptorBase< TInputImage, LevelSetDenseImage< TLevelSetImage > >, itk::CenteredTransformInitializer< VersorRigid3DTransform< double >, TFixedImage, TMovingImage >, itk::Function::ConvergenceMonitoringFunction< TScalar, TScalar >, itk::CostFunctionTemplate< TParametersValueType >, itk::DomainThreader< itk::ThreadedIndexedContainerPartitioner, itk::GradientDescentOptimizerBasev4Template >, itk::DomainThreader< itk::ThreadedIndexedContainerPartitioner, itk::QuasiNewtonOptimizerv4Template >, itk::DomainThreader< TDomainPartitioner, TImageToImageMetricv4 >, itk::DomainThreader< TDomainPartitioner, TJointHistogramMetric >, itk::DomainThreader< ThreadedImageRegionPartitioner< TIm-

age::ImageDimension >, TLevelSetEvolution >, itk::DomainThreader< ThreadedImageRegionPartitioner< TImageToImageMetricv4::VirtualImageDimension >, TImageToImageMetricv4 >, itk::DomainThreader< ThreadedImageRegionPartitioner< TJointHistogramMetric::VirtualImageDimension >, TJointHistogramMetric >, itk::DomainThreader< ThreadedIndexedContainerPartitioner, GradientDescentOptimizerBasev4Template< TInternalComputationValueType > >, itk::DomainThreader< ThreadedIndexedContainerPartitioner, QuasiNewtonOptimizerv4Template< TInternalComputationValueType > >, itk::DomainThreader< ThreadedIndexedContainerPartitioner, TImageToImageMetricv4 >, itk::DomainThreader< ThreadedIndexedContainerPartitioner, TJointHistogramMetric >, itk::DomainThreader< ThreadedIteratorRangePartitioner< TLevelSetEvolution::DomainMapImageFilterType::DomainMapType::const_iterator >, TLevelSetEvolution >, itk::DomainThreader< ThreadedIteratorRangePartitioner< WhitakerSparseLevelSetImage< TOutput, VDimension >::LayerConstIterator >, TLevelSetEvolution >, itk::FunctionBase< Array< double >, double >, itk::FunctionBase< ContinuousIndex< TCoordRep, VSpaceDimension >, Array< double > >, itk::FunctionBase< double, double >, itk::FunctionBase< Point< float, TInputImage::ImageDimension >, TOutput >, itk::FunctionBase< Point< TCoordRep, TImageType ::ImageDimension >, NumericTraits< TImageType ::PixelType >::RealType >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, bool >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, NumericTraits< TInputImage::PixelType >::RealType >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, TInputImage::PixelType >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, TOutput >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, TOutputType >, itk::FunctionBase< Point< TCoordRep, TInputImage::ImageDimension >, vnl_matrix< NumericTraits< TInputImage::PixelType::ValueType >::RealType > >, itk::FunctionBase< Point< TCoordRep, VSpaceDimension >, double >, itk::FunctionBase< Point< TOutput, TInputImage::ImageDimension >, SymmetricSecondRankTensor< TOutput, TInputImage::ImageDimension > >, itk::FunctionBase< Point< TOutput, TInputImage::ImageDimension >, TOutput >, itk::FunctionBase< Point< TOutput, TInputImage::ImageDimension >, Vector< TOutput, TInputImage::ImageDimension > >, itk::FunctionBase< TFunction::InputType, bool >, itk::FunctionBase< TInput, bool >, itk::FunctionBase< TInputPointSet::PointType, TOutput >, itk::FunctionBase< TMeasurementVector, double >, itk::FunctionBase< TMesh, TMesh::EdgeListPointerType >, itk::FunctionBase< TPointSet ::PointType, TOutput >, itk::FunctionBase< TRealValueType, TRealValueType >, itk::FunctionBase< TVector, double >, itk::GPUReduction< float >, itk::GPUReduction< int >, itk::HistogramAlgorithmBase< THistogram >, itk::ArchetypeSeriesFileNames, itk::BinaryImageToLevelSetImageAdaptorBase< TInputImage, TLevelSet >, itk::BoundingBox< TPointIdentifier, VPointDimension, TCoordRep, TPointsContainer >, itk::BSplineTransformInitializer< TTransform, TImage >, itk::BuildInformation, itk::ByteSwapper< T >, itk::CastSpatialObjectFilter< ObjectDimension >, itk::CenteredTransformInitializer< TTransform, TFixedImage, TMovingImage >, itk::ColorTable< TPixel >, itk::Command, itk::CostFunctionTemplate< TInternalComputationValueType >, itk::CreateObjectFunctionBase, itk::DataObject, itk::Directory, itk::DomainThreader< TDomainPartitioner, TAssociate >, itk::DOMNode, itk::DOMNodeXMLReader, itk::DOMNodeXMLWriter, itk::DOMReader< TOutput >, itk::DOMWriter< TInput >, itk::DynamicLoader, itk::FastMarchingImageToNodePairContainerAdaptor< TInput, TOutput, TImage >, itk::FFTWGlobalConfiguration, itk::Function::ColormapFunction< TScalar, TRGBPixel >, itk::Function::ConvergenceMonitoringFunction< TScalar, TEnergyValue >, itk::FunctionBase< TInput, TOutput >, itk::GPUDataManager, itk::GPUReduction< TElement >, itk::HistogramAlgorithmBase< TInputHistogram >, itk::ImageContainerInterface< TElementIdentifier, TElement >, itk::ImageDuplicator< TInputImage >, itk::ImageIOFactory, itk::ImageMomentsCalculator< TImage >, itk::ImagePCADecompositionCalculator< TInputImage, TBasisImage >, itk::ImageRegionSplitterBase, itk::ImportImageContainer< TElementIdentifier, TElement >, itk::IndexedContainerInterface< TElementIdentifier, TElement >, itk::KappaSigmaThresholdImageCalculator< TInputImage, TMaskImage >, itk::LandmarkBasedTransformInitializer< TTransform, TFixedImage, TMovingImage >, itk::LevelSetContainerBase< TIdentifier, TLevelSet >, itk::LevelSetDomainPartitionBase< TDomain >, itk::LevelSetEquationContainer< TTermContainer >, itk::LevelSetEquationTermBase< TInputImage, TLevelSetContainer >, itk::LevelSetEquationTermContainer< TInputImage, TLevelSetContainer >, itk::LevelSetEvolutionBase< TEquationContainer, TLevelSet >, itk::LightProcessObject, itk::LoggerBase, itk::LoggerManager, itk::LogOutput, itk::MapContainer< TElementIdentifier, TElement >, itk::MeshIOFactory, itk::MetaConverterBase< VDimension >, itk::MetaSceneConverter< NDimensions, PixelType, TMeshTraits

>, itk::MinimumMaximumImageCalculator< TInputImage >, itk::MRASlabIdentifier< TInputImage >, itk::MultiThreaderBase, itk::NumericSeriesFileNames, itk::ObjectFactoryBase, itk::ObjectStore< TObjectType >, itk::ObjectToObjectOptimizerBaseTemplate< TInternalComputationValueType >, itk::OctreeBase, itk::Optimizer, itk::OptimizerParameterScalesEstimatorTemplate< TInternalComputationValueType >, itk::OutputWindow, itk::PointsLocator< TPointsContainer >, itk::ProcessObject, itk::ProgressAccumulator, itk::QuadEdgeMeshDecimationCriterion< TMesh, TElement, TMeasure, TPriorityQueueWrapper >, itk::QuadEdgeMeshFunctionBase< TMesh, TOutput >, itk::QuadEdgeMeshTopologyChecker< TMesh >, itk::RealTimeClock, itk::RegularExpressionSeriesFileNames, itk::RingBuffer< TElement >, itk::RobustAutomaticThresholdCalculator< TInputImage, TGradientImage >, itk::SegmentationBorder, itk::SegmentationRegion, itk::SimplexMeshVolumeCalculator< TInputMesh >, itk::SparseFieldLayer< TNodeType >, itk::SpatialObjectDuplicator< TInputSpatialObject >, itk::SpatialObjectReader< NDimensions, PixelType, TMeshTraits >, itk::SpatialObjectToImageStatisticsCalculator< TInputImage, TInputSpatialObject, TSampleDimension >, itk::SpatialObjectWriter< NDimensions, PixelType, TMeshTraits >, itk::Statistics::DecisionRule, itk::Statistics::DenseFrequencyContainer2, itk::Statistics::ExpectationMaximizationMixtureModelEstimator< TSample >, itk::Statistics::KdTree< TSample >, itk::Statistics::KdTreeBasedKmeansEstimator< TKdTree >, itk::Statistics::KdTreeGenerator< TSample >, itk::Statistics::MixtureModelComponentBase< TSample >, itk::Statistics::ProbabilityDistribution, itk::Statistics::RandomVariateGeneratorBase, itk::Statistics::ScalarImageToHistogramGenerator< TImageType >, itk::Statistics::SparseFrequencyContainer2, itk::Statistics::SubsamplerBase< TSample >, itk::StoppingCriterionBase, itk::testhelper::ImageRegistrationMethodImageSource< TFixedPixelType, TMovingPixelType, NDimension >, itk::ThreadedDomainPartitioner< TDomain >, itk::ThreadPool, itk::TransformBaseTemplate< TParametersValueType >, itk::TransformIOFactoryTemplate< TParametersValueType >, itk::TransformParametersAdaptorBase< TTransform >, itk::UpdateMalcolmSparseLevelSet< VDimension, TEquationContainer >, itk::UpdateShiSparseLevelSet< VDimension, TEquationContainer >, itk::UpdateWhitakerSparseLevelSet< VDimension, TLevelSetValueType, TEquationContainer >, itk::ValarrayImageContainer< TElementIdentifier, TElement >, itk::VectorContainer< TElementIdentifier, TElement >, itk::Version, itk::VideoIOFactory, itk::VTKPolyDataWriter< TInputMesh >, itk::WarpHarmonicEnergyCalculator< TInputImage >, itk::LevelSetContainerBase< TIdentifier, LevelSetDenseImage< TImage > >, itk::LevelSetDomainPartitionBase< TImage >, itk::LevelSetDomainPartitionBase< TMesh >, itk::LevelSetEquationTermBase< TInput, TLevelSetContainer >, itk::LevelSetEvolutionBase< TEquationContainer, LevelSetDenseImage< TImage > >, itk::LevelSetEvolutionBase< TEquationContainer, MalcolmSparseLevelSetImage< VDimension > >, itk::LevelSetEvolutionBase< TEquationContainer, ShiSparseLevelSetImage< VDimension > >, itk::LevelSetEvolutionBase< TEquationContainer, WhitakerSparseLevelSetImage< TOutput, VDimension > >, itk::MetaConverterBase< NDimensions >, itk::ObjectToObjectOptimizerBaseTemplate< double >, itk::OptimizerParameterScalesEstimatorTemplate< TMetric::ParametersValueType >, itk::QuadEdgeMeshFunctionBase< TMesh, TQEType * >, itk::QuadEdgeMeshFunctionBase< TMesh, TQEType::OriginRefType >, itk::ThreadedDomainPartitioner< ImageRegion< VDimension > >, itk::ThreadedDomainPartitioner< Index< 2 > >, itk::ThreadedDomainPartitioner< ThreadedIteratorRangePartitionerDomain< TIterator > >, itk::TransformParametersAdaptorBase< Transform< TTransform::ScalarType, TTransform::InputSpaceDimension, TTransform::OutputSpaceDimension > >, itk::VectorContainer< TElementIdentifier, TElementWrapper >, itk::VTKPolyDataWriter< TMesh >

See itk::Object for additional documentation.

### Create Another Instance of an Image

### Synopsis

Create another instance of an image.

### Results

Output:

```
Image type FloatScalarImageType
Image type FloatScalarImageType
Image type FloatScalarImageType
```

### Code

### C++

```
#include "itkImage.h"

using FloatScalarImageType = itk::Image<float, 2>;

itk::ImageBase<2>::Pointer
CreateImageWithSameType(const itk::ImageBase<2> * input);
void
OutputImageType(const itk::ImageBase<2> * input);

int
main(int, char *[])
{
  FloatScalarImageType::Pointer floatImage = FloatScalarImageType::New();
  itk::ImageBase<2>::Pointer    floatCopy = CreateImageWithSameType(floatImage);
  OutputImageType(floatCopy);

  return EXIT_SUCCESS;
}

itk::ImageBase<2>::Pointer
CreateImageWithSameType(const itk::ImageBase<2> * input)
{
  OutputImageType(input);
  itk::LightObject::Pointer objectCopyLight = input->CreateAnother();

  itk::ImageBase<2>::Pointer objectCopy = dynamic_cast<itk::ImageBase<2> *>
↪(objectCopyLight.GetPointer());
  OutputImageType(objectCopy);
  return objectCopy;
}

void
OutputImageType(const itk::ImageBase<2> * input)
{
  if (dynamic_cast<const FloatScalarImageType *>(input))
```

(continues on next page)

(continued from previous page)

```
  {
    std::cout << "Image type FloatScalarImageType" << std::endl;
  }
  else
  {
    std::cout << "Image is Invalid type!" << std::endl;
  }
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image

- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Create Derivative Kernel

### Synopsis

Create a derivative kernel.

### Results

Output:

```
Size: [3, 3]
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffeec8f19e8, begin = 0x7f822c2a8f00,␣
↪size=9 }
[-1, -1] 0
[0, -1] 0
[1, -1] 0
[-1, 0] 0.5
[0, 0] 0
[1, 0] -0.5
[-1, 1] 0
[0, 1] 0
[1, 1] 0
```

### Code

### Python

```
#!/usr/bin/env python

import itk

derivativeOperator = itk.DerivativeOperator[itk.F, 2]()
derivativeOperator.SetDirection(0)  # Create the operator for the X axis derivative
radius = itk.Size[2]()
radius.Fill(1)
```

(continues on next page)

(continued from previous page)

```
derivativeOperator.CreateToRadius(radius)

print("Size: " + str(derivativeOperator.GetSize()))

print(derivativeOperator)

for i in range(9):
    print(
        str(derivativeOperator.GetOffset(i))
        + " "
        + str(derivativeOperator.GetElement(i))
    )
```

### C++

```
#include <itkDerivativeOperator.h>

int
main(int, char *[])
{
  using DerivativeOperatorType = itk::DerivativeOperator<float, 2>;
  DerivativeOperatorType derivativeOperator;
  derivativeOperator.SetDirection(0); // Create the operator for the X axis derivative
  itk::Size<2> radius;
  radius.Fill(1);
  derivativeOperator.CreateToRadius(radius);

  std::cout << "Size: " << derivativeOperator.GetSize() << std::endl;

  std::cout << derivativeOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << derivativeOperator.GetOffset(i) << " " << derivativeOperator.
↪GetElement(i) << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class DerivativeOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>

A NeighborhoodOperator for taking an n-th order derivative at a pixel.

DerivativeOperator's coefficients are a tightest-fitting convolution kernel for calculating the n-th order directional derivative at a pixel. DerivativeOperator is a directional NeighborhoodOperator that should be applied to a Neighborhood or NeighborhoodPointer using the inner product method.

An example operator to compute X derivatives of a 2D image can be created with:

```
using DerivativeOperatorType = itk::DerivativeOperator<float, 2>;
DerivativeOperatorType derivativeOperator;
```

(continues on next page)

(continued from previous page)

```
derivativeOperator.SetDirection(0); // X dimension
itk::Size<2> radius;
radius.Fill(1); // A radius of 1 in both dimensions is a 3x3 operator
derivativeOperator.CreateToRadius(radius);
```

and creates a kernel that looks like:

```
0         0 0
0.5   0   -0.5
0     0   0
```

**Note** DerivativeOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**See** NeighborhoodOperator

**See** Neighborhood

**See** ForwardDifferenceOperator

**See** BackwardDifferenceOperator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Derivative Kernel

See itk::DerivativeOperator for additional documentation.

### Create Forward Difference Kernel

### Synopsis

Create a forward difference kernel.

### Results

Output:

```
Size: [3, 3]
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffee23339e0, begin = 0x7fd6921a8f60,␣
↪size=9 }
[-1, -1] 0
[0, -1] 0
[1, -1] 0
[-1, 0] 0
[0, 0] -1
[1, 0] 1
[-1, 1] 0
[0, 1] 0
[1, 1] 0
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

forwardDifferenceOperator = itk.ForwardDifferenceOperator[itk.F, 2]()
forwardDifferenceOperator.SetDirection(
    0
)  # Create the operator for the X axis derivative
radius = itk.Size[2]()
radius.Fill(1)
forwardDifferenceOperator.CreateToRadius(radius)

print("Size: " + str(forwardDifferenceOperator.GetSize()))

print(forwardDifferenceOperator)

for i in range(9):
    print(
        str(forwardDifferenceOperator.GetOffset(i))
        + " "
        + str(forwardDifferenceOperator.GetElement(i))
    )
```

### C++

```cpp
#include <itkForwardDifferenceOperator.h>

int
main(int, char *[])
{
  using ForwardDifferenceOperatorType = itk::ForwardDifferenceOperator<float, 2>;
  ForwardDifferenceOperatorType forwardDifferenceOperator;
  forwardDifferenceOperator.SetDirection(0); // Create the operator for the X axis␣
↪derivative
  itk::Size<2> radius;
  radius.Fill(1);
  forwardDifferenceOperator.CreateToRadius(radius);

  std::cout << "Size: " << forwardDifferenceOperator.GetSize() << std::endl;

  std::cout << forwardDifferenceOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << forwardDifferenceOperator.GetOffset(i) << " " <<␣
↪forwardDifferenceOperator.GetElement(i) << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class ForwardDifferenceOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>
Operator whose inner product with a neighborhood returns a "half" derivative at the center of the neighborhood.

ForwardDifferenceOperator uses forward differences i.e. F(x+1) - F(x) to calculate a "half" derivative useful, among other things, in solving differential equations. It is a directional NeighborhoodOperator that should be applied to a Neighborhood using the inner product.

**Note** ForwardDifferenceOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Forward Difference Kernel

See itk::ForwardDifferenceOperator for additional documentation.

### Create Gaussian Derivative Kernel

### Synopsis

Create a Gaussian derivative kernel.

### Results

Output:

```
Size: [3, 3]
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffee86419a8, begin = 0x7fb16bc5cc50,␣
↪size=9 }
[-1, -1] 0
[0, -1] 0
[1, -1] 0
[-1, 0] 0.208375
[0, 0] 0
[1, 0] -0.208375
[-1, 1] 0
[0, 1] 0
[1, 1] 0
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

gaussianDerivativeOperator = itk.GaussianDerivativeOperator[itk.F, 2]()
gaussianDerivativeOperator.SetDirection(
    0
)  # Create the operator for the X axis derivative
radius = itk.Size[2]()
radius.Fill(1)
gaussianDerivativeOperator.CreateToRadius(radius)

print("Size: " + str(gaussianDerivativeOperator.GetSize()))

print(gaussianDerivativeOperator)

for i in range(9):
    print(
        str(gaussianDerivativeOperator.GetOffset(i))
        + " "
        + str(gaussianDerivativeOperator.GetElement(i))
    )
```

### C++

```cpp
#include <itkGaussianDerivativeOperator.h>

int
main(int, char *[])
{
  using GaussianDerivativeOperatorType = itk::GaussianDerivativeOperator<float, 2>;
  GaussianDerivativeOperatorType gaussianDerivativeOperator;
  gaussianDerivativeOperator.SetDirection(0); // Create the operator for the X axis␣
↪derivative
  itk::Size<2> radius;
  radius.Fill(1);
  gaussianDerivativeOperator.CreateToRadius(radius);

  std::cout << "Size: " << gaussianDerivativeOperator.GetSize() << std::endl;

  std::cout << gaussianDerivativeOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << gaussianDerivativeOperator.GetOffset(i) << " " <<␣
↪gaussianDerivativeOperator.GetElement(i)
              << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class GaussianDerivativeOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>

A NeighborhoodOperator whose coefficients are a one dimensional, discrete derivative Gaussian kernel.

GaussianDerivativeOperator can be used to calculate Gaussian derivatives by taking its inner product with to a Neighborhood (NeighborhooIterator) that is swept across an image region. It is a directional operator. N successive applications oriented along each dimensional direction will calculate separable, efficient, N-D Gaussian derivatives of an image region.

GaussianDerivativeOperator takes three parameters:

(1) The floating-point variance of the desired Gaussian function.

(2) The order of the derivative to be calculated (zero order means it performs only smoothing as a standard itk::GaussianOperator)

(3) The "maximum error" allowed in the discrete Gaussian function. "Maximum errror" is defined as the difference between the area under the discrete Gaussian curve and the area under the continuous Gaussian. Maximum error affects the Gaussian operator size. Care should be taken not to make this value too small relative to the variance lest the operator size become unreasonably large.

References: The Gaussian kernel contained in this operator was described by Tony Lindeberg (Discrete Scale-Space Theory and the Scale-Space Primal Sketch. Dissertation. Royal Institute of Technology, Stockholm, Sweden. May 1991.).

This implementation is derived from the Insight Journal paper: https://www.insight-journal.org/browse/publication/179

**Author** Ivan Macia, Vicomtech, Spain, https://www.vicomtech.org/en

**Note** GaussianDerivativeOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**See** GaussianOperator

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**See** Neighborhood

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Gaussian Derivative Kernel

See itk::GaussianDerivativeOperator for additional documentation.

### Create Gaussian Kernel

### Synopsis

Create a Gaussian kernel.

### Results

Output:

```
Size: [3, 3]
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffee598a9d8, begin = 0x7f7f502572a0,␣
↪size=9 }
[-1, -1] 0
[0, -1] 0
[1, -1] 0
[-1, 0] 0.208375
[0, 0] 0.466801
[1, 0] 0.208375
[-1, 1] 0
[0, 1] 0
[1, 1] 0
```

### Code

### Python

```
#!/usr/bin/env python

import itk

gaussianOperator = itk.GaussianOperator[itk.F, 2]()
gaussianOperator.SetDirection(0)  # Create the operator for the X axis derivative
radius = itk.Size[2]()
radius.Fill(1)
gaussianOperator.CreateToRadius(radius)

print("Size: " + str(gaussianOperator.GetSize()))

print(gaussianOperator)

for i in range(9):
    print(
        str(gaussianOperator.GetOffset(i)) + " " + str(gaussianOperator.GetElement(i))
    )
```

### C++

```
#include <itkGaussianOperator.h>

int
main(int, char *[])
{
  using GaussianOperatorType = itk::GaussianOperator<float, 2>;
  GaussianOperatorType gaussianOperator;
  gaussianOperator.SetDirection(0); // Create the operator for the X axis derivative
  itk::Size<2> radius;
  radius.Fill(1);
  gaussianOperator.CreateToRadius(radius);

  std::cout << "Size: " << gaussianOperator.GetSize() << std::endl;

  std::cout << gaussianOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << gaussianOperator.GetOffset(i) << " " << gaussianOperator.
↪GetElement(i) << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class GaussianOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>

A NeighborhoodOperator whose coefficients are a one dimensional, discrete Gaussian kernel.

GaussianOperator can be used to perform Gaussian blurring by taking its inner product with a Neighborhood (NeighborhoodIterator) that is swept across an image region. It is a directional operator. N successive applications oriented along each dimensional direction will effect separable, efficient, N-D Gaussian blurring of an image region.

GaussianOperator takes two parameters:

(1) The floating-point variance of the desired Gaussian function.

(2) The "maximum error" allowed in the discrete Gaussian function. "Maximum errror" is defined as the difference between the area under the discrete Gaussian curve and the area under the continuous Gaussian. Maximum error affects the Gaussian operator size. Care should be taken not to make this value too small relative to the variance lest the operator size become unreasonably large.

References: The Gaussian kernel contained in this operator was described by Tony Lindeberg (Discrete Scale-Space Theory and the Scale-Space Primal Sketch. Dissertation. Royal Institute of Technology, Stockholm, Sweden. May 1991.).

**Note** GaussianOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**See** Neighborhood

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Gaussian Kernel

See itk::GaussianOperator for additional documentation.

### Create Laplacian Kernel

### Synopsis

Create a Laplacian kernel.

### Results

Output:

```
Size: [3, 3]
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffee97e29e0, begin = 0x7f87798a8810,␣
↪size=9 }
[-1, -1] 0
[0, -1] 1
[1, -1] 0
[-1, 0] 1
[0, 0] -4
[1, 0] 1
[-1, 1] 0
[0, 1] 1
[1, 1] 0
```

### Code

### Python

```
#!/usr/bin/env python

import itk

laplacianOperator = itk.LaplacianOperator[itk.F, 2]()
radius = itk.Size[2]()
radius.Fill(1)
laplacianOperator.CreateToRadius(radius)

print("Size: " + str(laplacianOperator.GetSize()))

print(laplacianOperator)

for i in range(laplacianOperator.GetSize()[0] * laplacianOperator.GetSize()[1]):
```

(continues on next page)

(continued from previous page)

```
    print(
        str(laplacianOperator.GetOffset(i)) + " " + str(laplacianOperator.
↪GetElement(i))
    )
```

### C++

```
#include <itkLaplacianOperator.h>

int
main(int, char *[])
{
  using LaplacianOperatorType = itk::LaplacianOperator<float, 2>;
  LaplacianOperatorType laplacianOperator;
  itk::Size<2>          radius;
  radius.Fill(1);
  laplacianOperator.CreateToRadius(radius);

  std::cout << "Size: " << laplacianOperator.GetSize() << std::endl;

  std::cout << laplacianOperator << std::endl;

  for (unsigned int i = 0; i < laplacianOperator.GetSize()[0] * laplacianOperator.
↪GetSize()[1]; i++)
  {
    std::cout << laplacianOperator.GetOffset(i) << " " << laplacianOperator.
↪GetElement(i) << std::endl;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class LaplacianOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>

A NeighborhoodOperator for use in calculating the Laplacian at a pixel.

A NeighborhoodOperator for use in calculating the Laplacian at a pixel. The LaplacianOperator's coefficients are a tightest-fitting convolution kernel.

For example, the simplest Laplacian Operator for 2D has the form:

```
0   1   0
1  -4   1
0   1   0
```

The LaplacianOperator is a non-directional NeighborhoodOperator that should be applied to a Neighborhood or NeighborhoodIterator using an inner product method (itkNeighborhoodInnerProduct). To initialize the operator, you need call CreateOperator() before using it.

By default the operator will be created for an isotropic image, but you can modify the operator to handle different pixel spacings by calling SetDerivativeScalings. The argument to SetDerivativeScalings is an array of

doubles that is of length VDimension (the dimensionality of the image). Make sure to use 1/pixel_spacing to properly scale derivatives.

**Note** LaplacianOperator does not have any user-declared "special member function" for copy, move, or destruction, following the C++ Rule of Zero: the compiler will generate them if necessary.

**See** NeighborhoodOperator

**See** Neighborhood

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Laplacian Kernel

See itk::LaplacianOperator for additional documentation.

### Create Sobel Kernel

### Synopsis

Create the Sobel kernel.

### Results

Output:

```
Neighborhood:
Radius:[1, 1]
Size:[3, 3]
DataBuffer:NeighborhoodAllocator { this = 0x7ffee3e84a00, begin = 0x7fca93256b10,␣
↪size=9 }
-1
0
1
-2
0
2
-1
0
1
```

### Code

### Python

```
#!/usr/bin/env python

import itk

sobelOperator = itk.SobelOperator[itk.F, 2]()
sobelOperator.SetDirection(0)  # Create the operator for the X axis derivative
```

(continues on next page)

(continued from previous page)

```
radius = itk.Size[2]()
radius.Fill(1)
sobelOperator.CreateToRadius(radius)

print(sobelOperator)

for i in range(9):
    print(sobelOperator.GetElement(i))
```

### C++

```
#include <itkSobelOperator.h>

int
main(int, char *[])
{
  using SobelOperatorType = itk::SobelOperator<float, 2>;
  SobelOperatorType sobelOperator;
  sobelOperator.SetDirection(0); // Create the operator for the X axis derivative
  itk::Size<2> radius;
  radius.Fill(1);
  sobelOperator.CreateToRadius(radius);

  std::cout << sobelOperator << std::endl;

  for (unsigned int i = 0; i < 9; i++)
  {
    std::cout << sobelOperator.GetElement(i) << std::endl;
  }
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class SobelOperator** : **public** itk::*NeighborhoodOperator*<*TPixel*, *VDimension*, *TAllocator*>

A NeighborhoodOperator for performing a directional Sobel edge-detection operation at a pixel location.

SobelOperator is a directional NeighborhoodOperator that should be applied a NeighborhoodIterator using the NeighborhoodInnerProduct method. To create the operator:

1) Set the direction by calling

```
SetDirection
```

2) call

```
itk::Size<2> radius;
radius.Fill(1);
sobelOperator.CreateToRadius(radius);
```

3) You may optionally scale the coefficients of this operator using the

```
ScaleCoefficients
```

method. This is useful if you want to take the spacing of the image into account when computing the edge strength. Apply the scaling only after calling to

```
CreateToRadius
```

.

The Sobel Operator in vertical direction for 2 dimensions is

```
*               -1  -2  -1
*               0    0   0
*               1    2   1
*
*
```

The Sobel Operator in horizontal direction is for 2 dimensions is

```
*               -1   0   1
*               -2   0   2
*               -1   0   1
*
```

The current implementation of the Sobel operator is for 2 and 3 dimensions only. The ND version is planned for future releases.

The extension to 3D is from the publication "Irwin Sobel. An Isotropic 3x3x3 Volume Gradient Operator.

Technical report, Hewlett-Packard Laboratories, April 1995."

The Sobel operator in 3D has the kernel

```
* -1 -3 -1    0 0 0   1 3 1
* -3 -6 -3    0 0 0   3 6 3
* -1 -3 -1    0 0 0   1 3 1
*
*     x-1        x       x+1
*
```

The `x` kernel is just rotated as required to obtain the kernel in the `y` and `z` directions.

**Note** SobelOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**See** NeighborhoodOperator

**See** Neighborhood

**See** ForwardDifferenceOperator

**See** BackwardDifferenceOperator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Sobel Kernel

See itk::SobelOperator for additional documentation.

### Create Vector Image

### Synopsis

Create a vector image.

### Results

Output:

```
pixel (1,1) = [0, 0]
pixel (1,1) = [1.1, 2.2]
```

### Code

### C++

```
#include "itkVectorImage.h"

int
main(int, char *[])
{
  // Create an image
  using ImageType = itk::VectorImage<float, 2>;

  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(2);

  ImageType::RegionType region(start, size);

  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->SetVectorLength(2);
  image->Allocate();

  ImageType::IndexType pixelIndex;
  pixelIndex[0] = 1;
  pixelIndex[1] = 1;

  ImageType::PixelType pixelValue = image->GetPixel(pixelIndex);

  std::cout << "pixel (1,1) = " << pixelValue << std::endl;

  using VariableVectorType = itk::VariableLengthVector<double>;
  VariableVectorType variableLengthVector;
  variableLengthVector.SetSize(2);
  variableLengthVector[0] = 1.1;
  variableLengthVector[1] = 2.2;

  image->SetPixel(pixelIndex, variableLengthVector);
```

(continues on next page)

(continued from previous page)

```
  std::cout << "pixel (1,1) = " << pixelValue << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 3>
**class VectorImage** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional vector image class.

This class differs from Image in that it is intended to represent multiple images. Each pixel represents *k* measurements, each of datatype *TPixel*. The memory organization of the resulting image is as follows: ... Pi0 Pi1 Pi2 Pi3 P(i+1)0 P(i+1)1 P(i+1)2 P(i+1)3 P(i+2)0 ... where Pi0 represents the 0th measurement of the pixel at index i.

Conceptually, a `VectorImage< TPixel, 3 >` is the same as a `Image< VariableLengthVector< TPixel >, 3 >`. The difference lies in the memory organization. The latter results in a fragmented organization with each location in the Image holding a pointer to an `VariableLengthVector` holding the actual pixel. The former stores the *k* pixels instead of a pointer reference, which apart from avoiding fragmentation of memory also avoids storing a 8 bytes of pointer reference for each pixel. The parameter *k* can be set using `SetVectorLength`.

The API of the class is such that it returns a pixeltype VariableLengthVector< TPixel > when queried, with the data internally pointing to the buffer. (the container does not manage the memory). Similarly SetPixel calls can be made with VariableLengthVector< TPixel >.

The API of this class is similar to Image.

**Caveats:** When using Iterators on this image, you cannot use the it.Value(). You must use Set/Get() methods instead.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**See** DefaultVectorPixelAccessor

**See** DefaultVectorPixelAccessorFunctor

**See** VectorImageToImagePixelAccessor

**See** VectorImageToImageAdaptor

**See** Image

**See** ImportImageContainer

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cast Vector Image To Another Type
- Create Vector Image
- Neighborhood Iterator On Vector Image

See itk::VectorImage for additional documentation.

### Crop Image by Specifying Region

### Synopsis

Crop an image by specifying the region to keep.

### Results

Output:

```
Image largest region: ImageRegion (0x7f886fc0de00)
Dimension: 2
Index: [0, 0]
Size: [10, 10]

desiredRegion: ImageRegion (0x7ffeef707978)
Dimension: 2
Index: [3, 3]
Size: [4, 4]

new largest region: ImageRegion (0x7f886fc0eb00)
Dimension: 2
Index: [3, 3]
Size: [4, 4]

new: 2
Original: 5
```

### Code

### C++

```
// This is different from CropImageFilter only in the way
// that the region to crop is specified.
#include "itkImage.h"
#include <itkImageFileReader.h>
#include <itkExtractImageFilter.h>

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned char, 2>;

  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region(start, size);
```

(continues on next page)

(continued from previous page)

```
  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(5);

  std::cout << "Image largest region: " << image->GetLargestPossibleRegion() << std::
↪endl;

  ImageType::IndexType desiredStart;
  desiredStart.Fill(3);

  ImageType::SizeType desiredSize;
  desiredSize.Fill(4);

  ImageType::RegionType desiredRegion(desiredStart, desiredSize);

  std::cout << "desiredRegion: " << desiredRegion << std::endl;

  using FilterType = itk::ExtractImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetExtractionRegion(desiredRegion);
  filter->SetInput(image);
#if ITK_VERSION_MAJOR >= 4
  filter->SetDirectionCollapseToIdentity(); // This is required.
#endif
  filter->Update();

  ImageType::Pointer output = filter->GetOutput();
  output->DisconnectPipeline();
  output->FillBuffer(2);

  itk::Index<2> index;
  index.Fill(5);

  std::cout << "new largest region: " << output->GetLargestPossibleRegion() << std::
↪endl;
  std::cout << "new: " << (int)output->GetPixel(index) << std::endl;
  std::cout << "Original: " << (int)image->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ExtractImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Decrease the image size by cropping the image to the selected region bounds.

ExtractImageFilter changes the image boundary of an image by removing pixels outside the target region. The target region must be specified.

ExtractImageFilter also collapses dimensions so that the input image may have more dimensions than the output image (i.e. 4-D input image to a 3-D output image). To specify what dimensions to collapse, the Extraction-Region must be specified. For any dimension dim where ExtractionRegion.Size[dim] = 0, that dimension is collapsed. The index to collapse on is specified by ExtractionRegion.Index[dim]. For example, we have a image

4D = a 4x4x4x4 image, and we want to get a 3D image, 3D = a 4x4x4 image, specified as [x,y,z,2] from 4D (i.e. the 3rd "time" slice from 4D). The ExtractionRegion.Size = [4,4,4,0] and ExtractionRegion.Index = [0,0,0,2].

The number of dimension in ExtractionRegion.Size and Index must = InputImageDimension. The number of non-zero dimensions in ExtractionRegion.Size must = OutputImageDimension.

The output image produced by this filter will have the same origin as the input image, while the ImageRegion of the output image will start at the starting index value provided in the ExtractRegion parameter. If you are looking for a filter that will re-compute the origin of the output image, and provide an output image region whose index is set to zeros, then you may want to use the RegionOfInterestImageFilter. The output spacing is is simply the collapsed version of the input spacing.

Determining the direction of the collapsed output image from an larger dimensional input space is an ill defined problem in general. It is required that the application developer select the desired transformation strategy for collapsing direction cosines. It is REQUIRED that a strategy be explicitly requested (i.e. there is no working default). Direction Collapsing Strategies: 1) DirectionCollapseToUnknown(); This is the default and the filter can not run when this is set. The reason is to explicitly force the application developer to define their desired behavior. 1) DirectionCollapseToIdentity(); Output has identity direction no matter what 2) DirectionCollapseToSubmatrix(); Output direction is the sub-matrix if it is positive definite, else throw an exception.

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Note** This filter is derived from InPlaceImageFilter. When the input to this filter matched the output requested region, like with streaming filter for input, then setting this filter to run in-place will result in no copying of the bulk pixel data.

**See** CropImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Crop Image By Specifying Region

Subclassed by itk::CropImageFilter< TInputImage, TOutputImage >

See itk::ExtractImageFilter for additional documentation.

### Deep Copy Image

### Synopsis

Deep copy an image.

### Results

**Note:** No output is printed, this example simply displays functionality.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageRegionIterator.h"

template <typename TImage>
void
DeepCopy(typename TImage::Pointer input, typename TImage::Pointer output)
{
  output->SetRegions(input->GetLargestPossibleRegion());
  output->Allocate();

  itk::ImageRegionConstIterator<TImage> inputIterator(input, input->
↪GetLargestPossibleRegion());
  itk::ImageRegionIterator<TImage>      outputIterator(output, output->
↪GetLargestPossibleRegion());

  while (!inputIterator.IsAtEnd())
  {
    outputIterator.Set(inputIterator.Get());
    ++inputIterator;
    ++outputIterator;
  }
}

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned char, 2>;
  ImageType::Pointer image1 = ImageType::New();
  ImageType::Pointer image2 = ImageType::New();

  DeepCopy<ImageType>(image1, image2);


  return 0;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>
Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that

define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Demonstrate All Operators

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Demonstrate all operators.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include <itkNeighborhoodOperator.h>

#include <itkDerivativeOperator.h>
#include <itkForwardDifferenceOperator.h>
#include <itkGaussianDerivativeOperator.h>
#include <itkGaussianOperator.h>
#include <itkImageKernelOperator.h>
#include <itkLaplacianOperator.h>
#include <itkSobelOperator.h>
#include <itkAnnulusOperator.h>
#include <itkBackwardDifferenceOperator.h>

#include <vector>

int
main(int, char *[])
{
  using DerivativeOperatorType = itk::DerivativeOperator<float, 2>;
  using ForwardDifferenceOperatorType = itk::ForwardDifferenceOperator<float, 2>;
  using GaussianDerivativeOperatorType = itk::GaussianDerivativeOperator<float, 2>;
  using GaussianOperatorType = itk::GaussianOperator<float, 2>;
  using ImageKernelOperatorType = itk::ImageKernelOperator<float, 2>;
  using LaplacianOperatorType = itk::LaplacianOperator<float, 2>;
  using SobelOperatorType = itk::SobelOperator<float, 2>;
  using AnnulusOperatorType = itk::AnnulusOperator<float, 2>;
  using BackwardDifferenceOperatorType = itk::BackwardDifferenceOperator<float, 2>;

  std::vector<itk::NeighborhoodOperator<float, 2> *> operators;
  operators.push_back(new DerivativeOperatorType);
  operators.push_back(new ForwardDifferenceOperatorType);
  operators.push_back(new GaussianDerivativeOperatorType);
  operators.push_back(new GaussianOperatorType);
  operators.push_back(new ImageKernelOperatorType);
  operators.push_back(new LaplacianOperatorType);
  operators.push_back(new SobelOperatorType);
  operators.push_back(new AnnulusOperatorType);
  operators.push_back(new BackwardDifferenceOperatorType);
```

(continues on next page)

(continued from previous page)

```
  itk::Size<2> radius;
  radius.Fill(1);

  for (auto & operatorId : operators)
  {
    operatorId->SetDirection(0); // Create the operator for the X axis derivative
    operatorId->CreateToRadius(radius);
    // std::cout << *(operators[operatorId]) << std::endl;
    // operators[operatorId]->Print(std::cout);
    // std::cout << operators[operatorId]->GetNameOfClass() << std::endl;

    for (auto i = -operatorId->GetSize()[0] / 2; i <= operatorId->GetSize()[0] / 2;␣
↪i++)
    {
      for (auto j = -operatorId->GetSize()[1] / 2; j <= operatorId->GetSize()[1] / 2;␣
↪j++)
      {
        itk::Offset<2> offset;
        offset[0] = i;
        offset[1] = j;

        unsigned int neighborhoodIndex = operatorId->GetNeighborhoodIndex(offset);
        std::cout << operatorId->GetElement(neighborhoodIndex) << " ";
      }
      std::cout << std::endl;
    }
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension**, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class NeighborhoodOperator** : **public** itk::Neighborhood<*TPixel*, *VDimension*, *TAllocator*>

Virtual class that defines a common interface to all neighborhood operator subtypes.

A NeighborhoodOperator is a set of pixel values that can be applied to a Neighborhood to perform a user-defined operation (i.e. convolution kernel, morphological structuring element). A NeighborhoodOperator is itself a specialized Neighborhood, with functionality to generate its coefficients according to user-defined parameters. Because the operator is a subclass of Neighborhood, it is a valid operand in any of the operations defined on the Neighborhood object (convolution, inner product, etc.).

NeighborhoodOperator is a pure virtual object that must be subclassed to be used. A user's subclass must implement two methods:

(1) GenerateCoefficients the algorithm that computes the scalar coefficients of the operator.

(2) Fill the algorithm that places the scalar coefficients into the memory buffer of the operator (arranges them spatially in the neighborhood).

NeighborhoodOperator supports the concept of a "directional operator." A directional operator is defined in this context to be an operator that is applied along a single dimension. Examples of this type of operator are directional derivatives and the individual, directional components of separable processes such as Gaussian smoothing.

How a NeighborhoodOperator is applied to data is up to the user who defines it. One possible use of an operator would be to take its inner product with a neighborhood of values to produce a scalar result. This process effects convolution when applied to successive neighborhoods across a region of interest in an image.

**Note** NeighborhoodOperator does not have any user-declared "special member function", following the C++ Rule of Zero: the compiler will generate them if necessary.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Demonstrate All Operators

Subclassed by itk::DerivativeOperator< TPixel, VDimension, TAllocator >, itk::ForwardDifferenceOperator< TPixel, VDimension, TAllocator >, itk::GaussianDerivativeOperator< TPixel, VDimension, TAllocator >, itk::GaussianOperator< TPixel, VDimension, TAllocator >, itk::ImageKernelOperator< TPixel, VDimension, TAllocator >, itk::LaplacianOperator< TPixel, VDimension, TAllocator >, itk::SobelOperator< TPixel, VDimension, TAllocator >

See itk::NeighborhoodOperator for additional documentation.

### Direct Warning to File

### Synopsis

Direct itk warnings to a file.

### Results

Output:

```
Look in itkMessageLog.txt for the output
```

### Code

### C++

```
#include <itkFileOutputWindow.h>
#include <itkScaleTransform.h>

int
main(int argc, char * argv[])
{
  using myFileOutputWindow = itk::FileOutputWindow;
  myFileOutputWindow::Pointer window = myFileOutputWindow::New();

  if (argc > 1)
  {
    window->SetFileName(argv[1]);
  }
  window->FlushOn();
```

(continues on next page)

(continued from previous page)

```
  // Set the singelton instance
  itk::OutputWindow::SetInstance(window);

  // Generic output
  itkGenericOutputMacro("This should be in the file: " << window->GetFileName());
  // Warning
  using TransformType = itk::ScaleTransform<float, 2>;
  TransformType::Pointer             transform = TransformType::New();
  TransformType::FixedParametersType parameters(3);
  transform->SetFixedParameters(parameters);

  std::cout << "Look in " << window->GetFileName() << " for the output" << std::endl;
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class FileOutputWindow** : **public** itk::OutputWindow
Messages sent from the system are sent to a file.

Text messages that the system should display to the user are sent to this object (or subclasses of this object) and are logged to a file.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Direct Warning To File

Subclassed by itk::XMLFileOutputWindow

See itk::FileOutputWindow for additional documentation.

### Display Image

### Synopsis

Display an image.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkRescaleIntensityImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
```

(continues on next page)

Fig. 20: Displayed Image.

(continued from previous page)

```
#endif

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType * const image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image;

  if (argc < 2)
  {
    // std::cerr << "Required: filename" << std::endl;
    // return EXIT_FAILURE;
    image = ImageType::New();
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    image = reader->GetOutput();
  }

  using RescaleFilterType = itk::RescaleIntensityImageFilter<ImageType, ImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(image);
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image.GetPointer());
  viewer.AddImage(rescaleFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType * const image)
{
  // Create an image with 2 connected components
  ImageType::IndexType corner = { { 0, 0 } };

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  ImageType::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing

data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Distance Between Indices

### Synopsis

Compute the distance between two Indices.

### Results

Output:

```
Distance: 2.82843
```

### Code

### C++

```
#include "itkPoint.h"
#include "itkIndex.h"
#include "itkMath.h"

#include <iostream>

int
main(int, char *[])
{
  itk::Index<2> pixel1;
  pixel1.Fill(2);

  itk::Index<2> pixel2;
  pixel2.Fill(4);

  itk::Point<double, 2> p1;
  p1[0] = pixel1[0];
  p1[1] = pixel1[1];

  itk::Point<double, 2> p2;
  p2[0] = pixel2[0];
  p2[1] = pixel2[1];

  double distance = p2.EuclideanDistanceTo(p1);
  std::cout << "Distance: " << distance << std::endl;
}
```

### Classes demonstrated

template<typename **TCoordRep**, unsigned int **NPointDimension** = 3>
**class Point** : **public** itk::*FixedArray*<*TCoordRep*, *NPointDimension*>

A templated class holding a geometric point in n-Dimensional space.

Point is a templated class that holds a set of coordinates (components). Point can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NPointDimension defines the number of components in the point array.

**See** Image

**See** Mesh

**See** Vector

**See** CovariantVector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Distance between two points
- Distance between two indices

See itk::Point for additional documentation.

template<unsigned int **VDimension** = 2>
**struct Index**

Represent a n-dimensional index in a n-dimensional image.

Index is a templated class to represent a multi-dimensional index, i.e. (i,j,k,...). Index is templated over the dimension of the index. ITK assumes the first element of an index is the fastest moving index.

For efficiency sake, Index does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Index is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Index<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Index<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Distance between two indices
- Create A Index

See itk::Index for additional documentation.

### Distance Between Points

### Synopsis

Compute the distance between two 3D points. This can easily be extended to ND by changing the constant Dimension.

### Results

Output:

```
Dist: 1.73205
Dist2: 3
```

### Code

#### C++

```
#include "itkPoint.h"
#include "itkMath.h"

#include <iostream>

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using PointType = itk::Point<CoordType, Dimension>;

  PointType p0;
  p0[0] = 0.0;
  p0[1] = 0.0;
  p0[2] = 0.0;

  PointType p1;
  p1[0] = 1.0;
  p1[1] = 1.0;
  p1[2] = 1.0;

  PointType::RealType dist = p0.EuclideanDistanceTo(p1);
  std::cout << "Dist: " << dist << std::endl;

  if (dist != p1.EuclideanDistanceTo(p0))
  {
    std::cerr << "p0.EuclideanDistanceTo(p1) != p1.EuclideanDistanceTo(p0)" << std::
↪endl;
    return EXIT_FAILURE;
  }

  if (p1.EuclideanDistanceTo(p1) != 0.)
  {
    std::cerr << "p1.EuclideanDistanceTo(p1) != 0." << std::endl;
    return EXIT_FAILURE;
  }

  PointType::RealType dist2 = p0.SquaredEuclideanDistanceTo(p1);
  std::cout << "Dist2: " << dist2 << std::endl;

  if (std::abs(dist2 - dist * dist) < itk::Math::eps)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "dist2 != dist * dist" << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TCoordRep**, unsigned int **NPointDimension** = 3>
**class Point** : **public** itk::*FixedArray*<*TCoordRep*, *NPointDimension*>

A templated class holding a geometric point in n-Dimensional space.

Point is a templated class that holds a set of coordinates (components). Point can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NPointDimension defines the number of components in the point array.

**See** Image

**See** Mesh

**See** Vector

**See** CovariantVector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Distance between two points
- Distance between two indices

See itk::Point for additional documentation.

### Do Data Parallel Threading

### Synopsis

In this example, we show how to do an operation on data across multiple threads to take advantage of multi-core processors.

In this example we have an *enum* of central nervous system cell types, a *std::vector< CELL_TYPE >* that contains the cells that we have encountered, and we want to count how many we have of each *CELL_TYPE*.

### Results

Output:

```
Result of the multi-threaded cell count:
      NEURON:          3
      ASTROCYTE:       5
      OLIGODENDROCYTE: 2
Result of the single-threaded cell count:
      NEURON:          3
      ASTROCYTE:       5
      OLIGODENDROCYTE: 2
```

### Code

### C++

```
#include "itkDomainThreader.h"
#include "itkThreadedIndexedContainerPartitioner.h"

// We have central nervous system cells of different types.
enum CELL_TYPE
{
  NEURON,
  ASTROCYTE,
  OLIGODENDROCYTE
};

// Type to hold our list of cells to count.
using CellContainerType = std::vector<CELL_TYPE>;
// Type to hold the count for each CELL_TYPE.
using CellCountType = std::map<CELL_TYPE, unsigned int>;


// This class performs the multi-threaded cell type counting for the
// CellCounter class, show below.  The CellCounter class is the TAssociate, and
// since this class is declared as a friend, it can access the CellCounter's
// private members to compute the cell type count for the CellCounter.
//
// While the threading class can access its associate's private members, it
// generally should only do so in a read-only manner.  Otherwise, attempting to
// write to the same member from multiple threads will cause race conditions
// and result in erroreous output or crash the program.  For this reason, the
// threading class contains its own data structures that can be written to in
// individual threads.  These data structures are set up in the
// BeforeThreadedExecution method, and the results contained in each data
// structure are collected in AfterThreadedExecution.  In this case, we have
// m_CellCountPerThread whose counts are initialized to zero in
// BeforeThreadedExecution and collected together in AfterThreadedExecution.
//
// All members and methods related to the multi-threaded computation are
// encapsulated in this class.
//
// The class inherits from itk::DomainThreader, which provides common
// functionality and defines the stages of the multi-threaded operation.
```

(continues on next page)

(continued from previous page)

```
//
// The itk::DomainThreader is templated over the type of DomainPartitioner used
// to split up the domain, and type of the associated class.  The domain in
// this case is a range of indices of a std::vector< CELL_TYPE > to process, so
// we use a ThreadedIndexedContainerPartitioner.  Other options for a domains
// defined as an iterator range or an image region are the
// ThreadedIteratorRangePartitioner and the ThreadedImageRegionPartitioner,
// respectively.

template <class TAssociate>
class ComputeCellCountThreader : public itk::DomainThreader<itk::
↪ThreadedIndexedContainerPartitioner, TAssociate>
{
public:
  // Standard ITK type alias.
  using Self = ComputeCellCountThreader;
  using Superclass = itk::DomainThreader<itk::ThreadedIndexedContainerPartitioner,␣
↪TAssociate>;
  using Pointer = itk::SmartPointer<Self>;
  using ConstPointer = itk::SmartPointer<const Self>;

  // The domain is an index range.
  using DomainType = typename Superclass::DomainType;

  // This creates the ::New() method for instantiating the class.
  itkNewMacro(Self);

protected:
  // We need a constructor for the itkNewMacro.
  ComputeCellCountThreader() = default;

private:
  void
  BeforeThreadedExecution() override
  {
    // Reset the counts for all cell types to zero.
    this->m_Associate->m_CellCount[NEURON] = 0;
    this->m_Associate->m_CellCount[ASTROCYTE] = 0;
    this->m_Associate->m_CellCount[OLIGODENDROCYTE] = 0;

    // Resize our per-thread data structures to the number of threads that we
    // are actually going to use.  At this point the number of threads that
    // will be used have already been calculated and are available.  The number
    // of threads used depends on the number of cores or processors available
    // on the current system.  It will also be truncated if, for example, the
    // number of cells in the CellContainer is smaller than the number of cores
    // available.
    const itk::ThreadIdType numberOfThreads = this->GetNumberOfWorkUnitsUsed();
    this->m_CellCountPerThread.resize(numberOfThreads);
    for (itk::ThreadIdType ii = 0; ii < numberOfThreads; ++ii)
    {
      this->m_CellCountPerThread[ii][NEURON] = 0;
      this->m_CellCountPerThread[ii][ASTROCYTE] = 0;
      this->m_CellCountPerThread[ii][OLIGODENDROCYTE] = 0;
    }
  }
```

(continues on next page)

(continued from previous page)

```
  void
  ThreadedExecution(const DomainType & subDomain, const itk::ThreadIdType threadId)␣
↪override
  {
    // Look only at the range of cells by the set of indices in the subDomain.
    for (itk::IndexValueType ii = subDomain[0]; ii <= subDomain[1]; ++ii)
    {
      switch (this->m_Associate->m_Cells[ii])
      {
        case NEURON:
          // Accumulate in the per thread cell count.
          ++(this->m_CellCountPerThread[threadId][NEURON]);
          break;
        case ASTROCYTE:
          ++(this->m_CellCountPerThread[threadId][ASTROCYTE]);
          break;
        case OLIGODENDROCYTE:
          ++(this->m_CellCountPerThread[threadId][OLIGODENDROCYTE]);
          break;
      }
    }
  }

  void
  AfterThreadedExecution() override
  {
    // Accumulate the cell counts per thread in the associate's total cell
    // count.
    const itk::ThreadIdType numberOfThreads = this->GetNumberOfWorkUnitsUsed();
    for (itk::ThreadIdType ii = 0; ii < numberOfThreads; ++ii)
    {
      this->m_Associate->m_CellCount[NEURON] += this->m_
↪CellCountPerThread[ii][NEURON];

      this->m_Associate->m_CellCount[ASTROCYTE] += this->m_
↪CellCountPerThread[ii][ASTROCYTE];

      this->m_Associate->m_CellCount[OLIGODENDROCYTE] += this->m_
↪CellCountPerThread[ii][OLIGODENDROCYTE];
    }
  }

  std::vector<CellCountType> m_CellCountPerThread;
};


// A class to count the cells.
class CellCounter
{
public:
  using Self = CellCounter;

  using ComputeCellCountThreaderType = ComputeCellCountThreader<Self>;

  // Constructor.  Create our Threader class instance.
  CellCounter() { this->m_ComputeCellCountThreader = ComputeCellCountThreaderType::
↪New(); }
```

(continues on next page)

(continued from previous page)

```
  // Set the cells we want to count.
  void
  SetCells(const CellContainerType & cells)
  {
    this->m_Cells.resize(cells.size());
    for (size_t ii = 0; ii < cells.size(); ++ii)
    {
      this->m_Cells[ii] = cells[ii];
    }
  }

  // Count the cells and return the count of each type.
  const CellCountType &
  ComputeCellCount()
  {
    ComputeCellCountThreaderType::DomainType completeDomain;
    completeDomain[0] = 0;
    completeDomain[1] = this->m_Cells.size() - 1;
    this->m_ComputeCellCountThreader->Execute(this, completeDomain);
    return this->m_CellCount;
  }

private:
  // Stores the count of each type of cell.
  CellCountType m_CellCount;
  // Stores the cells to count.
  CellContainerType m_Cells;

  // The ComputeCellCountThreader gets to access m_CellCount and m_Cells as
  // needed.
  friend class ComputeCellCountThreader<Self>;
  ComputeCellCountThreaderType::Pointer m_ComputeCellCountThreader;
};


int
main(int, char *[])
{
  // Our cells.
  static const CELL_TYPE cellsArr[] = { NEURON, ASTROCYTE, ASTROCYTE, OLIGODENDROCYTE,
↪ ASTROCYTE,
                                        NEURON, NEURON,    ASTROCYTE, ASTROCYTE,
↪ OLIGODENDROCYTE };

  CellContainerType cells(cellsArr, cellsArr + sizeof(cellsArr) /
↪sizeof(cellsArr[0]));

  // Count them in a multi-threader way.
  CellCounter cellCounter;
  cellCounter.SetCells(cells);
  const CellCountType multiThreadedCellCount = cellCounter.ComputeCellCount();
  std::cout << "Result of the multi-threaded cell count:\n";
  std::cout << "\tNEURON:          " << (*multiThreadedCellCount.find(NEURON)).second
↪<< "\n";
  std::cout << "\tASTROCYTE:       " << (*multiThreadedCellCount.find(ASTROCYTE)).
↪second << "\n";
```

(continues on next page)

(continued from previous page)

```
  std::cout << "\tOLIGODENDROCYTE: " << (*multiThreadedCellCount.
↪find(OLIGODENDROCYTE)).second << "\n";


  // Count them in a single-threaded way.
  CellCountType singleThreadedCellCount;
  singleThreadedCellCount[NEURON] = 0;
  singleThreadedCellCount[ASTROCYTE] = 0;
  singleThreadedCellCount[OLIGODENDROCYTE] = 0;
  for (auto & cell : cells)
  {
    switch (cell)
    {
      case NEURON:
        // Accumulate in the per thread cell count.
        ++(singleThreadedCellCount[NEURON]);
        break;
      case ASTROCYTE:
        ++(singleThreadedCellCount[ASTROCYTE]);
        break;
      case OLIGODENDROCYTE:
        ++(singleThreadedCellCount[OLIGODENDROCYTE]);
        break;
    }
  }
  std::cout << "Result of the single-threaded cell count:\n";
  std::cout << "\tNEURON:          " << (*singleThreadedCellCount.find(NEURON)).
↪second << "\n";
  std::cout << "\tASTROCYTE:       " << (*singleThreadedCellCount.find(ASTROCYTE)).
↪second << "\n";
  std::cout << "\tOLIGODENDROCYTE: " << (*singleThreadedCellCount.
↪find(OLIGODENDROCYTE)).second << "\n";


  // Did we get what was expected?  It is always good to check a multi-threaded
  // implementation against a single-threaded implementation to ensure that it
  // gets the same results.
  if ((*multiThreadedCellCount.find(NEURON)).second !=␣
↪singleThreadedCellCount[NEURON] ||
      (*multiThreadedCellCount.find(ASTROCYTE)).second !=␣
↪singleThreadedCellCount[ASTROCYTE] ||
      (*multiThreadedCellCount.find(OLIGODENDROCYTE)).second !=␣
↪singleThreadedCellCount[OLIGODENDROCYTE])
  {
    std::cerr << "Error: did not get the same results"
              << "for a single-threaded and multi-threaded calculation." << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TDomainPartitioner**, typename **TAssociate**>
**class DomainThreader** : **public** itk::*Object*

Multi-threaded processing on a domain by processing sub-domains per thread.

This class uses a ThreadedDomainPartitioner as a helper to split the domain into subdomains. Each subdomain is then processed in the `ThreadedExecution` method.

The data on which to perform the processing is assumed to be members of an associating class. Therefore, to perform a threaded operation in a class, the associating class usually will declare derived versions of this class as a friend class.

To use this class, at a minimum,

- Inherit from it.
- Implement ThreadedExecution.
- Create a member instance.
- Run with m_DomainThreader->Execute( this, domain );

If a 'threaded method' is desired to perform some data processing in a class, a derived version of this class can be defined to perform the threaded operation. Since a threaded operation is relatively complex compared to a simple serial operation, a class instead of a simple method is required. Inside this class, the method to partition the data is handled, the logic for deciding the number of work units is determined, and operations surrounding the threading are encapsulated into the class with the `DetermineNumberOfWorkUnitsToUse`, `BeforeThreadedExecution`, `ThreadedExecution`, and `AfterThreadedExecution` virtual methods.

**Template Parameters**

- `TDomainPartitioner`: A class that inherits from ThreadedDomainPartitioner.
- `TAssociate`: The associated class that uses a derived version of this class as a "threaded method". The associated class usually declares derived version of this class as nested classes so there is easy access to its protected and private members in ThreadedExecution.

See itk::DomainThreader for additional documentation.

### Duplicate an Image

### Synopsis

This example demonstrates how to copy/clone/duplicate an image so it can continue down two separate paths of the pipeline.

### Results

### Code

### Python

```python
#!/usr/bin/env python

import itk

Dimension = 2
PixelType = itk.UC

ImageType = itk.Image[PixelType, Dimension]

randomImageSource = itk.RandomImageSource[ImageType].New()
randomImageSource.SetNumberOfWorkUnits(1)  # to produce non-random results

image = randomImageSource.GetOutput()

clonedImage = itk.image_duplicator(image)
```

### C++

```cpp
#include "itkImage.h"
#include "itkImageDuplicator.h"
#include "itkRandomImageSource.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  using RandomSourceType = itk::RandomImageSource<ImageType>;

  RandomSourceType::Pointer randomImageSource = RandomSourceType::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce non-random results

  ImageType::Pointer image = randomImageSource->GetOutput();

  using DuplicatorType = itk::ImageDuplicator<ImageType>;
  DuplicatorType::Pointer duplicator = DuplicatorType::New();
  duplicator->SetInputImage(image);
  duplicator->Update();

  ImageType::Pointer clonedImage = duplicator->GetOutput();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ImageDuplicator** : **public** itk::*Object*

A helper class which creates an image which is perfect copy of the input image.

This class is NOT a filter. Although it has an API similar to a filter, this class is not intended to be used in a pipeline. Instead, the typical use will be like it is illustrated in the following code:

```
medianFilter->Update();
ImageType::Pointer image = medianFilter->GetOutput();
using DuplicatorType = itk::ImageDuplicator< ImageType >;
DuplicatorType::Pointer duplicator = DuplicatorType::New();
duplicator->SetInputImage(image);
duplicator->Update();
ImageType::Pointer clonedImage = duplicator->GetOutput();
```

Note that the Update() method must be called explicitly in the filter that provides the input to the ImageDuplicator object. This is needed because the ImageDuplicator is not a pipeline filter.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Duplicate An Image

See itk::ImageDuplicator for additional documentation.

### Filter and ParallelizeImageRegion Comparison

### Synopsis

This example demonstrates how to take advantage of MultiThreaderBase::ParallelizeImageRegion. A comparison is made with LogImageFilter for purpose of computing log(1+x), where x is pixel value.

With ParallelizeImageRegion, we can process an image efficiently and in parallel with a function that is defined in a flexible way. The entire operation, defined in 10 lines of code, previously required the definition of an entire class.

### Results

Output:

```
LogImageFilter and ParallelizeImageRegion generate the same result.
```

### Code

### C++

```
#include "itkLogImageFilter.h"
#include "itkRandomImageSource.h"
#include "itkImageDuplicator.h"
#include "itkMultiThreaderBase.h"
#include "itkImageRegionIterator.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned int;
using ImageType = itk::Image<PixelType, Dimension>;

// calculate log(1+x), where x is pixel value, using LogImageFilter
void
log1xViaLogImageFilter(ImageType::Pointer & image)
{
  // LogImageFilter calculates log(x), so we have to modify the image first
  // by increase its every pixel value by 1, and then apply log filter to it
  itk::ImageRegionIterator<ImageType> it(image, image->GetBufferedRegion());
  for (; !it.IsAtEnd(); ++it)
  {
    it.Set(1 + it.Get());
  }

  // classic filter declaration and invocation
  using LogType = itk::LogImageFilter<ImageType, ImageType>;
  LogType::Pointer logF = LogType::New();
  logF->SetInput(image);
  logF->SetInPlace(true);
  logF->Update();
  image = logF->GetOutput();
  image->DisconnectPipeline();
}

// calculate log(1+x), where x is pixel value, using ParallelizeImageRegion
void
log1xViaParallelizeImageRegion(ImageType::Pointer & image)
{
  itk::MultiThreaderBase::Pointer mt = itk::MultiThreaderBase::New();
  // ParallelizeImageRegion invokes the provided lambda function in parallel
  // each invocation will contain a piece of the overall region
  mt->ParallelizeImageRegion<Dimension>(
    image->GetBufferedRegion(),
    // here we creat an ad-hoc lambda function to process the region pieces
    // the lambda will have access to variable 'image' from the outer function
    // it will have parameter 'region', which needs to be processed
    [image](const ImageType::RegionType & region) {
      itk::ImageRegionIterator<ImageType> it(image, region);
      for (; !it.IsAtEnd(); ++it)
      {
        it.Set(std::log(1 + it.Get()));
      }
    },
    nullptr); // we don't have a filter whose progress needs to be updated
}
```

(continues on next page)

(continued from previous page)

```
int
main(int, char *[])
{
  int result = EXIT_SUCCESS;

  // create an image
  ImageType::RegionType region = { { 0, 0 }, { 50, 20 } }; // indices zero, size 50x20
  using RandomSourceType = itk::RandomImageSource<ImageType>;
  RandomSourceType::Pointer randomImageSource = RandomSourceType::New();
  randomImageSource->SetSize(region.GetSize());
  // we don't want overflow on 1+x operation, so set max pixel value
  randomImageSource->SetMax(itk::NumericTraits<PixelType>::max() - 1);
  randomImageSource->SetNumberOfWorkUnits(1); // to produce deterministic results
  randomImageSource->Update();

  ImageType::Pointer image = randomImageSource->GetOutput();
  image->DisconnectPipeline();

  // create another image, to be passed to the alternative method
  using DuplicatorType = itk::ImageDuplicator<ImageType>;
  DuplicatorType::Pointer duplicator = DuplicatorType::New();
  duplicator->SetInputImage(image);
  duplicator->Update();

  ImageType::Pointer clonedImage = duplicator->GetOutput();
  clonedImage->DisconnectPipeline();

  // invoke the two functions
  log1xViaLogImageFilter(image);
  log1xViaParallelizeImageRegion(clonedImage);

  // compare to make sure the results are the same
  unsigned                                  diffCount = 0;
  itk::ImageRegionConstIterator<ImageType> it1(image, region);
  itk::ImageRegionConstIterator<ImageType> it2(clonedImage, region);
  for (; !it1.IsAtEnd(); ++it1, ++it2)
  {
    if (it1.Get() != it2.Get())
    {
      std::cerr << "Pixel values are different at index " << it1.GetIndex() << it1.
↪Get() << " vs. " << it2.Get()
                << std::endl;
      //<< "\n\tlog1xViaLogImageFilter's value: " << it1.Get()
      //<< "\n\tlog1xViaParallelizeImageRegion: " << it2.Get() << std::endl;
      diffCount++;
      result = EXIT_FAILURE;
    }
  }

  if (diffCount == 0)
  {
    std::cout << "LogImageFilter and ParallelizeImageRegion generate the same result.
↪" << std::endl;
  }
  else
  {
```

(continues on next page)

(continued from previous page)

```
    std::cout << "Discrepancy! " << diffCount << " pixels out of " << region.
↪GetNumberOfPixels() << " are different."
              << std::endl;
  }
  return result;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LogImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the log() of each pixel.

See itk::LogImageFilter for additional documentation.

**class MultiThreaderBase** : **public** itk::*Object*
A class for performing multithreaded execution.

Multithreaders are a class hierarchy that provides support for multithreaded execution by abstracting away platform-specific details. This class can be used to execute a single method on multiple threads or to parallelize an operation over a given image region or array.

Subclassed by itk::PoolMultiThreader, itk::TBBMultiThreader

See itk::MultiThreaderBase for additional documentation.

### Filter Image

### Synopsis

Filter an image.

### Results

Output:

```
0
3
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "ImageFilter.h"

template <typename TImage>
```

(continues on next page)

(continued from previous page)

```
static void
CreateImage(TImage * const image);

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<int, 2>;
  using FilterType = itk::ImageFilter<ImageType>;

  ImageType::Pointer image = ImageType::New();
  CreateImage(image.GetPointer());

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->Update();

  itk::Index<2> cornerPixel = image->GetLargestPossibleRegion().GetIndex();

  // The output here is:
  // 0
  // 3
  // That is, the filter changed the pixel, but the input remained unchagned.
  std::cout << image->GetPixel(cornerPixel) << std::endl;
  std::cout << filter->GetOutput()->GetPixel(cornerPixel) << std::endl;

  return EXIT_SUCCESS;
}


template <typename TImage>
void
CreateImage(TImage * const image)
{
  // Create an image with 2 connected components
  typename TImage::IndexType corner = { { 0, 0 } };

  unsigned int              NumRows = 200;
  unsigned int              NumCols = 300;
  typename TImage::SizeType size = { { NumRows, NumCols } };

  typename TImage::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  image->FillBuffer(0);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >,

itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage

>, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Filter Image Using Multiple Threads

### Synopsis

Filter an image using multiple threads.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "MultiThreadedImageFilter.h"

template <typename TImage>
static void
CreateImage(TImage * const image);

template <typename TImage>
```

(continues on next page)

(continued from previous page)

```
static void
OutputImage(TImage * const image);

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<int, 2>;
  using FilterType = itk::MultiThreadedImageFilter<ImageType>;

  ImageType::Pointer image = ImageType::New();
  CreateImage(image.GetPointer());

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  // filter->SetNumberOfThreads(3); // There is no need to specify this, it is␣
↪automatically determined
  filter->Update();

  std::cout << "Image after filter: " << std::endl;
  OutputImage(image.GetPointer());

  std::cout << "Output: " << std::endl;
  OutputImage(filter->GetOutput());

  return EXIT_SUCCESS;
}


template <typename TImage>
void
CreateImage(TImage * const image)
{
  // Create an image with 2 connected components
  typename TImage::IndexType corner = { { 0, 0 } };

  //   unsigned int NumRows = 200;
  //   unsigned int NumCols = 300;

  unsigned int              NumRows = 3;
  unsigned int              NumCols = 2;
  typename TImage::SizeType size = { { NumRows, NumCols } };

  typename TImage::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  image->FillBuffer(0);
}

template <typename TImage>
void
OutputImage(TImage * const image)
{
  itk::ImageRegionConstIterator<TImage> imageIterator(image, image->
↪GetLargestPossibleRegion());
```

(continues on next page)

(continued from previous page)

```
  while (!imageIterator.IsAtEnd())
  {
    std::cout << imageIterator.Get() << std::endl;

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image

- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >,

itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputIm-

age >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage >, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Filter Image Without Copying Its Data

### Synopsis

Filter an image without copying its data.

### Results

Output:

```
3
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "MyInPlaceImageFilter.h"

template <typename TImage>
static void
CreateImage(TImage * const image);

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<int, 2>;
  using FilterType = itk::MyInPlaceImageFilter<ImageType>;

  ImageType::Pointer image = ImageType::New();
  CreateImage(image.GetPointer());

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->SetInPlace(true);
  filter->Update();

  itk::Index<2> cornerPixel = image->GetLargestPossibleRegion().GetIndex();

  // The output here should be "3"
  std::cout << "Filter output:" << std::endl;
  std::cout << filter->GetOutput()->GetPixel(cornerPixel) << std::endl;

  // The follow calls fail. This is because the output has stolen the input's
  // buffer and the input has no image buffer.
  //   std::cout << "Image output:" << std::endl;
  //   std::cout << image->GetPixel(cornerPixel) << std::endl;

  return EXIT_SUCCESS;
}


template <typename TImage>
void
CreateImage(TImage * const image)
{
```

(continues on next page)

(continued from previous page)

```
  // Create an image with 2 connected components
  typename TImage::IndexType corner = { { 0, 0 } };

  unsigned int              NumRows = 200;
  unsigned int              NumCols = 300;
  typename TImage::SizeType size = { { NumRows, NumCols } };

  typename TImage::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  image->FillBuffer(0);
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class InPlaceImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for filters that take an image as input and overwrite that image as the output.

InPlaceImageFilter is the base class for all process objects whose output image data is constructed by overwriting the input image data. In other words, the output bulk data is the same block of memory as the input bulk data. This filter provides the mechanisms for in place image processing while maintaining general pipeline mechanics. InPlaceImageFilters use less memory than standard ImageToImageFilters because the input buffer is reused as the output buffer. However, this benefit does not come without a cost. Since the filter overwrites its input, the ownership of the bulk data is transitioned from the input data object to the output data object. When a data object has multiple consumers with one of the consumers being an in place filter, the in place filter effectively destroys the bulk data for the data object. Upstream filters will then have to re-execute to regenerate the data object's bulk data for the remaining consumers.

Since an InPlaceImageFilter reuses the input bulk data memory for the output bulk data memory, the input image type must match the output image type. If the input and output image types are not identical, the filter reverts to a traditional ImageToImageFilter behaviour where an output image is allocated. Additionally, the requested region of the output must match that of the input. In place operation can also be controlled (when the input and output image type match) via the methods InPlaceOn() and InPlaceOff().

Subclasses of InPlaceImageFilter must take extra care in how they manage memory using (and perhaps overriding) the implementations of ReleaseInputs() and AllocateOutputs() provided here.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image Without Copying Its Data
- In Place Filter Of Image

Subclassed by itk::BinaryGeneratorImageFilter< TInputImage, TLabelImage, TOutputImage >, itk::BinaryGeneratorImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::BinaryContourImageFilter< TInputImage, TOutputImage >, itk::CastImageFilter< TInputImage, TOutputImage >, itk::ExtractImageFilter< TInputImage, TOutputImage >, itk::FiniteDifferenceImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LabelContourImageFilter< TInputImage, TOutputImage >, itk::NaryFunctorImageFilter<

TInputImage, TOutputImage, TFunction >, itk::NoiseBaseImageFilter< TInputImage, TOutputImage >, itk::PasteImageFilter< TInputImage, TSourceImage, TOutputImage >, itk::RecursiveSeparableImageFilter< TInputImage, TOutputImage >, itk::RelabelComponentImageFilter< TInputImage, TOutputImage >, itk::SmoothingRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, TFunction >, itk::UnaryGeneratorImageFilter< TInputImage, TOutputImage >, itk::NaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Add1< TInputImage::PixelType, TInputImage::PixelType > >, itk::NaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Maximum1< TInputImage::PixelType, TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::AccessorFunctor< TInputImage::PixelType, TAccessor > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::BinaryThreshold< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ChangeLabel< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Clamp< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ExpNegative< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::IntensityLinearTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::IntensityWindowingTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::InvertIntensityTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::MatrixIndexSelection< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::NOT< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Sigmoid< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::SymmetricEigenAnalysisFixedDimensionFunction< TMatrixDimension, TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::SymmetricEigenAnalysisFunction< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::TensorFractionalAnisotropyFunction< TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::TensorRelativeAnisotropyFunction< TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ThresholdLabeler< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::VectorIndexSelectionCast< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::VectorMagnitudeLinearTransform< TInputImage::PixelType, TOutputImage::PixelType > >

See itk::InPlaceImageFilter for additional documentation.

### Find Max and Min in Image

### Synopsis

Find the minimum and maximum value (and the position of the value) in an image.

### Results

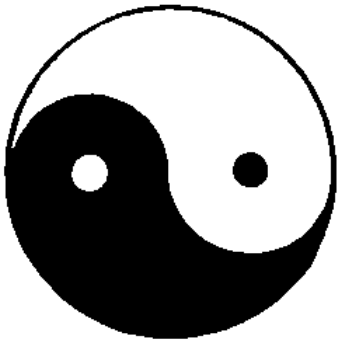


Fig. 21: Yingyang.png

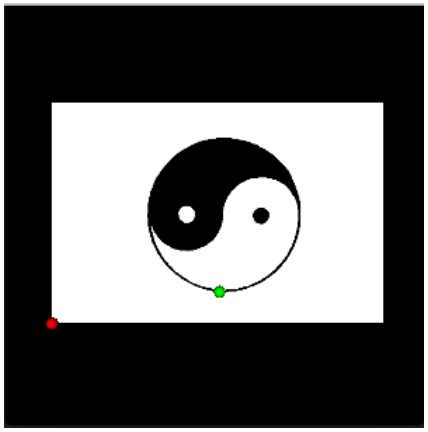


Fig. 22: Output Image

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"
#include "itkImageFileReader.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
```

(continues on next page)

(continued from previous page)

```
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"
#include "vtkSphereSource.h"
#include "vtkPolyDataMapper.h"
#include "vtkActor.h"
#include "vtkProperty.h"

using ImageType = itk::Image<unsigned char, 2>;

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;

    return EXIT_FAILURE;
  }
  std::string inputFilename = argv[1];

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();

  reader->SetFileName(inputFilename.c_str());
  reader->Update();

  using ImageCalculatorFilterType = itk::MinimumMaximumImageCalculator<ImageType>;

  ImageCalculatorFilterType::Pointer imageCalculatorFilter =␣
↪ImageCalculatorFilterType::New();
  imageCalculatorFilter->SetImage(reader->GetOutput());
  imageCalculatorFilter->Compute();

  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer originalConnector = ConnectorType::New();

  originalConnector->SetInput(reader->GetOutput());

  vtkSmartPointer<vtkImageActor> originalActor = vtkSmartPointer<vtkImageActor>::
↪New();
#if VTK_MAJOR_VERSION <= 5
  originalActor->SetInput(originalConnector->GetOutput());
#else
  originalConnector->Update();
  originalActor->GetMapper()->SetInputData(originalConnector->GetOutput());
#endif

  vtkSmartPointer<vtkSphereSource> minimumSphereSource = vtkSmartPointer
↪<vtkSphereSource>::New();
  ImageType::IndexType             minimumLocation = imageCalculatorFilter->
↪GetIndexOfMinimum();
  minimumSphereSource->SetCenter(minimumLocation[0], minimumLocation[1], 0);
  minimumSphereSource->SetRadius(10);
```

(continues on next page)

(continued from previous page)

```
  vtkSmartPointer<vtkPolyDataMapper> minimumMapper = vtkSmartPointer
↪<vtkPolyDataMapper>::New();
  minimumMapper->SetInputConnection(minimumSphereSource->GetOutputPort());
  vtkSmartPointer<vtkActor> minimumActor = vtkSmartPointer<vtkActor>::New();
  minimumActor->SetMapper(minimumMapper);
  minimumActor->GetProperty()->SetColor(0, 1, 0);

  vtkSmartPointer<vtkSphereSource> maximumSphereSource = vtkSmartPointer
↪<vtkSphereSource>::New();
  maximumSphereSource->SetRadius(10);
  ImageType::IndexType maximumLocation = imageCalculatorFilter->GetIndexOfMaximum();
  maximumSphereSource->SetCenter(maximumLocation[0], maximumLocation[1], 0);

  vtkSmartPointer<vtkPolyDataMapper> maximumMapper = vtkSmartPointer
↪<vtkPolyDataMapper>::New();
  maximumMapper->SetInputConnection(maximumSphereSource->GetOutputPort());
  vtkSmartPointer<vtkActor> maximumActor = vtkSmartPointer<vtkActor>::New();
  maximumActor->SetMapper(maximumMapper);
  maximumActor->GetProperty()->SetColor(1, 0, 0);


  // Visualize
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(renderer);

  // Add the sphere to the left and the cube to the right
  renderer->AddActor(originalActor);
  renderer->AddActor(minimumActor);
  renderer->AddActor(maximumActor);

  renderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class MinimumMaximumImageCalculator** : **public** itk::*Object*
Computes the minimum and the maximum intensity values of an image.

This calculator computes the minimum and the maximum intensity values of an image. It is templated over input image type. If only Maximum or Minimum value is needed, just call ComputeMaximum() (ComputeMinimum()) otherwise Compute() will compute both.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Find Max And Min In Image
- Multi-thread Oil Painting

See itk::MinimumMaximumImageCalculator for additional documentation.

### Get Image Size

### Synopsis

Get the size of a itk::Image

### Results

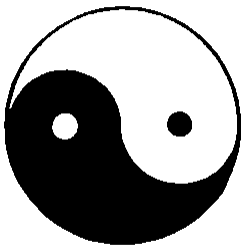


Fig. 23: Input Image

**Output::** [512, 342]

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, 2>;
  using ReaderType = itk::ImageFileReader<ImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::RegionType region = image->GetLargestPossibleRegion();

  ImageType::SizeType size = region.GetSize();

  std::cout << size << std::endl;

  // An example image had w = 200 and h = 100
  // (it is wider than it is tall). The above output
  // 200 x 100
  // so w = GetSize()[0]
  // and h = GetSize()[1]

  // A pixel inside the region
  ImageType::IndexType indexInside;
  indexInside[0] = 150;
  indexInside[1] = 50;
  std::cout << region.IsInside(indexInside) << std::endl;

  // A pixel outside the region
  ImageType::IndexType indexOutside;
  indexOutside[0] = 50;
  indexOutside[1] = 150;
  std::cout << region.IsInside(indexOutside) << std::endl;

  // This means that the [0] component of the index is referencing the
  // left to right (x) value and the [1] component of Index is referencing
  // the top to bottom (y) value
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python
import numpy as np
import itk
import argparse

parser = argparse.ArgumentParser(description="Get Image Size.")
parser.add_argument("input_image")
args = parser.parse_args()

image = itk.imread(args.input_image, itk.UC)

region = image.GetLargestPossibleRegion()
size = region.GetSize()

print(size)

# Equivalently
size = itk.size(image)

print(size)

# Corresponds to the NumPy ndarray shape. Note that the ordering is reversed.
print(np.asarray(image).shape)

# An example image had w = 200 and h = 100
# (it is wider than it is tall). The above output
# 200 x 100
# so w = GetSize()[0]
# and h = GetSize()[1]

# A pixel inside the region
indexInside = itk.Index[2]()
indexInside[0] = 150
indexInside[1] = 50
print(region.IsInside(indexInside))

# A pixel outside the region
indexOutside = itk.Index[2]()
indexOutside[0] = 50
indexOutside[1] = 150
print(region.IsInside(indexOutside))

# This means that the [0] component of the index is referencing the
# left to right (x) value and the [1] component of Index is referencing
# the top to bottom (y) value
```

### Classes demonstrated

template<unsigned int **VDimension** = 2>
**struct Size**

Represent a n-dimensional size (bounds) of a n-dimensional image.

Size is a templated class to represent multi-dimensional array bounds, i.e. (I,J,K,...). Size is templated over teh dimension of the bounds. ITK assumes the first element of a size (bounds) is the fastest moving index.

For efficiency, Size does not define a default constructor, a copy constructor, or an operator=. We rely on the compiler to provide efficient bitwise copies.

Size is an "aggregate" class. Its data is public (m_InternalArray) allowing for fast and convenient instantiations/assignments.

The following syntax for assigning an aggregate type like this is allowed/suggested:

Size<3> var{{ 256, 256, 20 }}; // Also prevent narrowing conversions Size<3> var = {{ 256, 256, 20 }};

The doubled braces {{ and }} are required to prevent gcc -Wall (and perhaps other compilers) from complaining about a partly bracketed initializer.

As an aggregate type that is intended to provide highest performance characteristics, this class is not appropriate to inherit from, so setting this struct as final.

**See** Index

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Size

See itk::Size for additional documentation.

### Get Name of Class

### Synopsis

Get the name of the class of an object.

### Results

**Output::** image is type: Image

### Code

### C++

```
#include "itkImage.h"

int
main(int, char *[])
{
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;
  ImageType::Pointer image = ImageType::New();

  std::cout << "image is type: " << image->GetNameOfClass() << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class LightObject**

Light weight base class for most itk classes.

LightObject is the highest level base class for most itk objects. It implements reference counting and the API for object printing. It can be used as a lightweight base class in preference to Object. (LightObject does not support callbacks or modified time as Object does.) All ITK objects should be a subclass of LightObject or Object with few exceptions (due to performance concerns).

**See** Object

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

\sphinx{Core/Common/GetNameOfClass, Get Name Of Class}

Subclassed by itk::FiniteDifferenceFunction< TDisplacementField >, itk::FiniteDifferenceFunction< TImage >, itk::FiniteDifferenceFunction< TInput >, itk::FiniteDifferenceFunction< TInputImage >, itk::FiniteDifferenceFunction< TSparseImageType >, itk::CellInterface< TPixelType, TCellTraits >::MultiVisitor, itk::CellInterfaceVisitor< TPixelType, TCellTraits >, itk::FiniteDifferenceFunction< TImageType >, itk::GPUContextManager, itk::GPUKernelManager, itk::LabelObject< TLabel, VImageDimension >, itk::MetaDataObjectBase, itk::MRCHeaderObject, itk::NarrowBand< NodeType >, itk::Object, itk::RegionBasedLevelSetFunctionData< TInputImage, TFeatureImage >, itk::RegionBasedLevelSetFunctionSharedData< TInputImage, TFeatureImage, TSingleData >

See itk::LightObject for additional documentation.

### Get or Set Member Variable of ITK Class

### Synopsis

Get or set a member variable of an ITK class.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "ImageFilterX.h"

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::ImageFilter<ImageType>;

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Get Type Basic Information

### Synopsis

Get some basic information about a plain old data (POD) type, in this case the *float* type.

### Results

Output:

```
Min: 1.17549e-38
Max: 3.40282e+38
Zero: 0
ZeroValue: 0
Is -1 negative? 1
Is 1 negative? 0
One: 1
Epsilon: 1.19209e-07
```

(continues on next page)

(continued from previous page)

```
Infinity: inf
Good
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

MyType = itk.F

print("Min: " + str(itk.NumericTraits[MyType].min()))
print("Max: " + str(itk.NumericTraits[MyType].max()))
print("ZeroValue: " + str(itk.NumericTraits[MyType].ZeroValue()))

print("Is -1 negative? " + str(itk.NumericTraits[MyType].IsNegative(-1)))

print("Is 1 negative? " + str(itk.NumericTraits[MyType].IsNegative(1)))

print("OneValue: " + str(itk.NumericTraits[MyType].OneValue()))

print("Epsilon: " + str(itk.NumericTraits[MyType].epsilon()))

print("Infinity: " + str(itk.NumericTraits[MyType].infinity()))

if 0 == itk.NumericTraits[MyType].infinity():
    print(" 0 == inf!")
    exit(1)
else:
    print("Good")
    exit(0)
```

### C++

```cpp
#include <iostream>
#include "itkNumericTraits.h"

int
main(int, char *[])
{
  using MyType = float;

  std::cout << "Min: " << itk::NumericTraits<MyType>::min() << std::endl;
  std::cout << "Max: " << itk::NumericTraits<MyType>::max() << std::endl;
  std::cout << "Zero: " << itk::NumericTraits<MyType>::Zero << std::endl;
  std::cout << "ZeroValue: " << itk::NumericTraits<MyType>::ZeroValue() << std::endl;

  std::cout << "Is -1 negative? " << itk::NumericTraits<MyType>::IsNegative(-1) <<␣
↪std::endl;
```

(continues on next page)

(continued from previous page)

```
  std::cout << "Is 1 negative? " << itk::NumericTraits<MyType>::IsNegative(1) << std::
↪endl;

  std::cout << "One: " << itk::NumericTraits<MyType>::One << std::endl;

  std::cout << "Epsilon: " << itk::NumericTraits<MyType>::epsilon() << std::endl;

  std::cout << "Infinity: " << itk::NumericTraits<MyType>::infinity() << std::endl;

  if (0 == itk::NumericTraits<MyType>::infinity())
  {
    std::cout << " 0 == inf!" << std::endl;
    return EXIT_FAILURE;
  }
  else
  {
    std::cout << "Good" << std::endl;
    return EXIT_SUCCESS;
  }
}
```

## Classes demonstrated

template<typename **T**>
**class NumericTraits** : **public** std::numeric_limits<*T*>

Define additional traits for native types such as int or float.

Define numeric traits for std::vector.

NumericTraits is used to extend the traits associated with native types such as float, char, int, and so on. These traits are extensions of the standard numeric_limits defined by the C++ compilers. Some of the added traits include minimum and maximum value; accumulation type; etc.

We provide here a generic implementation based on creating types of std::vector whose components are the types of the NumericTraits from the original std::vector components. This implementation require support for partial specializations, since it is based on the concept that: NumericTraits<std::vector< T > > is defined piecewise by std::vector< NumericTraits< T > >

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Get Type Basic Information

**Template Parameters**

- `T`: Component type of std::vector

**Note** The Zero(), One(), min() and max() methods here take references to a pixel as input. This is due to the fact that the length of the std::vector is not known until run-time. Since the most common use of Zero and One is for comparison purposes or initialization of sums etc, this might just as easily be re-written with a pixel passed in as a reference and the length is inferred from this pixel.

**See** NumericTraits

See itk::NumericTraits for additional documentation.

### Image Region Intersection

### Synopsis

This example demonstrates how to check if two regions of an image overlap/intersect.

### Results

Output:

```
Small inside region is 1
Small outside region is 0
Small overlap region is 0
```

### Code

### C++

```
#include "itkImage.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  // Big region
  ImageType::RegionType bigRegion;

  ImageType::SizeType bigSize;
  bigSize[0] = 100;
  bigSize[1] = 100;

  ImageType::IndexType bigStart;
  bigStart[0] = 0;
  bigStart[1] = 0;

  bigRegion.SetSize(bigSize);
  bigRegion.SetIndex(bigStart);

  // Small inside region
  ImageType::RegionType smallInsideRegion;

  ImageType::SizeType smallInsideSize;
  smallInsideSize[0] = 10;
  smallInsideSize[1] = 10;

  ImageType::IndexType smallInsideStart;
  smallInsideStart[0] = 50;
  smallInsideStart[1] = 50;
```

(continues on next page)

(continued from previous page)

```
  smallInsideRegion.SetSize(smallInsideSize);
  smallInsideRegion.SetIndex(smallInsideStart);

  std::cout << "Small inside region is " << bigRegion.IsInside(smallInsideRegion) <<␣
↪std::endl;

  // Small outside region
  ImageType::RegionType smallOutsideRegion;

  ImageType::SizeType smallOutsideSize;
  smallOutsideSize[0] = 10;
  smallOutsideSize[1] = 10;

  ImageType::IndexType smallOutsideStart;
  smallOutsideStart[0] = 110;
  smallOutsideStart[1] = 110;

  smallOutsideRegion.SetSize(smallOutsideSize);
  smallOutsideRegion.SetIndex(smallOutsideStart);

  std::cout << "Small outside region is " << bigRegion.IsInside(smallOutsideRegion) <
↪< std::endl;

  // Small overlap region
  ImageType::RegionType smallOverlapRegion;

  ImageType::SizeType smallOverlapSize;
  smallOverlapSize[0] = 10;
  smallOverlapSize[1] = 10;

  ImageType::IndexType smallOverlapStart;
  smallOverlapStart[0] = 97;
  smallOverlapStart[1] = 97;

  smallOverlapRegion.SetSize(smallOverlapSize);
  smallOverlapRegion.SetIndex(smallOverlapStart);

  std::cout << "Small overlap region is " << bigRegion.IsInside(smallOverlapRegion) <
↪< std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VImageDimension**>
**class ImageRegion** : **public** itk::Region

An image region represents a structured region of data.

ImageRegion is an class that represents some structured portion or piece of an Image. The ImageRegion is represented with an index and a size in each of the n-dimensions of the image. (The index is the corner of the image, the size is the lengths of the image in each of the topological directions.)

**See** Region

**See** Index

**See** Size

**See** MeshRegion

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- An object which holds the index (start) and size of a region of an image
- Determine image region intersection
- Determine if a pixel is inside of a region
- Determine the overlap of two regions

See itk::ImageRegion for additional documentation.

### Image Region Overlap

### Synopsis

Determine the overlap of two regions

### Results

Fig. 24: Output image

### Code

### C++

```
#include <cstdlib>
#include <cstdio>

#include "itkRGBPixel.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileWriter.h"


int
main(int argc, char * argv[])
{
  if (argc != 2)
```

(continues on next page)

(continued from previous page)

```
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << " <OutputFileName>" << std::endl;

    return EXIT_FAILURE;
  }
  const char * out_file_name = argv[1];

  constexpr unsigned int Dimension = 2;
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, Dimension>;
  using RegionType = RGBImageType::RegionType;
  using IndexType = RGBImageType::IndexType;
  using SizeType = RGBImageType::SizeType;
  using IteratorType = itk::ImageRegionIterator<RGBImageType>;


  IndexType index;
  index[0] = 0;
  index[1] = 0;

  SizeType size;
  size[0] = 100;
  size[1] = 100;

  RegionType region;
  region.SetIndex(index);
  region.SetSize(size);

  IndexType indexA;
  indexA[0] = 9;
  indexA[1] = 9;

  SizeType sizeA;
  sizeA[0] = 50;
  sizeA[1] = 50;

  RegionType regionA;
  regionA.SetIndex(indexA);
  regionA.SetSize(sizeA);

  IndexType indexB;
  indexB[0] = 39;
  indexB[1] = 39;

  SizeType sizeB;
  sizeB[0] = 50;
  sizeB[1] = 50;

  RegionType regionB;
  regionB.SetIndex(indexB);
  regionB.SetSize(sizeB);

  // Region C is the intersection of A and B
  // padded by 10 pixels.
  RegionType regionC = regionA;
```

(continues on next page)

(continued from previous page)

```
regionC.Crop(regionB);
regionC.PadByRadius(10);

RGBPixelType pix_black;
pix_black.Fill(0);

RGBPixelType pix_red;
pix_red.Fill(0);
pix_red[0] = 255;

RGBPixelType pix_green;
pix_green.Fill(0);
pix_green[1] = 255;

RGBPixelType pix_blue;
pix_blue.Fill(0);
pix_blue[2] = 255;

// A black canvas
RGBImageType::Pointer image = RGBImageType::New();
image->SetRegions(region);
image->Allocate();
image->FillBuffer(pix_black);

// Paint region A red.
IteratorType itA(image, regionA);
itA.GoToBegin();
while (!itA.IsAtEnd())
{
  itA.Set(itA.Get() + pix_red);
  ++itA;
}

// Paint region B green.
IteratorType itB(image, regionB);
itB.GoToBegin();
while (!itB.IsAtEnd())
{
  itB.Set(itB.Get() + pix_green);
  ++itB;
}

// Paint region C blue.
IteratorType itC(image, regionC);
itC.GoToBegin();
while (!itC.IsAtEnd())
{
  itC.Set(itC.Get() + pix_blue);
  ++itC;
}

// Writer
using FileWriterType = itk::ImageFileWriter<RGBImageType>;
FileWriterType::Pointer writer = FileWriterType::New();
writer->SetFileName(out_file_name);
writer->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VImageDimension**>
**class ImageRegion** : **public** itk::Region

An image region represents a structured region of data.

ImageRegion is an class that represents some structured portion or piece of an Image. The ImageRegion is represented with an index and a size in each of the n-dimensions of the image. (The index is the corner of the image, the size is the lengths of the image in each of the topological directions.)

**See** Region

**See** Index

**See** Size

**See** MeshRegion

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- An object which holds the index (start) and size of a region of an image
- Determine image region intersection
- Determine if a pixel is inside of a region
- Determine the overlap of two regions

See itk::ImageRegion for additional documentation.

### Import Pixel Buffer Into an Image

### Synopsis

This example illustrates how to import data into an `Image` class. This is particularly useful for interfacing with other software systems. Many systems use a contiguous block of memory as a buffer for image pixel data. The current example assumes this is the case and feeds the buffer into an `ImportImageFilter`, thereby producing an image as output.

Here we create a synthetic image with a centered sphere in a locally allocated buffer and pass this block of memory to the `ImportImageFilter`. This example is set up so that on execution, the user must provide the name of an output file as a command-line argument.

### Results

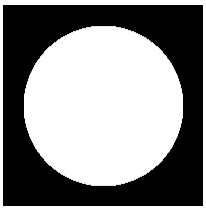


Fig. 25: Output image

### Code

### C++

```
#include "itkImage.h"
#include "itkImportImageFilter.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "  outputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  // We select the data type used to represent the image pixels. We
  // assume that the external block of memory uses the same data type to
  // represent the pixels.
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ImportFilterType = itk::ImportImageFilter<PixelType, Dimension>;
  ImportFilterType::Pointer importFilter = ImportFilterType::New();
  // This filter requires the user to specify the size of the image to be
  // produced as output.  The `SetRegion()` method is used to this end.
  // The image size should exactly match the number of pixels available in the
  // locally allocated buffer.
  ImportFilterType::SizeType size;

  size[0] = 200; // size along X
  size[1] = 200; // size along Y

  ImportFilterType::IndexType start;
```

(continues on next page)

(continued from previous page)

```
  start.Fill(0);

  ImportFilterType::RegionType region;
  region.SetIndex(start);
  region.SetSize(size);

  importFilter->SetRegion(region);

  const itk::SpacePrecisionType origin[Dimension] = { 0.0, 0.0 };
  importFilter->SetOrigin(origin);
  const itk::SpacePrecisionType spacing[Dimension] = { 1.0, 1.0 };
  importFilter->SetSpacing(spacing);
  // Next we allocate the memory block containing the pixel data to be
  // passed to the `ImportImageFilter`. Note that we use exactly the
  // same size that was specified with the `SetRegion()` method. In a
  // practical application, you may get this buffer from some other library
  // using a different data structure to represent the images.
  const unsigned int numberOfPixels = size[0] * size[1];
  auto *             localBuffer = new PixelType[numberOfPixels];
  constexpr double   radius = 80.0;
  // Here we fill up the buffer with a binary sphere.
  const double radius2 = radius * radius;
  PixelType *  it = localBuffer;

  for (unsigned int y = 0; y < size[1]; y++)
  {
    const double dy = static_cast<double>(y) - static_cast<double>(size[1]) / 2.0;
    for (unsigned int x = 0; x < size[0]; x++)
    {
      const double dx = static_cast<double>(x) - static_cast<double>(size[0]) / 2.0;
      const double d2 = dx * dx + dy * dy;
      *it++ = (d2 < radius2) ? 255 : 0;
    }
  }
  // The buffer is passed to the ImportImageFilter with the
  // `SetImportPointer()` method. Note that the last argument of this method
  // specifies who will be responsible for deleting the memory block once it
  // is no longer in use. A `true` value, will allow the
  // filter to delete the memory block upon destruction of the import filter.
  //
  // For the `ImportImageFilter` to appropriately delete the
  // memory block, the memory must be allocated with the C++
  // `new()` operator. Memory allocated with other memory
  // allocation mechanisms, such as C `malloc` or `calloc`, will not
  // be deleted properly by the `ImportImageFilter`. In
  // other words, it is the application programmer's responsibility
  // to ensure that `ImportImageFilter` is only given
  // permission to delete the C++ `new` operator-allocated memory.
  const bool importImageFilterWillOwnTheBuffer = true;
  importFilter->SetImportPointer(localBuffer, numberOfPixels,␣
↪importImageFilterWillOwnTheBuffer);
  // Finally, we can connect the output of this filter to a pipeline.
  // For simplicity we just use a writer here, but it could be any other filter.
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName(argv[1]);
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(importFilter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & exp)
  {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << exp << std::endl;
    return EXIT_FAILURE;
  }

  // Note that we do not call `delete` on the buffer since we pass
  // `true` as the last argument of `SetImportPointer()`. Now the
  // buffer is owned by the `ImportImageFilter`.

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class ImportImageFilter** : **public** itk::*ImageSource*<*Image*<*TPixel*, *VImageDimension*>>
Import data from a standard C array into an itk::Image.

ImportImageFilter provides a mechanism for importing data into an itk::Image. ImportImageFilter is an image source, so it behaves like any other pipeline object.

This class is templated over the pixel type and the image dimension of the output image.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert Array To Image

See itk::ImportImageFilter for additional documentation.

### In Place Filter of Image

### Synopsis

In-place filtering of an image.

### Results

Fig. 26: Input Image

Fig. 27: Output In QuickView

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkBinaryThresholdImageFilter.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"

#include "vtkSmartPointer.h"
#include "vtkRenderWindow.h"
#include "vtkRenderer.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkImageActor.h"
#include "vtkImageMapper3D.h"
```

(continues on next page)

(continued from previous page)

```
#include "vtkImageViewer.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
ApplyThresholding(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  std::string inputFilename = argv[1];

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();

  reader->SetFileName(inputFilename.c_str());
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ApplyThresholding(image);

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector = ConnectorType::New();
  connector->SetInput(image);

  vtkSmartPointer<vtkImageActor> actor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor->SetInput(connector->GetOutput());
#else
  connector->Update();
  actor->GetMapper()->SetInputData(connector->GetOutput());
#endif

  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(renderer);

  renderer->AddActor(actor);
  renderer->ResetCamera();

  renderWindow->Render();
```

(continues on next page)

(continued from previous page)

```
  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();

  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}

void
ApplyThresholding(ImageType::Pointer image)
{

  using BinaryThresholdImageFilterType = itk::BinaryThresholdImageFilter<ImageType,␣
↪ImageType>;

  BinaryThresholdImageFilterType::Pointer thresholdFilter =␣
↪BinaryThresholdImageFilterType::New();
  thresholdFilter->SetInput(image);
  thresholdFilter->SetLowerThreshold(10);
  thresholdFilter->SetUpperThreshold(50);
  thresholdFilter->SetInsideValue(255);
  thresholdFilter->SetOutsideValue(0);
  thresholdFilter->InPlaceOn();
  thresholdFilter->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class InPlaceImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for filters that take an image as input and overwrite that image as the output.

InPlaceImageFilter is the base class for all process objects whose output image data is constructed by overwriting the input image data. In other words, the output bulk data is the same block of memory as the input bulk data. This filter provides the mechanisms for in place image processing while maintaining general pipeline mechanics. InPlaceImageFilters use less memory than standard ImageToImageFilters because the input buffer is reused as the output buffer. However, this benefit does not come without a cost. Since the filter overwrites its input, the ownership of the bulk data is transitioned from the input data object to the output data object. When a data object has multiple consumers with one of the consumers being an in place filter, the in place filter effectively destroys the bulk data for the data object. Upstream filters will then have to re-execute to regenerate the data object's bulk data for the remaining consumers.

Since an InPlaceImageFilter reuses the input bulk data memory for the output bulk data memory, the input image type must match the output image type. If the input and output image types are not identical, the filter reverts to a traditional ImageToImageFilter behaviour where an output image is allocated. Additionally, the requested region of the output must match that of the input. In place operation can also be controlled (when the input and output image type match) via the methods InPlaceOn() and InPlaceOff().

Subclasses of InPlaceImageFilter must take extra care in how they manage memory using (and perhaps overriding) the implementations of ReleaseInputs() and AllocateOutputs() provided here.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image Without Copying Its Data
- In Place Filter Of Image

Subclassed by itk::BinaryGeneratorImageFilter< TInputImage, TLabelImage, TOutputImage >, itk::BinaryGeneratorImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::BinaryContourImageFilter< TInputImage, TOutputImage >, itk::CastImageFilter< TInputImage, TOutputImage >, itk::ExtractImageFilter< TInputImage, TOutputImage >, itk::FiniteDifferenceImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LabelContourImageFilter< TInputImage, TOutputImage >, itk::NaryFunctorImageFilter< TInputImage, TOutputImage, TFunction >, itk::NoiseBaseImageFilter< TInputImage, TOutputImage >, itk::PasteImageFilter< TInputImage, TSourceImage, TOutputImage >, itk::RecursiveSeparableImageFilter< TInputImage, TOutputImage >, itk::RelabelComponentImageFilter< TInputImage, TOutputImage >, itk::SmoothingRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, TFunction >, itk::UnaryGeneratorImageFilter< TInputImage, TOutputImage >, itk::NaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Add1< TInputImage::PixelType, TInputImage::PixelType > >, itk::NaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Maximum1< TInputImage::PixelType, TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::AccessorFunctor< TInputImage::PixelType, TAccessor > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::BinaryThreshold< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ChangeLabel< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Clamp< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ExpNegative< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::IntensityLinearTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::IntensityWindowingTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::InvertIntensityTransform< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::MatrixIndexSelection< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::NOT< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::Sigmoid< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::SymmetricEigenAnalysisFixedDimensionFunction< TMatrixDimension, TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::SymmetricEigenAnalysisFunction< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::TensorFractionalAnisotropyFunction< TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::TensorRelativeAnisotropyFunction< TInputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::ThresholdLabeler< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::VectorIndexSelectionCast< TInputImage::PixelType, TOutputImage::PixelType > >, itk::UnaryFunctorImageFilter< TInputImage, TOutputImage, Functor::VectorMagnitudeLinearTransform< TInputImage::PixelType, TOutputImage::PixelType > >

See itk::InPlaceImageFilter for additional documentation.

### Is Pixel Inside Region

### Synopsis

Determine if a pixel is inside of a region

### Results

Output:

```
[1, 1] Inside
[6, 6] Outside
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

Dimension = 2

SizeType = itk.Size[Dimension]
size = SizeType()
size.Fill(3)

IndexType = itk.Index[Dimension]
start = IndexType()
start.Fill(0)

RegionType = itk.ImageRegion[Dimension]
region = RegionType(start, size)

testPixel1 = IndexType()
testPixel1[0] = 1
testPixel1[1] = 1

testPixel2 = IndexType()
testPixel2[0] = 6
testPixel2[1] = 6

print(testPixel1, end=" ")
if region.IsInside(testPixel1):
    print("Inside")
else:
    print("Outside")

print(testPixel2, end=" ")
if region.IsInside(testPixel2):
    print("Inside")
else:
    print("Outside")
```

### C++

```
#include "itkImageRegion.h"
#include "itkIndex.h"
#include "itkSize.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;

  using RegionType = itk::ImageRegion<Dimension>;
  using SizeType = RegionType::SizeType;
  using IndexType = RegionType::IndexType;

  SizeType size;
  size.Fill(3);

  IndexType start;
  start.Fill(0);

  RegionType region(start, size);

  IndexType testPixel1;
  testPixel1[0] = 1;
  testPixel1[1] = 1;

  IndexType testPixel2;
  testPixel2[0] = 6;
  testPixel2[1] = 6;

  std::cout << testPixel1 << " ";

  if (region.IsInside(testPixel1))
  {
    std::cout << "Inside";
  }
  else
  {
    std::cout << "Outside";
    return EXIT_FAILURE;
  }
  std::cout << std::endl;

  std::cout << testPixel2 << " ";
  if (region.IsInside(testPixel2))
  {
    std::cout << "Inside";
    return EXIT_FAILURE;
  }
  else
  {
    std::cout << "Outside";
  }
  std::cout << std::endl;

  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

template<unsigned int **VImageDimension**>
**class ImageRegion** : **public** itk::Region

An image region represents a structured region of data.

ImageRegion is an class that represents some structured portion or piece of an Image. The ImageRegion is represented with an index and a size in each of the n-dimensions of the image. (The index is the corner of the image, the size is the lengths of the image in each of the topological directions.)

**See** Region

**See** Index

**See** Size

**See** MeshRegion

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- An object which holds the index (start) and size of a region of an image
- Determine image region intersection
- Determine if a pixel is inside of a region
- Determine the overlap of two regions

See itk::ImageRegion for additional documentation.

### Iterate Image Starting at Seed

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Iterate over an image starting at a seed and following a rule for connectivity decisions.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkFloodFilledImageFunctionConditionalIterator.h"
#include "itkBinaryThresholdImageFunction.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int /*argc*/, char * /*argv*/[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using FunctionType = itk::BinaryThresholdImageFunction<ImageType, double>;
  FunctionType::Pointer function = FunctionType::New();
  function->SetInputImage(image);
  function->ThresholdAbove(100); // we are looking to capture 255

  using IteratorType = itk::FloodFilledImageFunctionConditionalIterator<ImageType,␣
↪FunctionType>;

  itk::Index<2> seed;
  seed[0] = 25;
  seed[1] = 25;

  std::vector<itk::Index<2>> seeds;
  seeds.push_back(seed);

  IteratorType it(image, function, seeds);
  it.GoToBegin();

  while (!it.IsAtEnd())
  {
    std::cout << it.GetIndex() << std::endl;
    ++it;
  }

  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
void
CreateImage(ImageType::Pointer image)
{
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // The line doesn't work at the moment, because it needs 8-connectivity.

  // Make a line
  for (unsigned int i = 20; i < 50; ++i)
  {
    itk::Index<2> pixelIndex;
    pixelIndex.Fill(i);

    image->SetPixel(pixelIndex, 255);
  }

  // Make a square
  //   for(unsigned int r = 20; r < 50; r++)
  //     {
  //     for(unsigned int c = 20; c < 50; c++)
  //       {
  //       itk::Index<2> pixelIndex;
  //       pixelIndex[0] = r;
  //       pixelIndex[1] = c;
  //
  //       image->SetPixel(pixelIndex, 255);
  //       }
  //     }
}
```

### Classes demonstrated

template<typename **TImage**, typename **TFunction**>
**class FloodFilledImageFunctionConditionalIterator** : **public** itk::FloodFilledImageFunctionConditionalConstIt

Iterates over a flood-filled image function with write access to pixels.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Image Starting At Seed

See itk::FloodFilledImageFunctionConditionalIterator for additional documentation.

### Iterate Line Through Image

### Synopsis

Iterate over a line through an image.

### Results


Fig. 28: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkRGBPixel.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"
#include "itkLineIterator.h"

#include "itksys/SystemTools.hxx"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using PixelType = itk::RGBPixel<unsigned char>;
using ImageType = itk::Image<PixelType, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  std::string        inputFilename;
```

(continues on next page)

(continued from previous page)

```
  if (argc < 2)
  {
    CreateImage(image);
    inputFilename = "Synthetic";
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
    inputFilename = argv[1];
  }

  PixelType pixel;
  pixel.SetRed(255);
  pixel.SetGreen(127);
  pixel.SetBlue(50);

  ImageType::RegionType region = image->GetLargestPossibleRegion();
  ImageType::IndexType  corner1 = region.GetIndex();
  ImageType::IndexType  corner2;

  corner2[0] = corner1[0] + region.GetSize()[0] - 1;
  corner2[1] = corner1[1] + region.GetSize()[1] - 1;

  itk::LineIterator<ImageType> it1(image, corner1, corner2);
  it1.GoToBegin();
  while (!it1.IsAtEnd())
  {

    it1.Set(pixel);
    ++it1;
  }

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  if (argc > 1)
  {
    viewer.AddImage<ImageType>(image, true, itksys::SystemTools::
↪GetFilenameName(argv[1]));
  }
  else
  {
    viewer.AddImage<ImageType>(image, true, "Synthetic");
  }
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::SizeType regionSize;
  regionSize.Fill(100);
```

(continues on next page)

(continued from previous page)

```
  ImageType::IndexType regionIndex;
  regionIndex.Fill(0);

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  PixelType pixel;
  pixel.SetRed(0);
  pixel.SetGreen(127);
  pixel.SetBlue(200);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(pixel);
}
```

### Classes demonstrated

template<typename **TImage**>
**class LineIterator** : **public** itk::*LineConstIterator<TImage>*

An iterator that walks a Bresenham line through an ND image with write access to pixels.

LineIterator is an iterator that walks a Bresenham line through an image. The iterator is constructed similar to other image iterators, except instead of specifying a region to traverse, you specify two indices. The interval specified by the two indices is closed. So, a line iterator specified with the same start and end index will visit exactly one pixel.

```
LineConstIterator<ImageType> it(image, I1, I2);
while (!it.IsAtEnd())
{
   // visits at least 1 pixel
}
```

**Author** Benjamin King, Experimentelle Radiologie, Medizinische Hochschule Hannover.

**See** LineConstIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Line Through Image

See itk::LineIterator for additional documentation.

### Iterate Line Through Image Without Write Access

### Synopsis

Iterate over a line through an image without write access.

### Results

Output:

```
255
255
255
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"
#include "itkLineConstIterator.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  ImageType::RegionType region = image->GetLargestPossibleRegion();
  ImageType::IndexType  corner1 = region.GetIndex();
  ImageType::IndexType  corner2;
  corner2[0] = corner1[0] + region.GetSize()[0] - 1;
  corner2[1] = corner1[1] + region.GetSize()[1] - 1;

  itk::LineConstIterator<ImageType> it(image, corner1, corner2);
  it.GoToBegin();
  while (!it.IsAtEnd())
  {
    std::cout << (int)it.Get() << std::endl;
    ++it;
  }

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
```

(continues on next page)

(continued from previous page)

```
{
  ImageType::SizeType regionSize;
  regionSize.Fill(3);

  ImageType::IndexType regionIndex;
  regionIndex.Fill(0);

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(255);
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TImage**>
**class LineConstIterator**

An iterator that walks a Bresenham line through an ND image with read-only access to pixels.

LineConstIterator is an iterator that walks a Bresenham line through an image. The iterator is constructed similar to other image iterators, except instead of specifying a region to traverse, you specify two indices. The interval specified by the two indices is closed. So, a line iterator specified with the same start and end index will visit exactly one pixel.

```
LineConstIterator<ImageType> it(image, I1, I2);
while (!it.IsAtEnd())
{
   // visits at least 1 pixel
}
```

**Author** Benjamin King, Experimentelle Radiologie, Medizinische Hochschule Hannover.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Line Through Image Without Write Access

Subclassed by itk::LineIterator< TImage >

See itk::LineConstIterator for additional documentation.

### Iterate on a Vector Container

### Synopsis

basic operation on VectorContainer

### Results

Output:

```
[1, 2]
[2, 3]
```

### Code

### C++

```
#include "itkVectorContainer.h"
#include "itkPoint.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using CoordType = double;

  using PointType = itk::Point<CoordType, Dimension>;
  using VectorContainerType = itk::VectorContainer<int, PointType>;

  PointType p0;
  p0[0] = 1.0;
  p0[1] = 2.0;

  PointType p1;
  p1[0] = 2.0;
  p1[1] = 3.0;

  VectorContainerType::Pointer points = VectorContainerType::New();
  points->Reserve(2);

  VectorContainerType::Iterator point = points->Begin();
  point->Value() = p0;

  ++point;
  point->Value() = p1;

  point = points->Begin();
  while (point != points->End())
  {
    std::cout << point->Value() << std::endl;
    ++point;
  }
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TElementIdentifier**, typename **TElement**>
**class VectorContainer** : **public** itk::*Object*, **private** std::vector<*TElement*>
Define a front-end to the STL "vector" container that conforms to the IndexedContainerInterface.

This is a full-fleged Object, so there is modification time, debug, and reference count information.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate On A Vector Container

**Template Parameters**

- `TElementIdentifier`: An INTEGRAL type for use in indexing the vector.
- `TElement`: The element type stored in the container.

See itk::VectorContainer for additional documentation.

### Iterate Over a Region With a Shaped Neighborhood Iterator

### Synopsis

Iterate over a region of an image with a shaped neighborhood.

### Results

Output:

```
By default there are 0 active indices.
Now there are 2 active indices.
1 7
New position:
Centered at [0, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [0, -1]
Centered at [0, 0]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [0, 1]
New position:
Centered at [1, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [1, -1]
Centered at [1, 0]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [1, 1]
New position:
Centered at [2, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [2, -1]
Centered at [2, 0]
```

(continues on next page)

(continued from previous page)

```
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [2, 1]

[...]

New position:
Centered at [7, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [7, 8]
Centered at [7, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [7, 10]
New position:
Centered at [8, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [8, 8]
Centered at [8, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [8, 10]
New position:
Centered at [9, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [9, 8]
Centered at [9, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [9, 10]
```

## Code

### C++

```
#include "itkImage.h"
#include "itkShapedNeighborhoodIterator.h"
#include "itkImageRegionIterator.h"
#include "itkNeighborhoodAlgorithm.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;

  // Notice that char type pixel values will not appear
  // properly on the command prompt therefore for the
  // demonstration purposes it is best to use the int
  // type, however in real applications iterators have
  // no problems with char type images.
  // using ImageType = itk::Image<unsigned char, 2>;
  using PixelType = unsigned int;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::Pointer image = ImageType::New();

  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region(start, size);
```

(continues on next page)

(continued from previous page)

```
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  using IteratorType = itk::ShapedNeighborhoodIterator<ImageType>;
  IteratorType::RadiusType radius;
  radius.Fill(1);

  IteratorType iterator(radius, image, image->GetLargestPossibleRegion());
  std::cout << "By default there are " << iterator.GetActiveIndexListSize() << "␣
↪active indices." << std::endl;

  IteratorType::OffsetType top = { { 0, -1 } };
  iterator.ActivateOffset(top);
  IteratorType::OffsetType bottom = { { 0, 1 } };
  iterator.ActivateOffset(bottom);

  std::cout << "Now there are " << iterator.GetActiveIndexListSize() << " active␣
↪indices." << std::endl;

  IteratorType::IndexListType                indexList = iterator.
↪GetActiveIndexList();
  IteratorType::IndexListType::const_iterator listIterator = indexList.begin();
  while (listIterator != indexList.end())
  {
    std::cout << *listIterator << " ";
    ++listIterator;
  }
  std::cout << std::endl;

  // Note that ZeroFluxNeumannBoundaryCondition is used by default so even
  // pixels outside of the image will have valid values (equivalent to
  // their neighbors just inside the image)
  for (iterator.GoToBegin(); !iterator.IsAtEnd(); ++iterator)
  {
    std::cout << "New position: " << std::endl;
    IteratorType::ConstIterator ci = iterator.Begin();

    while (!ci.IsAtEnd())
    {
      std::cout << "Centered at " << iterator.GetIndex() << std::endl;
      std::cout << "Neighborhood index " << ci.GetNeighborhoodIndex() << " is offset
↪" << ci.GetNeighborhoodOffset()
                << " and has value " << ci.Get() << " The real index is "
                << iterator.GetIndex() + ci.GetNeighborhoodOffset() << std::endl;
      ++ci;
    }
  }

  std::cout << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**, typename **TBoundaryCondition** = ZeroFluxNeumannBoundaryCondition<*TImage*>>
**class ShapedNeighborhoodIterator** : **public** itk::ConstShapedNeighborhoodIterator<*TImage*, *TBoundaryCondition*>

A neighborhood iterator which can take on an arbitrary shape.

```
using ImageType = Image<float, 3>;
ShapedNeighborhoodIterator<ImageType> it(radius, image, region);
ShapedNeighborhoodIterator<ImageType>::OffsetType offset = {{0,0,0}};
it.ActivateOffset(offset);
```

**General Information** The ShapedNeighborhoodIterator is a refinement of NeighborhoodIterator which allows the user to specify which of the neighborhood elements are active and which will be ignored. This is useful for applications which only need to work with a subset of the neighborhood around a pixel such as morphological operations or cellular automata. This iterator can also be used, for example, to specify "cross-shaped" neighborhood where only elements along a spatial axes are significant.

**Constructing a shaped neighborhood iterator** A shaped neighborhood iterator is constructed by constructing a list of active neighbor locations. The list is called the ActiveIndexList. The methods ActivateOffset, DeactivateOffset, and ClearActiveList are used to construct the ActiveIndexList. The argument to Activate/DeactivateOffset is an itk::Offset which represents the ND spatial offset from the center of the neighborhood. For example, to activate the center pixel in the neighborhood, you would do the following:

where radius, image, and region are as described in NeighborhoodIterator.

Once a neighborhood location has been activated, iteration (operator++, operator–, operator+=, operator-=) will update the value at the active location. Note that values at inactive locations will NOT be valid if queried.

A second way to access active shaped neighborhood values is through a ShapedNeighborhoodIterator::Iterator or ConstShapedNeighborhoodIterator::ConstIterator. The following example demonstrates the use of these iterators.

**Accessing elements in a shaped neighborhood.** To access the value at an active neighborhood location, you can use the standard GetPixel, SetPixel methods. SetPixel is not defined for ConstShapedNeighborhoodIterator. The class will not complain if you attempt to access a value at a non-active location, but be aware that the result will be undefined. Error checking is not done in this case for the sake of efficiency.

```
using ImageType = Image<float, 3>;
ShapedNeighborhoodIterator<ImageType> it(radius, image, region);
...
it.ActivateOffset(offset1);
it.ActivateOffset(offset2);
it.ActivateOffset(offset3);
// etc..
...
ShapedNeighborhoodIterator<ImageType>::Iterator i;
for (i = it.Begin(); ! i.IsAtEnd(); ++i)
{ i.Set(i.Get() + 1.0); }
// you may also use i != i.End(), but IsAtEnd() may be slightly faster.
```

You can also iterate backward through the neighborhood active list.

```
i = it.End();
i--;
while (i != it.Begin())
{
  i.Set(i.Get() + 1.0);
```

(continues on next page)

(continued from previous page)

```
  i--;
}
 i.Set(i.Get() + 1.0);
```

The Get() Set() syntax was chosen versus defining operator* for these iterators because lvalue vs. rvalue context information is needed to determine whether bounds checking must take place.

**See** Neighborhood

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ShapedImageNeighborhoodRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Over A Region With A Shaped Neighborhood Iterator Manually
- Iterate Over A Region With A Shaped Neighborhood Iterator

See itk::ShapedNeighborhoodIterator for additional documentation.

### Iterate Over a Region With a Shaped Neighborhood Iterator Manually

### Synopsis

Iterate over a region of an image with a shaped neighborhood manually.

### Results

Output:

```
By default there are 0 active indices.
Now there are 2 active indices.
1 7
New position:
Centered at [0, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [0, -1]
Centered at [0, 0]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [0, 1]
New position:
Centered at [1, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [1, -1]
Centered at [1, 0]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [1, 1]
New position:
Centered at [2, 0]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [2, -1]
Centered at [2, 0]
```

(continues on next page)

(continued from previous page)

```
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [2, 1]

[...]

New position:
Centered at [7, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [7, 8]
Centered at [7, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [7, 10]
New position:
Centered at [8, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [8, 8]
Centered at [8, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [8, 10]
New position:
Centered at [9, 9]
Neighborhood index 1 is offset [0, -1] and has value 0 The real index is [9, 8]
Centered at [9, 9]
Neighborhood index 7 is offset [0, 1] and has value 0 The real index is [9, 10]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkShapedNeighborhoodIterator.h"
#include "itkImageRegionIterator.h"
#include "itkNeighborhoodAlgorithm.h"

// Notice that char type pixel values will not appear
// properly on the command prompt therefore for the
// demonstration purposes it is best to use the int
// type, however in real applications iterators have
// no problems with char type images.
// using ImageType = itk::Image<unsigned char, 2>;
using ImageType = itk::Image<unsigned int, 2>;

void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using IteratorType = itk::ShapedNeighborhoodIterator<ImageType>;

  itk::Size<2> radius;
  radius.Fill(1);
  IteratorType iterator(radius, image, image->GetLargestPossibleRegion());
  std::cout << "By default there are " << iterator.GetActiveIndexListSize() << "␣
↪active indices." << std::endl;
```

(continues on next page)

(continued from previous page)

```
  IteratorType::OffsetType top = { { 0, -1 } };
  iterator.ActivateOffset(top);
  IteratorType::OffsetType bottom = { { 0, 1 } };
  iterator.ActivateOffset(bottom);

  std::cout << "Now there are " << iterator.GetActiveIndexListSize() << " active␣
↪indices." << std::endl;

  IteratorType::IndexListType                 indexList = iterator.
↪GetActiveIndexList();
  IteratorType::IndexListType::const_iterator listIterator = indexList.begin();
  while (listIterator != indexList.end())
  {
    std::cout << *listIterator << " ";
    ++listIterator;
  }
  std::cout << std::endl;

  // Note that ZeroFluxNeumannBoundaryCondition is used by default so even
  // pixels outside of the image will have valid values (equivalent to their␣
↪neighbors just inside the image)
  for (iterator.GoToBegin(); !iterator.IsAtEnd(); ++iterator)
  {
    std::cout << "New position: " << std::endl;
    IteratorType::ConstIterator ci = iterator.Begin();

    while (!ci.IsAtEnd())
    {
      std::cout << "Centered at " << iterator.GetIndex() << std::endl;
      std::cout << "Neighborhood index " << ci.GetNeighborhoodIndex() << " is offset
↪" << ci.GetNeighborhoodOffset()
                << " and has value " << ci.Get() << " The real index is "
                << iterator.GetIndex() + ci.GetNeighborhoodOffset() << std::endl;
      ci++;
    }
  }

  std::cout << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);
}
```

### Classes demonstrated

template<typename **TImage**, typename **TBoundaryCondition** = ZeroFluxNeumannBoundaryCondition<*TImage*>>
**class ShapedNeighborhoodIterator** : **public** itk::ConstShapedNeighborhoodIterator<*TImage*, *TBoundaryCondition*>

A neighborhood iterator which can take on an arbitrary shape.

```
using ImageType = Image<float, 3>;
ShapedNeighborhoodIterator<ImageType> it(radius, image, region);
ShapedNeighborhoodIterator<ImageType>::OffsetType offset = {{0,0,0}};
it.ActivateOffset(offset);
```

**General Information** The ShapedNeighborhoodIterator is a refinement of NeighborhoodIterator which allows the user to specify which of the neighborhood elements are active and which will be ignored. This is useful for applications which only need to work with a subset of the neighborhood around a pixel such as morphological operations or cellular automata. This iterator can also be used, for example, to specify "cross-shaped" neighborhood where only elements along a spatial axes are significant.

**Constructing a shaped neighborhood iterator** A shaped neighborhood iterator is constructed by constructing a list of active neighbor locations. The list is called the ActiveIndexList. The methods ActivateOffset, DeactivateOffset, and ClearActiveList are used to construct the ActiveIndexList. The argument to Activate/DeactivateOffset is an itk::Offset which represents the ND spatial offset from the center of the neighborhood. For example, to activate the center pixel in the neighborhood, you would do the following:

where radius, image, and region are as described in NeighborhoodIterator.

Once a neighborhood location has been activated, iteration (operator++, operator–, operator+=, operator-=) will update the value at the active location. Note that values at inactive locations will NOT be valid if queried.

A second way to access active shaped neighborhood values is through a ShapedNeighborhoodIterator::Iterator or ConstShapedNeighborhoodIterator::ConstIterator. The following example demonstrates the use of these iterators.

**Accessing elements in a shaped neighborhood.** To access the value at an active neighborhood location, you can use the standard GetPixel, SetPixel methods. SetPixel is not defined for ConstShapedNeighborhoodIterator. The class will not complain if you attempt to access a value at a non-active location, but be aware that the result will be undefined. Error checking is not done in this case for the sake of efficiency.

```
using ImageType = Image<float, 3>;
ShapedNeighborhoodIterator<ImageType> it(radius, image, region);
...
it.ActivateOffset(offset1);
it.ActivateOffset(offset2);
it.ActivateOffset(offset3);
// etc..
...
ShapedNeighborhoodIterator<ImageType>::Iterator i;
for (i = it.Begin(); ! i.IsAtEnd(); ++i)
{ i.Set(i.Get() + 1.0); }
// you may also use i != i.End(), but IsAtEnd() may be slightly faster.
```

You can also iterate backward through the neighborhood active list.

```
i = it.End();
i--;
while (i != it.Begin())
{
  i.Set(i.Get() + 1.0);
```

(continues on next page)

(continued from previous page)

```
  i--;
}
 i.Set(i.Get() + 1.0);
```

The Get() Set() syntax was chosen versus defining operator* for these iterators because lvalue vs. rvalue context information is needed to determine whether bounds checking must take place.

**See** Neighborhood

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ShapedImageNeighborhoodRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Over A Region With A Shaped Neighborhood Iterator Manually
- Iterate Over A Region With A Shaped Neighborhood Iterator

See itk::ShapedNeighborhoodIterator for additional documentation.

### Iterate Over Image While Skipping Specific Region

### Synopsis

Iterator over an image skipping a specified region.

### Results

Output:

```
0
0
0
0
0
0
Visited 6
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionExclusionConstIteratorWithIndex.h"

namespace
{
using ImageType = itk::Image<int, 2>;
}

static void CreateImage(ImageType::Pointer);

int
main(int, char *[])
{

  ImageType::SizeType exclusionRegionSize;
  exclusionRegionSize.Fill(2);

  ImageType::IndexType exclusionRegionIndex;
  exclusionRegionIndex.Fill(0);

  ImageType::RegionType exclusionRegion(exclusionRegionIndex, exclusionRegionSize);

  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  itk::ImageRegionExclusionConstIteratorWithIndex<ImageType> imageIterator(image,␣
↪image->GetLargestPossibleRegion());
  imageIterator.SetExclusionRegion(exclusionRegion);

  unsigned int numberVisited = 0;
  while (!imageIterator.IsAtEnd())
  {
    std::cout << imageIterator.Get() << std::endl;
    ++imageIterator;
    ++numberVisited;
  }

  std::cout << "Visited " << numberVisited << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::SizeType regionSize;
  regionSize.Fill(3);

  ImageType::IndexType regionIndex;
  regionIndex.Fill(0);

  ImageType::RegionType region(regionIndex, regionSize);
```

(continues on next page)

(continued from previous page)

```
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRegionExclusionConstIteratorWithIndex** : **public** itk::*ImageRegionConstIteratorWithIndex<TImage>*

A multi-dimensional image iterator that walks an image region, excluding a second region that may partially or completely overlap the first, with read-only access to pixels.

ImageRegionExclusionConstIteratorWithIndex is a class templated over the image type that represents a multi-dimensional iterator. The exclusion region is set after construction. By default the exclusion region is empty and the iterator will behave as the itk::ImageRegionConstIteratorWithIndex with a penalty in performance due to internal bounds checking. There are no restrictions on valid exclusion regions.

As with other ITK image iterators, ImageRegionExclusionConstIteratorWithIndex requires that more information be specified before the iterator can be used than conventional iterators. Whereas the std::vector::iterator from the STL only needs to be passed a pointer to establish the iterator, the multi-dimensional image iterator needs a pointer, the size of the buffer, the size of the region, the start index of the buffer, and the start index of the region. To gain access to this information, ImageRegionExclusionConstIteratorWithIndex holds a reference to the image over which it is traversing.

ImageRegionExclusionConstIteratorWithIndex assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

The operator++ method provides a simple syntax for walking around a region of a multidimensional image. operator++ iterates across a row, constraining the movement to within a region of image. When the iterator reaches the boundary of the region along a row, the iterator automatically wraps to the next row, starting at the first pixel in the row that is part of the region. This allows for simple processing loops of the form:

```
IteratorType it( image, image->GetRequestedRegion() );

it.SetExclusionRegion( exclusionRegion );
it.GoToBegin();

while( ! it.IsAtEnd() )
{
  it.Set( 100.0 + it.Get() );
  ++it;
}
```

It also can be used for walking in the reverse direction like

```
IteratorType it( image, image->GetRequestedRegion() );

it.SetExclusionRegion( exclusionRegion );
it.GoToEnd();

while( !it.IsAtBegin() )
{
  it.Set( 100.0 );
```

(continues on next page)

(continued from previous page)

```
  --it;
}
```

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Over Image While Skipping Specific Region

Subclassed by itk::ImageRegionExclusionIteratorWithIndex< TImage >

See itk::ImageRegionExclusionConstIteratorWithIndex for additional documentation.

### Iterate Region in Image With Access to Index Without Write Access

### Synopsis

Iterate over a region of an image with efficient access to the current index (without write access).

### Results

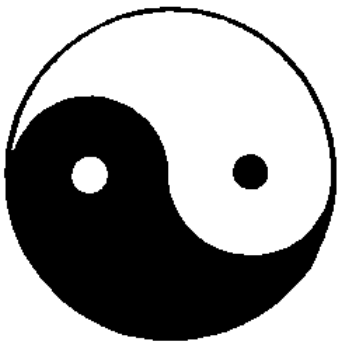


Fig. 29: Yinyang.png

Output:

```
An extensive list of the neighborhood will be printed to the screen.
```

Fig. 30: Yinyang.png In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionConstIteratorWithIndex.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 5;
  regionSize[1] = 4;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;
```

(continues on next page)

(continued from previous page)

```
  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  itk::ImageRegionConstIteratorWithIndex<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    std::cout << "Index: " << imageIterator.GetIndex() << " value: " <<␣
↪(int)imageIterator.Get() << std::endl;

    ++imageIterator;
  }

#ifdef ENABLE_QUICKVIEW
  // Visualize
  QuickView viewer;
  viewer.AddImage<ImageType>(image);
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRegionConstIteratorWithIndex** : **public** itk::ImageConstIteratorWithIndex<*TImage*>
A multi-dimensional iterator templated over image type that walks an image region and is specialized to keep track of its index location.

The "WithIndex" family of iteators was designed for algorithms that use both the values and locations of image pixels in calculations. Unlike ImageRegionIterator, which calculates an index only when requested, ImageRegionIteratorWithIndex maintains its index location as a member variable that is updated during increment and decrement operations. Iteration speed is penalized, but index queries become more efficient.

ImageRegionConstIteratorWithIndex is a multi-dimensional iterator, requiring more information be specified before the iterator can be used than conventional iterators. Whereas the std::vector::iterator from the STL only needs to be passed a pointer to establish the iterator, the multi-dimensional image iterator needs a pointer, the size of the buffer, the size of the region, the start index of the buffer, and the start index of the region. To gain access to this information, ImageRegionConstIteratorWithIndex holds a reference to the image over which it is traversing.

ImageRegionConstIteratorWithIndex assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

operator++ provides a simple syntax for walking around a region of a multidimensional image. operator++ iterates across a row, constraining the movement to within a region of image. When the iterator reaches the boundary of the region along a row, the iterator automatically wraps to the next row, starting at the first pixel in the row that is part of the region. This allows for simple processing loops of the form:

```
IteratorType it( image, image->GetRequestedRegion() );

it.Begin();

while( ! it.IsAtEnd() )
```

(continues on next page)

(continued from previous page)

```
{
  it.Set( 100.0 + it.Get() );
  ++it;
}
```

It also can be used for walking in the reverse direction like

```
IteratorType it( image, image->GetRequestedRegion() );

it.End();

while( !it.IsAtBegin() )
{
  it.Set( 100.0 );
  --it;
}
```

For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**MORE INFORMATION**

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ImageRegionRange

**See** ImageRegionIndexRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Access To Index Without Write Access

Subclassed by itk::FrequencyFFTLayoutImageRegionConstIteratorWithIndex< TImage >, itk::FrequencyHalfHermitianFFTLayoutImageRegionConstIteratorWithIndex< TImage >, itk::FrequencyImageRegionConstIteratorWithIndex< TImage >, itk::FrequencyShiftedFFTLayoutImageRegionConstIteratorWithIndex< TImage >, itk::ImageRegionExclusionConstIteratorWithIndex< TImage >, itk::ImageRegionIteratorWithIndex< TImage >

See itk::ImageRegionConstIteratorWithIndex for additional documentation.

### Iterate Region in Image With Access to Current Index With Write Access

### Synopsis

Iterate over a region of an image with efficient access to the current index (with write access).

### Results

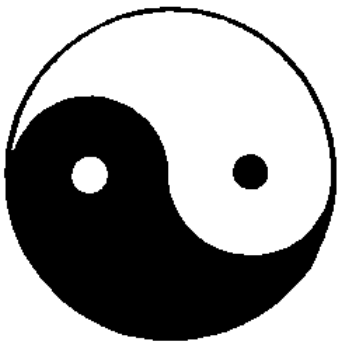


Fig. 31: Yinyang.png

Fig. 32: Yinyang.png In VTK Window With Index Access

Output:

```
Indices:
Index: [0, 0] value: ?
Index: [1, 0] value: ?
Index: [2, 0] value: ?
Index: [3, 0] value: ?
Index: [4, 0] value: ?
Index: [0, 1] value: ?
Index: [1, 1] value: ?
Index: [2, 1] value: ?
Index: [3, 1] value: ?
Index: [4, 1] value: ?
Index: [0, 2] value: ?
Index: [1, 2] value: ?
Index: [2, 2] value: ?
Index: [3, 2] value: ?
```

(continues on next page)

(continued from previous page)

```
Index: [4, 2] value: ?
Index: [0, 3] value: ?
Index: [1, 3] value: ?
Index: [2, 3] value: ?
Index: [3, 3] value: ?
Index: [4, 3] value: ?
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIteratorWithIndex.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 5;
  regionSize[1] = 4;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  itk::ImageRegionIteratorWithIndex<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    std::cout << "Index: " << imageIterator.GetIndex() << " value: " <<␣
↪(int)imageIterator.Get() << std::endl;
```

(continues on next page)

(continued from previous page)

```
    // Set the current pixel to white
    imageIterator.Set(255);

    ++imageIterator;
  }

#ifdef ENABLE_QUICKVIEW
  // Visualize
  QuickView viewer;
  viewer.AddImage<ImageType>(image);
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRegionIteratorWithIndex** : **public** itk::*ImageRegionConstIteratorWithIndex<TImage>*

A multi-dimensional iterator templated over image type that walks pixels within a region and is specialized to keep track of its image index location.

This class is a specialization of ImageRegionConstIteratorWithIndex that adds write-access (the Set() method). Please see ImageRegionConstIteratorWithIndex for more information.

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ImageRegionRange

**See** ImageRegionIndexRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Access To Current Index With Write Access

See itk::ImageRegionIteratorWithIndex for additional documentation.

### Iterate Region in Image With Neighborhood

### Synopsis

Iterate over a region of an image with a neighborhood (with write access).

### Results

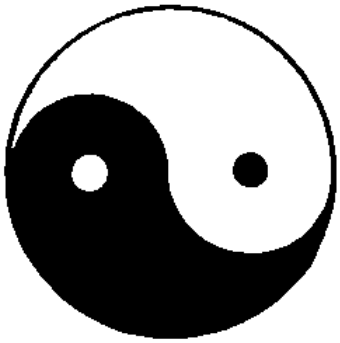


Fig. 33: Yinyang.png

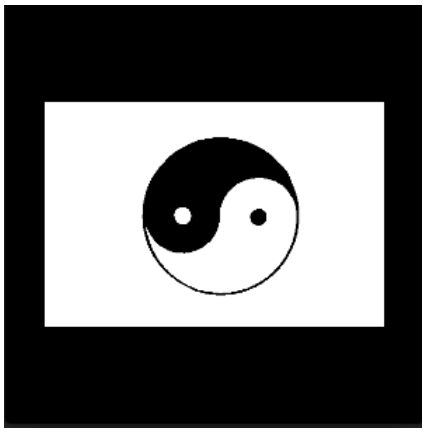


Fig. 34: Yinyang.png In VTK Window

Output:

```
An extensive list of the neighborhood will be printed to the screen.
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkNeighborhoodIterator.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 50;
  regionSize[1] = 1;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  ImageType::SizeType radius;
  radius[0] = 1;
  radius[1] = 1;

  itk::NeighborhoodIterator<ImageType> iterator(radius, image, region);


  while (!iterator.IsAtEnd())
```

(continues on next page)

(continued from previous page)

```
  {
    // Set the current pixel to white
    iterator.SetCenterPixel(255);

    for (unsigned int i = 0; i < 9; i++)
    {
      ImageType::IndexType index = iterator.GetIndex(i);
      std::cout << index[0] << " " << index[1] << std::endl;

      bool IsInBounds;
      iterator.GetPixel(i, IsInBounds);
      if (IsInBounds)
      {
        iterator.SetPixel(i, 255);
      }
    }
    ++iterator;
  }

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector = ConnectorType::New();
  connector->SetInput(image);

  vtkSmartPointer<vtkImageActor> actor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor->SetInput(connector->GetOutput());
#else
  connector->Update();
  actor->GetMapper()->SetInputData(connector->GetOutput());
#endif

  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(renderer);

  renderer->AddActor(actor);
  renderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();

  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**, typename **TBoundaryCondition** = ZeroFluxNeumannBoundaryCondition<*TImage*>>
**class NeighborhoodIterator** : **public** itk::*ConstNeighborhoodIterator*<*TImage*, *TBoundaryCondition*>

Defines iteration of a local N-dimensional neighborhood of pixels across an itk::Image.

This class is a loose extension of the Standard Template Library (STL) bi-directional iterator concept to *masks* of pixel neighborhoods within itk::Image objects. This NeighborhoodIterator base class defines simple forward and reverse iteration of an N-dimensional neighborhood mask across an image. Elements within the mask can be accessed like elements within an array.

NeighborhoodIterators are designed to encapsulate some of the complexity of working with image neighborhoods, complexity that would otherwise have to be managed at the algorithmic level. Use NeighborhoodIterators to simplify writing algorithms that perform geometrically localized operations on images (for example, convolution and morphological operations).

To motivate the discussion of NeighborhoodIterators and their use in Itk, consider the following code that takes directional derivatives at each point in an image.

```
itk::NeighborhoodInnerProduct<ImageType> innerProduct;

itk::DerivativeOperator<ImageType> operator;
 operator->SetOrder(1);
 operator->SetDirection(0);
 operator->CreateDirectional();

itk::NeighborhoodIterator<ImageType>
  iterator(operator->GetRadius(), myImage, myImage->GetRequestedRegion());

iterator.SetToBegin();
while ( ! iterator.IsAtEnd() )
{
  std::cout << "Derivative at index " << iterator.GetIndex() << is <<
    innerProduct(iterator, operator) << std::endl;
  ++iterator;
}
```

Most of the work for the programmer in the code above is in setting up for the iteration. There are three steps. First an inner product function object is created which will be used to effect convolution with the derivative kernel. Setting up the derivative kernel, DerivativeOperator, involves setting the order and direction of the derivative. Finally, we create an iterator over the RequestedRegion of the itk::Image (see Image) using the radius of the derivative kernel as the size.

Itk iterators only loosely follow STL conventions. Notice that instead of asking myImage for myImage.begin() and myImage.end(), iterator.SetToBegin() and iterator.IsAtEnd() are called. Itk iterators are typically more complex objects than traditional, pointer-style STL iterators, and the increased overhead required to conform to the complete STL API is not always justified.

The API for creating and manipulating a NeighborhoodIterator mimics that of the itk::ImageIterators. Like the itk::ImageIterator, a ConstNeighborhoodIterator is defined on a region of interest in an itk::Image. Iteration is constrained within that region of interest.

A NeighborhoodIterator is constructed as a container of pointers (offsets) to a geometric neighborhood of image pixels. As the central pixel position in the mask is moved around the image, the neighboring pixel pointers (offsets) are moved accordingly.

A *pixel neighborhood* is defined as a central pixel location and an N-dimensional radius extending outward from that location.

Pixels in a neighborhood can be accessed through a NeighborhoodIterator like elements in an array. For example, a 2D neighborhood with radius 2x1 has indices:

```
0  1  2  3  4
5  6  7  8  9
10 11 12 13 14
```

Now suppose a NeighborhoodIterator with the above dimensions is constructed and positioned over a neighborhood of values in an Image:

```
1.2 1.3 1.8 1.4 1.1
1.8 1.1 0.7 1.0 1.0
2.1 1.9 1.7 1.4 2.0
```

Shown below is some sample pixel access code and the values that it returns.

```
SizeValueType c = (SizeValueType) (iterator.Size() / 2); // get offset of center␣
↪pixel
SizeValueType s = iterator.GetStride(1);            // y-dimension step size

std::cout << iterator.GetPixel(7)      << std::endl;
std::cout << iterator.GetCenterPixel() << std::endl;
std::cout << iterator.GetPixel(c)      << std::endl;
std::cout << iterator.GetPixel(c-1)    << std::endl;
std::cout << iterator.GetPixel(c-s)    << std::endl;
std::cout << iterator.GetPixel(c-s-1)  << std::endl;
std::cout << *iterator[c]              << std::endl;
```

Results:

```
0.7
0.7
0.7
1.1
1.8
1.3
0.7
```

Use of GetPixel() is preferred over the *iterator[] form, and can be used without loss of efficiency in most cases. Some variations (subclasses) of NeighborhoodIterators may exist which do not support the latter API. Corresponding SetPixel() methods exist to modify pixel values in non-const NeighborhoodIterators.

NeighborhoodIterators are "bidirectional iterators". They move only in two directions through the data set. These directions correspond to the layout of the image data in memory and not to spatial directions of the N-dimensional itk::Image. Iteration always proceeds along the fastest increasing dimension (as defined by the layout of the image data). For itk::Image this is the first dimension specified (i.e. for 3-dimensional (x,y,z) NeighborhoodIterator proceeds along the x-dimension) (For random access iteration through N-dimensional indices, use RandomAccessNeighborhoodIterator).

Each subclass of a ConstNeighborhoodIterator may also define its own mechanism for iteration through an image. In general, the Iterator does not directly keep track of its spatial location in the image, but uses a set of internal loop variables and offsets to trigger wraps at itk::Image region boundaries, and to identify the end of the itk::Image region.

**See** DerivativeOperator

**See** NeighborhoodInnerProduct

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ShapedImageNeighborhoodRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Neighborhood
- Neighborhood Iterator On Vector Image

Subclassed by itk::ConstShapedNeighborhoodIterator< TImage, TBoundaryCondition >

See itk::NeighborhoodIterator for additional documentation.

### Iterate Region in Image With Neighborhood Without Write Access

### Synopsis

Iterate over a region of an image with a neighborhood (without write access).

### Results

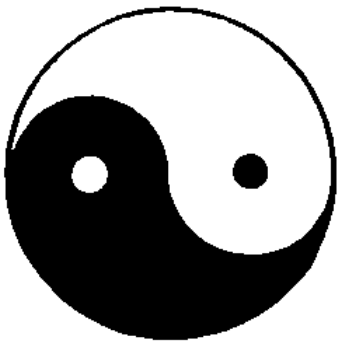


Fig. 35: Yinyang.png input image.

Output:

```
-1 -1
0 -1
1 -1
-1 0
0 0
1 0
```

(continues on next page)

(continued from previous page)

```
-1 1
0 1
1 1
0 -1
1 -1
2 -1
0 0
1 0
2 0
0 1
1 1
2 1
1 -1
2 -1
3 -1
1 0
2 0
3 0
1 1
2 1
3 1
2 -1
3 -1
4 -1
2 0
3 0
4 0
2 1
3 1
4 1
3 -1
4 -1
5 -1
3 0
4 0
5 0
3 1
4 1
5 1
4 -1
5 -1
6 -1
4 0
5 0
6 0
...
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConstNeighborhoodIterator.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 50;
  regionSize[1] = 1;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  ImageType::SizeType radius;
  radius[0] = 1;
  radius[1] = 1;

  itk::ConstNeighborhoodIterator<ImageType> iterator(radius, image, region);


  while (!iterator.IsAtEnd())
  {
    for (unsigned int i = 0; i < 9; i++)
    {
      ImageType::IndexType index = iterator.GetIndex(i);
      std::cout << index[0] << " " << index[1] << std::endl;

      bool IsInBounds;
      iterator.GetPixel(i, IsInBounds);
    }
    ++iterator;
  }
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**, typename **TBoundaryCondition** = ZeroFluxNeumannBoundaryCondition<*TImage*>>
**class ConstNeighborhoodIterator** : **public** itk::Neighborhood<*TImage*::InternalPixelType*, *TImage*::ImageDimension>

Const version of NeighborhoodIterator, defining iteration of a local N-dimensional neighborhood of pixels across an itk::Image.

ConstNeighborhoodIterator implements the read-only methods of NeighborhoodIterator. It serves as a base class from which other iterators are derived. See NeighborhoodIterator for more complete information.

**See** Neighborhood

**See** ImageIterator

**See** NeighborhoodIterator

**See** ShapedImageNeighborhoodRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Neighborhood Without Write Access

Subclassed by itk::NeighborhoodIterator< TImage, TBoundaryCondition >

See itk::ConstNeighborhoodIterator for additional documentation.

### Iterate Region in Image With Write Access

### Synopsis

Iterate over a region of an image (with write access).

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
```

(continues on next page)

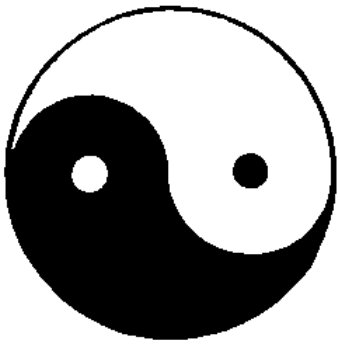


Fig. 36: Yinyang.png

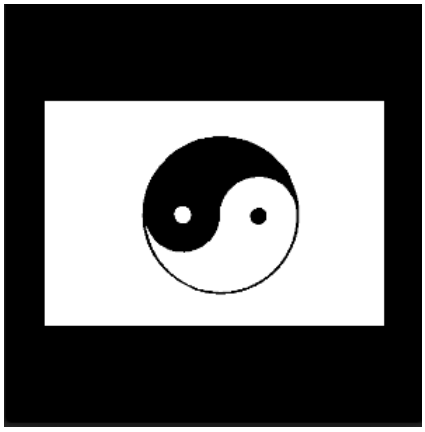


Fig. 37: Yinyang.png In VTK Window

(continued from previous page)

```
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 5;
  regionSize[1] = 4;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    // Get the value of the current pixel
    // unsigned char val = imageIterator.Get();
    // std::cout << (int)val << std::endl;

    // Set the current pixel to white
    imageIterator.Set(255);

    ++imageIterator;
  }

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector = ConnectorType::New();
  connector->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  vtkSmartPointer<vtkImageActor> actor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor->SetInput(connector->GetOutput());
#else
  connector->Update();
  actor->GetMapper()->SetInputData(connector->GetOutput());
#endif

  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(renderer);

  renderer->AddActor(actor);
  renderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();

  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRegionIterator** : **public** itk::*ImageRegionConstIterator*<*TImage*>

A multi-dimensional iterator templated over image type that walks a region of pixels.

The itk::ImageRegionIterator is optimized for iteration speed and is the first choice for iterative, pixel-wise operations on an image. ImageRegionIterator is the least specialized of the ITK image iterator classes. ImageRegionIterator is templated over the image type, and is constrained to walk only within the specified region and along a line parallel to one of the coordinate axes, "wrapping" to the next line as it reaches the boundary of the image. To walk the entire image, specify the region to be `image->GetRequestedRegion()`.

Most of the functionality is inherited from the ImageRegionConstIterator. The current class only adds write access to image pixels.

example ImageRegionIterator.cxx

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ImageRegionRange

**See** ImageRegionIndexRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Write Access

See itk::ImageRegionIterator for additional documentation.

### Iterate Region in Image Without Write Access

### Synopsis

Iterate over a region of an image (without write access).

### Results

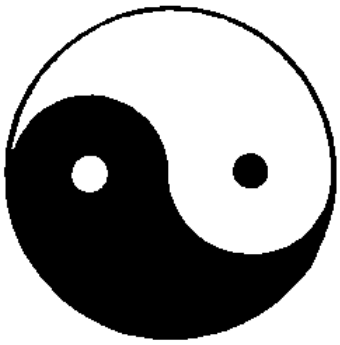


Fig. 38: Yinyang.png

Output:

```
255
255
255
255
255
255
255
255
255
255
255
255
255
```

(continues on next page)

(continued from previous page)

```
255
255
255
255
255
255
255
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRegionConstIterator.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  using ImageType = itk::Image<unsigned char, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  ImageType::SizeType regionSize;
  regionSize[0] = 5;
  regionSize[1] = 4;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  itk::ImageRegionConstIterator<ImageType> imageIterator(image, region);
  // itk::ImageRegionConstIterator<ImageType> imageIterator(image,image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    // Get the value of the current pixel
    unsigned char val = imageIterator.Get();
```

(continues on next page)

(continued from previous page)

```
    std::cout << (int)val << std::endl;

    ++imageIterator;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRegionConstIterator** : **public** itk::ImageConstIterator<*TImage*>

A multi-dimensional iterator templated over image type that walks a region of pixels.

The itk::ImageRegionConstIterator is optimized for iteration speed and is the first choice for iterative, pixel-wise operations on an image. ImageRegionIterator is the least specialized of the ITK image iterator classes. ImageRegionConstIterator is templated over the image type, and is constrained to walk only within the specified region and along a line parallel to one of the coordinate axes, "wrapping" to the next line as it reaches the boundary of the image. To walk the entire image, specify the region to be `image->GetRequestedRegion()`.

ImageRegionConstIterator provides read-only access to image data. It is the base class for the read/write access ImageRegionIterator.

ImageRegionConstIterator is a multi-dimensional iterator, requiring more information be specified before the iterator can be used than conventional iterators. Whereas the std::vector::iterator from the STL only needs to be passed a pointer to establish the iterator, the multi-dimensional image iterator needs a pointer, the size of the buffer, the size of the region, the start index of the buffer, and the start index of the region. To gain access to this information, ImageRegionConstIterator holds a reference to the image over which it is traversing.

ImageRegionConstIterator assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

operator++ provides a simple syntax for walking around a region of a multidimensional image. operator++ iterates across a row, constraining the movement to within a region of image. When the iterator reaches the boundary of the region along a row, the iterator automatically wraps to the next row, starting at the first pixel in the row that is part of the region. This allows for simple processing loops of the form:

```
it = it.Begin();
for (; !it.IsAtEnd(); ++it)
   {
   *it += 100.0;
   }
```

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ImageRegionRange

**See** ImageRegionIndexRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Iterate Region In Image Without Write Access

Subclassed by itk::ImageRegionIterator< TImage >

See itk::ImageRegionConstIterator for additional documentation.

### Make Out of Bounds Pixels Return Constant Value

### Synopsis

Make out of bounds pixels return a constant value.

### Results

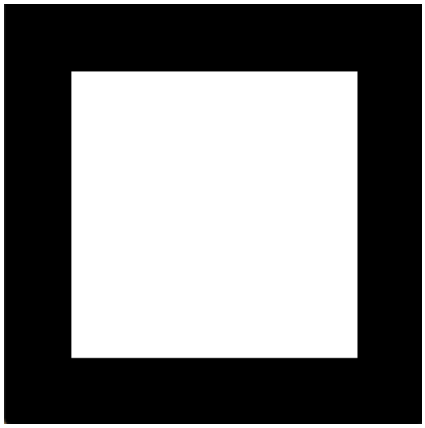


Fig. 39: Output In VTK Window

Output:

```
Index: [-1, -1] Pixel: 0 = 0
Index: [0, -1] Pixel: 1 = 0
Index: [1, -1] Pixel: 2 = 0
Index: [-1, 0] Pixel: 3 = 0
Index: [0, 0] Pixel: 4 = 255
Index: [1, 0] Pixel: 5 = 255
Index: [-1, 1] Pixel: 6 = 0
Index: [0, 1] Pixel: 7 = 255
Index: [1, 1] Pixel: 8 = 255
Index: [0, -1] Pixel: 0 = 0
Index: [1, -1] Pixel: 1 = 0
Index: [2, -1] Pixel: 2 = 0
Index: [0, 0] Pixel: 3 = 255
Index: [1, 0] Pixel: 4 = 255
Index: [2, 0] Pixel: 5 = 255
Index: [0, 1] Pixel: 6 = 255
Index: [1, 1] Pixel: 7 = 255
Index: [2, 1] Pixel: 8 = 255
Index: [1, -1] Pixel: 0 = 0
Index: [2, -1] Pixel: 1 = 0
Index: [3, -1] Pixel: 2 = 0
```

(continues on next page)

(continued from previous page)

```
Index: [1, 0] Pixel: 3 = 255
Index: [2, 0] Pixel: 4 = 255
Index: [3, 0] Pixel: 5 = 255
Index: [1, 1] Pixel: 6 = 255
Index: [2, 1] Pixel: 7 = 255
Index: [3, 1] Pixel: 8 = 255
Index: [2, -1] Pixel: 0 = 0
Index: [3, -1] Pixel: 1 = 0
Index: [4, -1] Pixel: 2 = 0
Index: [2, 0] Pixel: 3 = 255
Index: [3, 0] Pixel: 4 = 255
Index: [4, 0] Pixel: 5 = 255
Index: [2, 1] Pixel: 6 = 255
Index: [3, 1] Pixel: 7 = 255
Index: [4, 1] Pixel: 8 = 255
Index: [3, -1] Pixel: 0 = 0
Index: [4, -1] Pixel: 1 = 0
Index: [5, -1] Pixel: 2 = 0
Index: [3, 0] Pixel: 3 = 255
Index: [4, 0] Pixel: 4 = 255
Index: [5, 0] Pixel: 5 = 0
Index: [3, 1] Pixel: 6 = 255
Index: [4, 1] Pixel: 7 = 255
Index: [5, 1] Pixel: 8 = 0
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConstNeighborhoodIterator.h"
#include "itkConstantBoundaryCondition.h"
#include "itkImageRegionIterator.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
```

(continues on next page)

(continued from previous page)

```
  CreateImage(image);

  ImageType::SizeType regionSize;
  regionSize[0] = 50;
  regionSize[1] = 1;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  ImageType::SizeType radius;
  radius[0] = 1;
  radius[1] = 1;

  using BoundaryConditionType = itk::ConstantBoundaryCondition<ImageType>;
  itk::ConstNeighborhoodIterator<ImageType, BoundaryConditionType> iterator(radius,␣
↪image, region);

  while (!iterator.IsAtEnd())
  {
    for (unsigned int i = 0; i < 9; i++)
    {
      ImageType::IndexType index = iterator.GetIndex(i);

      std::cout << "Index: " << index << " Pixel: " << i << " = " << (int)iterator.
↪GetPixel(i) << std::endl;
    }
    ++iterator;
  }

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector = ConnectorType::New();
  connector->SetInput(image);

  vtkSmartPointer<vtkImageActor> actor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor->SetInput(connector->GetOutput());
#else
  connector->Update();
  actor->GetMapper()->SetInputData(connector->GetOutput());
#endif

  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  vtkSmartPointer<vtkRenderer> renderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(renderer);
```

(continues on next page)

(continued from previous page)

```
  renderer->AddActor(actor);
  renderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();

  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 5;
  unsigned int        NumCols = 5;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  // Set all pixels to white
  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(255);
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class ConstantBoundaryCondition** : **public** itk::ImageBoundaryCondition<*TInputImage*, *TOutputImage*>
This boundary condition returns a constant value for out-of-bounds image pixels.

For example, invoking this function object with a constant value of zero (the default) on each out-of-bounds element of a 7x5 iterator that masks a region at an image corner (iterator is centered on the 2):

```
* * * * * * *
* * * * * * *
* * 1 2 3 4 5  (where * denotes pixels that lie
* * 3 3 5 5 6          outside of the image boundary)
* * 4 4 6 7 8
```

would produce the following neighborhood of values:

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 2 3 4 5
0 0 3 3 5 5 6
0 0 4 4 6 7 8
```

**Note** If you are using an image with Array as the pixel type, you will need to set the constant explicitly with an array of the appropriate length. This is also true if your image type is a VectorImage.

**See** ImageBoundaryCondition

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Make Out Of Bounds Pixels Return Constant Value

See itk::ConstantBoundaryCondition for additional documentation.

### Make Part of an Image Transparent

### Synopsis

Demonstrates how to assign transparency to pixels in an image, then writing the result to a tif file for inspection and verification.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkTIFFImageIO.h"
```

(continues on next page)

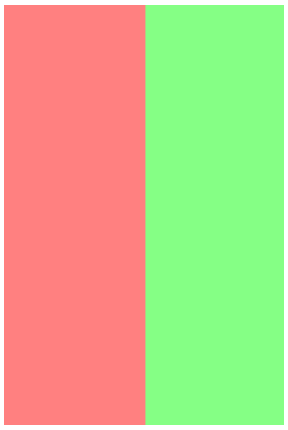


Fig. 40: Output Image.

(continued from previous page)

```
#include "itkRGBAPixel.h"

int
main(int argc, char * argv[])
{
  std::string outputFilename;
  if (argc > 1)
  {
    outputFilename = argv[1];
  }
  else
  {
    outputFilename = "test.tif";
  }

  using PixelType = itk::RGBAPixel<unsigned char>;
  using ImageType = itk::Image<PixelType, 2>;

  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
```

(continues on next page)

(continued from previous page)

```
    ImageType::PixelType pixel = imageIterator.Get();

    if (imageIterator.GetIndex()[0] > 100)
    {
      pixel.SetRed(0);
      pixel.SetGreen(255);
      pixel.SetBlue(0);
      // pixel.SetAlpha(255); // invisible
      pixel.SetAlpha(122);
    }
    else
    {
      pixel.SetRed(255);
      pixel.SetGreen(0);
      pixel.SetBlue(0);
      pixel.SetAlpha(static_cast<unsigned char>(0.5 * 255));
    }
    imageIterator.Set(pixel);

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  using TIFFIOType = itk::TIFFImageIO;
  WriterType::Pointer writer = WriterType::New();
  TIFFIOType::Pointer tiffIO = TIFFIOType::New();
  tiffIO->SetPixelType(itk::IOPixelEnum::RGBA);
  writer->SetFileName(outputFilename);
  writer->SetInput(image);
  writer->SetImageIO(tiffIO);
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TComponent** = unsigned short>
**class RGBAPixel** : **public** itk::*FixedArray*<*TComponent*, 4>

Represent Red, Green, Blue and Alpha components for color images.

This class is templated over the representation used for each component.

The following syntax for assigning an index is allowed/suggested:

```
RGBAPixel<float> pixel; pixel = 1.0f, 0.0f, .5f, .8;
RGBAPixel<char> pixelArray[2];
pixelArray[0] = 255, 255, 255, 230;
pixelArray[1] = 255, 255, 244, 255;
```

Since RGBAPixel is a subclass of Array, you can access its components as: pixel[0], pixel[1], pixel[2], pixel[3]

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Make Part Of An Image Transparent

See itk::RGBAPixel for additional documentation.

### Matrix

### Synopsis

This will create and display a matrix of NxN dimensions, then multiply it by a vector of N dimension.

### Results

Output:

```
M: 1 2 3
   4 5 6
   7 8 9

M: 1 2 3
   4 5 6
   7 8 9

V: [1, 2, 3]
MV: [14, 32, 50]
```

### Code

### C++

```
#include <itkMatrix.h>
#include <itkVector.h>

#include <iostream>

static void
Construct();
// static void ConstructRunTimeDims();
static void
Multiply();
// static void Inverse();

int
main(int, char *[])
{
  Construct();
  Multiply();
  return EXIT_SUCCESS;
}

void
Construct()
{
```

(continues on next page)

(continued from previous page)

```
  using MatrixType = itk::Matrix<double, 3, 3>;
  MatrixType M;
  M(0, 0) = 1.0;
  M(0, 1) = 2.0;
  M(0, 2) = 3.0;
  M(1, 0) = 4.0;
  M(1, 1) = 5.0;
  M(1, 2) = 6.0;
  M(2, 0) = 7.0;
  M(2, 1) = 8.0;
  M(2, 2) = 9.0;

  std::cout << "M: " << M << std::endl;
}

/*
void ConstructRunTimeDims()
{
  int matrixSize = 3;
  using MatrixType = itk::Matrix<double, matrixSize, matrixSize>;
  MatrixType M;
  M(0,0) = 1.0;
  M(0,1) = 2.0;
  M(0,2) = 3.0;
  M(1,0) = 4.0;
  M(1,1) = 5.0;
  M(1,2) = 6.0;
  M(2,0) = 7.0;
  M(2,1) = 8.0;
  M(2,2) = 9.0;

  std::cout << "M: " << M << std::endl;
}
*/

void
Multiply()
{
  using MatrixType = itk::Matrix<double, 3, 3>;
  MatrixType M;
  M(0, 0) = 1.0;
  M(0, 1) = 2.0;
  M(0, 2) = 3.0;
  M(1, 0) = 4.0;
  M(1, 1) = 5.0;
  M(1, 2) = 6.0;
  M(2, 0) = 7.0;
  M(2, 1) = 8.0;
  M(2, 2) = 9.0;

  std::cout << "M: " << M << std::endl;

  using VectorType = itk::Vector<double, 3>;
  VectorType V;
  V[0] = 1.0;
  V[1] = 2.0;
  V[2] = 3.0;
```

(continues on next page)

(continued from previous page)

```
  std::cout << "V: " << V << std::endl;

  std::cout << "MV: " << M * V << std::endl;
}

/*
void Inverse()
{

}
*/
```

## Classes demonstrated

template<typename **T**, unsigned int **NRows** = 3, unsigned int **NColumns** = 3>
**class Matrix**

A templated class holding a M x N size Matrix.

This class contains a vnl_matrix_fixed in order to make all the vnl mathematical methods available.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Matrix
- Matrix Inverse

See itk::Matrix for additional documentation.

## Matrix Inverse

## Synopsis

This will compute and display an NxN matrix and its inverse.

## Results

Output:

```
M: 1 2
   3 5
Inverse: -5 2
          3 -1
```

### Code

### C++

```
#include <itkMatrix.h>

#include <iostream>

int
main(int, char *[])
{
  using MatrixType = itk::Matrix<double, 2, 2>;
  MatrixType M;
  M(0, 0) = 1.0;
  M(0, 1) = 2.0;
  M(1, 0) = 3.0;
  M(1, 1) = 5.0;

  std::cout << "M: " << M << std::endl;

  MatrixType Minv(M.GetInverse());

  std::cout << "Inverse: " << Minv << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **T**, unsigned int **NRows** = 3, unsigned int **NColumns** = 3>
**class Matrix**

A templated class holding a M x N size Matrix.

This class contains a vnl_matrix_fixed in order to make all the vnl mathematical methods available.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Matrix
- Matrix Inverse

See itk::Matrix for additional documentation.

### Mersenne Twister Random Integer Generator

### Synopsis

Random integer generator

### Results

### Code

### C++

```
#include "itkMersenneTwisterRandomVariateGenerator.h"

int
main(int, char *[])
{
  using GeneratorType = itk::Statistics::MersenneTwisterRandomVariateGenerator;
  GeneratorType::Pointer generator = GeneratorType::New();
  generator->Initialize();

  // Get an int between 0 and 5
  // (inclusive - that is sample from the set {0,1,2,3,4,5})
  std::cout << generator->GetIntegerVariate(5) << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class MersenneTwisterRandomVariateGenerator** : **public** itk::Statistics::RandomVariateGeneratorBase
MersenneTwisterRandom random variate generator.

It is recommended to create a separate object in each thread. By default, each instantiated class will have a different seed created by the GetNextSeed method. The creation of the initial seeds are initialized once from the time. For deterministic behavior, the individual instances' seeds should be manual set to separate values in each thread.

It is no longer recommended to use this class using a "Singleton-like" GetInstance method for the global instance of this class. This usage may result in unsafe concurrent access to the global instance.

This notice was included with the original implementation. The only changes made were to obfuscate the author's email addresses.

**Warning** This class's instance methods are NEITHER reentrant nor concurrent thread-safe, except where marked as thread-safe. That is to say you can still use separate objects concurrently.

MersenneTwister.h Mersenne Twister random number generator a C++ class MTRand Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus Richard J. Wagner v1.0 15 May 2003 rjwagner at writeme dot com

The Mersenne Twister is an algorithm for generating random numbers. It was designed with consideration of the flaws in various other generators. The period, 2^19937-1, and the order of equidistribution, 623 dimensions,

are far greater. The generator is also fast; it avoids multiplication and division, and it benefits from caches and pipelines. For more information see the inventors' web page at http:*www.math.keio.ac.jp/~matumoto/emt.html

Reference M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally

Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30.



The original code included the following notice:

```
When you use this, send an email to: matumoto at math dot keio dot ac dot jp
with an appropriate reference to your work.
```

It would be nice to CC: rjwagner at writeme dot com and Cokus at math dot washington dot edu when you write.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mersenne Twister Random Number Generator

See itk::Statistics::MersenneTwisterRandomVariateGenerator for additional documentation.

### Mersenne Twister Random Number Generator

### Synopsis

Generate a random number

### Results

### Code

### C++

```
#include "itkMersenneTwisterRandomVariateGenerator.h"

int
main(int, char *[])
{
  using GeneratorType = itk::Statistics::MersenneTwisterRandomVariateGenerator;
  GeneratorType::Pointer generator = GeneratorType::New();

  generator->Initialize();

  std::cout << generator->GetUniformVariate(0, 5) << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class MersenneTwisterRandomVariateGenerator** : **public** itk::Statistics::RandomVariateGeneratorBase
MersenneTwisterRandom random variate generator.

It is recommended to create a separate object in each thread. By default, each instantiated class will have a different seed created by the GetNextSeed method. The creation of the initial seeds are initialized once from the time. For deterministic behavior, the individual instances' seeds should be manual set to separate values in each thread.

It is no longer recommended to use this class using a "Singleton-like" GetInstance method for the global instance of this class. This usage may result in unsafe concurrent access to the global instance.

This notice was included with the original implementation. The only changes made were to obfuscate the author's email addresses.

**Warning** This class's instance methods are NEITHER reentrant nor concurrent thread-safe, except where marked as thread-safe. That is to say you can still use separate objects concurrently.

MersenneTwister.h Mersenne Twister random number generator a C++ class MTRand Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus Richard J. Wagner v1.0 15 May 2003 rjwagner at writeme dot com

The Mersenne Twister is an algorithm for generating random numbers. It was designed with consideration of the flaws in various other generators. The period, 2^19937-1, and the order of equidistribution, 623 dimensions, are far greater. The generator is also fast; it avoids multiplication and division, and it benefits from caches and pipelines. For more information see the inventors' web page at http:*www.math.keio.ac.jp/~matumoto/emt.html

Reference M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally

Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30.



The original code included the following notice:

```
When you use this, send an email to: matumoto at math dot keio dot ac dot jp
with an appropriate reference to your work.
```

It would be nice to CC: rjwagner at writeme dot com and Cokus at math dot washington dot edu when you write.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mersenne Twister Random Number Generator

See itk::Statistics::MersenneTwisterRandomVariateGenerator for additional documentation.

### Mini Pipeline

### Synopsis

Mini Pipeline.

### Results

Output:

```
Input:
ImageRegion (0x7fe4c3aaadb0)
Dimension: 2
Index: [0, 0]
Size: [200, 300]

Input:
ImageRegion (0x7fe4c3aab920)
Dimension: 2
Index: [0, 0]
Size: [200, 300]
```

Fig. 41: Output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "ImageFilterY.h"

template <typename TImage>
static void
CreateImage(TImage * const image);

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::ImageFilter<ImageType>;
```

(continues on next page)

(continued from previous page)

```
  ImageType::Pointer image = ImageType::New();
  CreateImage(image.GetPointer());

  std::cout << "Input:" << std::endl;
  std::cout << image->GetLargestPossibleRegion() << std::endl;
  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->Update();

  std::cout << "Input:" << std::endl;
  std::cout << filter->GetOutput()->GetLargestPossibleRegion() << std::endl;

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("Output.png");
  writer->SetInput(filter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

template <typename TImage>
void
CreateImage(TImage * const image)
{
  // Create an image with 2 connected components
  typename TImage::IndexType corner = { { 0, 0 } };

  unsigned int              NumRows = 200;
  unsigned int              NumCols = 300;
  typename TImage::SizeType size = { { NumRows, NumCols } };

  typename TImage::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make another square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      typename TImage::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Multiple Inputs of Different Type

### Synopsis

Write a filter with multiple inputs of different types.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCovariantVector.h"

#include "ImageFilterMultipleInputsDifferentType.h"

int
main(int, char *[])
{
  // Setup types
  using VectorImageType = itk::Image<itk::CovariantVector<unsigned char, 3>, 2>;
  using ScalarImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::ImageFilterMultipleInputsDifferentType<VectorImageType,␣
↪ScalarImageType>;

  using ReaderType = itk::ImageFileReader<VectorImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName("Test.jpg");
  reader->Update();

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->Update();

  using WriterType = itk::ImageFileWriter<VectorImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("TestOutput.jpg");
  writer->SetInput(filter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >,

itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage

>, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Multiple Inputs of Same Type

### Synopsis

Write a filter with multiple inputs of the same type.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkImageFilterMultipleInputs.h"

int
main(int, char *[])
```

(continues on next page)

(continued from previous page)

```
{
  // Setup types
  using ImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::ImageFilterMultipleInputs<ImageType>;

  ImageType::Pointer image = ImageType::New();
  ImageType::Pointer mask = ImageType::New();

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->SetInputImage(image);
  filter->SetInputMask(mask);
  filter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOut-

putImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter<

TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage >, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Multiple Outputs of Different Type

### Synopsis

Write a filter with multiple outputs of different types.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "ImageFilterMultipleOutputsDifferentType.h"

int
main(int, char *[])
{
  // Setup types
  using InputImageType = itk::Image<unsigned char, 2>;
  using OutputImageType1 = itk::Image<float, 2>;
  using OutputImageType2 = itk::Image<int, 2>;
  using FilterType = itk::ImageFilterMultipleOutputsDifferentType<InputImageType,␣
↪OutputImageType1, OutputImageType2>;

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->Update();

  {
    using WriterType = itk::ImageFileWriter<OutputImageType1>;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("TestOutput1.jpg");
    writer->SetInput(filter->GetOutput1());
    writer->Update();
  }

  {
    using WriterType = itk::ImageFileWriter<OutputImageType2>;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("TestOutput2.jpg");
    writer->SetInput(filter->GetOutput2());
    writer->Update();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >,

itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage

>, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Multiple Outputs of Same Type

### Synopsis

Write a filter with multiple outputs of the same type.

### Results

Fig. 42: TestOuput1.jpg



Fig. 43: TestOutput2.jpg

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
```

(continues on next page)

(continued from previous page)

```
#include "ImageFilterMultipleOutputs.h"

int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::ImageFilterMultipleOutputs<ImageType>;

  // Create and the filter
  FilterType::Pointer filter = FilterType::New();
  filter->Update();

  {
    using WriterType = itk::ImageFileWriter<ImageType>;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("TestOutput1.jpg");
    writer->SetInput(filter->GetOutput1());
    writer->Update();
  }

  {
    using WriterType = itk::ImageFileWriter<ImageType>;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName("TestOutput2.jpg");
    writer->SetInput(filter->GetOutput2());
    writer->Update();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type
- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter<

TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasurement >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage

>, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter< TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage >, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >,

itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Multi-thread Oil Painting

### Synopsis

Multi-threaded oil painting image filter.

### Results

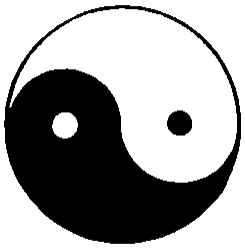


Fig. 44: Input image

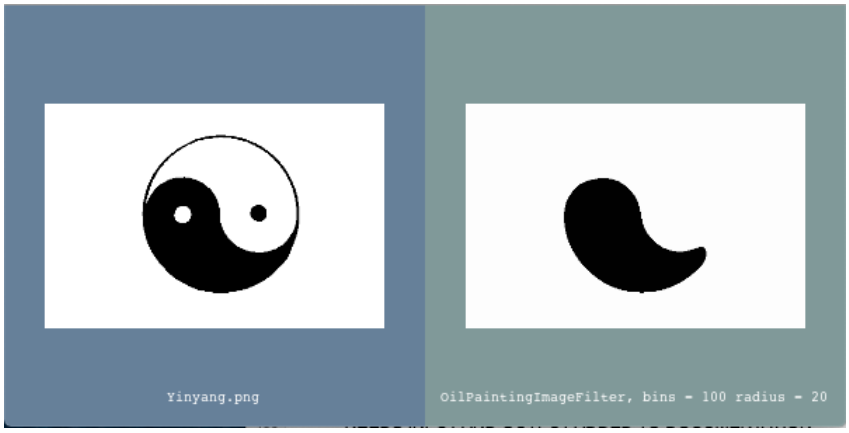



Fig. 45: Output In QuickView Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"

#include "itkOilPaintingImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0] << " inputFile [bins [radius]]" << std::endl;
    return EXIT_FAILURE;
  }
  unsigned int numberOfBins = 50;
  if (argc >= 3)
  {
    numberOfBins = std::stoi(argv[2]);
  }

  unsigned int radius = 2;
  if (argc >= 4)
  {
    radius = std::stoi(argv[3]);
  }

  using ImageType = itk::Image<unsigned char, 2>;
  using FilterType = itk::OilPaintingImageFilter<ImageType>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfBins(numberOfBins);
  filter->SetRadius(radius);
  filter->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(reader->GetFileName()));

  std::stringstream desc;
  desc << "OilPaintingImageFilter, bins = " << numberOfBins << " radius = " << radius;
  viewer.AddImage(filter->GetOutput(), true, desc.str());

  viewer.Visualize();
```

(continues on next page)

(continued from previous page)

```
#endif

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageToImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>, **private** itk::ImageToImageFilterCommon

Base class for filters that take an image as input and produce an image as output.

ImageToImageFilter is the base class for all process objects that output image data and require image data as input. Specifically, this class defines the SetInput() method for defining the input to a filter.

This class provides the infrastructure for supporting multithreaded processing of images. If a filter provides an implementation of GenerateData(), the image processing will run in a single thread and the implementation is responsible for allocating its output data. If a filter provides an implementation of ThreadedGenerateData() instead, the image will be divided into a number of work units, a number of threads will be spawned, and ThreadedGenerateData() will be called in each thread. Here, the output memory will be allocated by this superclass prior to calling ThreadedGenerateData().

ImageToImageFilter provides an implementation of GenerateInputRequestedRegion(). The base assumption to this point in the hierarchy is that a process object would ask for the largest possible region on input in order to produce any output. Imaging filters, however, can usually answer this question more precisely. The default implementation of GenerateInputRequestedRegion() in this class is to request an input that matches the size of the requested output. If a filter requires more input (say a filter that uses neighborhood information) or less input (for instance a magnify filter), then these filters will have to provide another implementation of this method. By convention, such implementations should call the Superclass' method first.

All inputs to ImageToImageFilter (if there is more than one) are checked in the VerifyInputInformation method to verify that they occupy the same physical space. If the input images are in the same physical space, then the location of each voxel is identical, and the filter can operate voxel-by-voxel in index space. Some filters registration filters, for example will violate this precondition, in which case they should redefine VerifyInputInformation to relax or eliminate this requirement.

Access methods Set/GetCoordinateTolerance and Set/GetDirectionTolerance are provided for cases where the default spatial-congruency tolerances are too fine, and images that are almost in the same space should be regard as being in the same space. This has come up, for example when comparing different on-disk image formats with differing digits of precision in the position, spacing, and orientation.

The default tolerance is govern by the GlobalDefaultCoordinateTolerance and the GlobalDefaultDirectionTolerance properties, defaulting to 1.0e-6. The default tolerance for spatial comparison is then scaled by the voxelSpacing for coordinates (i.e. the coordinates must be the same to within one part per million). For the direction cosines the values must be within the current absolute tolerance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Filter Image
- Multiple Inputs Of Same Type
- Multiple Inputs Of Different Type

- Multiple Outputs Of Same Type
- Multi-thread Oil Painting
- Multiple Outputs Of Different Type
- Filter Image Using Multiple Threads

Subclassed by itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::greater< TInputImage::PixelType > >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, std::less< TInputImage::PixelType > >, itk::ConvolutionImageFilterBase< TInputImage, TKernelSource::OutputImageType, TOutputImage >, itk::AccumulateImageFilter< TInputImage, TOutputImage >, itk::ApproximateSignedDistanceMapImageFilter< TInputImage, TOutputImage >, itk::AttributeMorphologyBaseImageFilter< TInputImage, TOutputImage, TAttribute, TFunction >, itk::BilateralImageFilter< TInputImage, TOutputImage >, itk::BinaryImageToLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::BinaryImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::BinaryMedianImageFilter< TInputImage, TOutputImage >, itk::BinaryPruningImageFilter< TInputImage, TOutputImage >, itk::BinaryThinningImageFilter< TInputImage, TOutputImage >, itk::BinomialBlurImageFilter< TInputImage, TOutputImage >, itk::BinShrinkImageFilter< TInputImage, TOutputImage >, itk::BoxImageFilter< TInputImage, TOutputImage >, itk::BSplineControlPointImageFilter< TInputImage, TOutputImage >, itk::BSplineDecompositionImageFilter< TInputImage, TOutputImage >, itk::BSplineResampleImageFilterBase< TInputImage, TOutputImage >, itk::CannyEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ClosingByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::CollidingFrontsImageFilter< TInputImage, TOutputImage >, itk::ComposeDisplacementFieldsImageFilter< TInputImage, TOutputImage >, itk::ComposeImageFilter< TInputImage, TOutputImage >, itk::ConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::ConnectedComponentImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::ConnectedThresholdImageFilter< TInputImage, TOutputImage >, itk::ConvolutionImageFilterBase< TInputImage, TKernelImage, TOutputImage >, itk::CyclicShiftImageFilter< TInputImage, TOutputImage >, itk::DanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::DerivativeImageFilter< TInputImage, TOutputImage >, itk::DirectFourierReconstructionImageToImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianDerivativeImageFilter< TInputImage, TOutputImage >, itk::DiscreteGaussianImageFilter< TInputImage, TOutputImage >, itk::DisplacementFieldJacobianDeterminantFilter< TInputImage, TRealType, TOutputImage >, itk::DisplacementFieldToBSplineImageFilter< TInputImage, TInputPointSet, TOutputImage >, itk::DoubleThresholdImageFilter< TInputImage, TOutputImage >, itk::ExpandImageFilter< TInputImage, TOutputImage >, itk::ExponentialDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::FastChamferDistanceImageFilter< TInputImage, TOutputImage >, itk::ForwardFFTImageFilter< TInputImage, TOutputImage >, itk::GradientMagnitudeImageFilter< TInputImage, TOutputImage >, itk::GradientRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::GradientVectorFlowImageFilter< TInputImage, TOutputImage, TInternalPixel >, itk::GrayscaleConnectedClosingImageFilter< TInputImage, TOutputImage >, itk::GrayscaleConnectedOpeningImageFilter< TInputImage, TOutputImage >, itk::GrayscaleFillholeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicDilateImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGeodesicErodeImageFilter< TInputImage, TOutputImage >, itk::GrayscaleGrindPeakImageFilter< TInputImage, TOutputImage >, itk::HalfHermitianToRealInverseFFTImageFilter< TInputImage, TOutputImage >, itk::HardConnectedComponentImageFilter< TInputImage, TOutputImage >, itk::HConcaveImageFilter< TInputImage, TOutputImage >, itk::HConvexImageFilter< TInputImage, TOutputImage >, itk::HessianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::HessianToObjectnessMeasureImageFilter< TInputImage, TOutputImage >, itk::HistogramMatchingImageFilter< TInputImage, TOutputImage, THistogramMeasure-

ment >, itk::HistogramThresholdImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::HMaximaImageFilter< TInputImage, TOutputImage >, itk::HMinimaImageFilter< TInputImage, TOutputImage >, itk::ImageAndPathToImageFilter< TInputImage, TInputPath, TOutputImage >, itk::ImageShapeModelEstimatorBase< TInputImage, TOutputImage >, itk::InPlaceImageFilter< TInputImage, TOutputImage >, itk::InterpolateImageFilter< TInputImage, TOutputImage >, itk::InterpolateImagePointsFilter< TInputImage, TOutputImage, TCoordType, InterpolatorType >, itk::InverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::InverseFFTImageFilter< TInputImage, TOutputImage >, itk::InvertDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::IsoContourDistanceImageFilter< TInputImage, TOutputImage >, itk::IsolatedConnectedImageFilter< TInputImage, TOutputImage >, itk::IsolatedWatershedImageFilter< TInputImage, TOutputImage >, itk::IterativeInverseDisplacementFieldImageFilter< TInputImage, TOutputImage >, itk::JoinSeriesImageFilter< TInputImage, TOutputImage >, itk::KappaSigmaThresholdImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::LabelImageToLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToShapeLabelMapFilter< TInputImage, TOutputImage >, itk::LabelImageToStatisticsLabelMapFilter< TInputImage, TFeatureImage, TOutputImage >, itk::LabelMapFilter< TInputImage, TOutputImage >, itk::LabelMapToAttributeImageFilter< TInputImage, TOutputImage, TAttributeAccessor >, itk::LabelVotingImageFilter< TInputImage, TOutputImage >, itk::LaplacianImageFilter< TInputImage, TOutputImage >, itk::LaplacianRecursiveGaussianImageFilter< TInputImage, TOutputImage >, itk::LaplacianSharpeningImageFilter< TInputImage, TOutputImage >, itk::LevelSetDomainMapImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage, TMaskImage >, itk::MorphologicalWatershedImageFilter< TInputImage, TOutputImage >, itk::MRIBiasFieldCorrectionFilter< TInputImage, TOutputImage, TMaskImage >, itk::MultiLabelSTAPLEImageFilter< TInputImage, TOutputImage, TWeights >, itk::MultiResolutionPyramidImageFilter< TInputImage, TOutputImage >, itk::MultiScaleHessianBasedMeasureImageFilter< TInputImage, THessianImage, TOutputImage >, itk::N4BiasFieldCorrectionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::NeighborhoodConnectedImageFilter< TInputImage, TOutputImage >, itk::NeighborhoodOperatorImageFilter< TInputImage, TOutputImage, TOperatorValueType >, itk::NormalizeImageFilter< TInputImage, TOutputImage >, itk::NormalizeToConstantImageFilter< TInputImage, TOutputImage >, itk::ObjectMorphologyImageFilter< TInputImage, TOutputImage, TKernel >, itk::OpeningByReconstructionImageFilter< TInputImage, TOutputImage, TKernel >, itk::OrientImageFilter< TInputImage, TOutputImage >, itk::OtsuMultipleThresholdsImageFilter< TInputImage, TOutputImage >, itk::PadImageFilterBase< TInputImage, TOutputImage >, itk::PatchBasedDenoisingBaseImageFilter< TInputImage, TOutputImage >, itk::PolylineMask2DImageFilter< TInputImage, TPolyline, TOutputImage >, itk::PolylineMaskImageFilter< TInputImage, TPolyline, TVector, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, TAccumulator >, itk::RealToHalfHermitianForwardFFTImageFilter< TInputImage, TOutputImage >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, TCompare >, itk::RegionalMaximaImageFilter< TInputImage, TOutputImage >, itk::RegionalMinimaImageFilter< TInputImage, TOutputImage >, itk::RegionGrowImageFilter< TInputImage, TOutputImage >, itk::RegionOfInterestImageFilter< TInputImage, TOutputImage >, itk::ResampleImageFilter< TInputImage, TOutputImage, TInterpolatorPrecisionType, TTransformPrecisionType >, itk::ResampleInPlaceImageFilter< TInputImage, TOutputImage >, itk::RobustAutomaticThresholdImageFilter< TInputImage, TGradientImage, TOutputImage >, itk::ScalarImageKmeansImageFilter< TInputImage, TOutputImage >, itk::ScalarToRGBColormapImageFilter< TInputImage, TOutputImage >, itk::ShiftScaleImageFilter< TInputImage, TOutputImage >, itk::ShrinkImageFilter< TInputImage, TOutputImage >, itk::SignedDanielssonDistanceMapImageFilter< TInputImage, TOutputImage, TVoronoiImage >, itk::SignedMaurerDistanceMapImageFilter< TInputImage, TOutputImage >, itk::SliceBySliceImageFilter< TInputImage, TOutputImage, TInputFilter, TOutputFilter, TInternalInputImage, TInternalOutputImage >, itk::SliceImageFilter< TInputImage, TOutputImage >, itk::SLICImageFilter< TInputImage, TOutputImage, TDistancePixel >, itk::SobelEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::SpatialFunctionImageEvaluatorFilter< TSpatialFunction, TInputImage, TOutputImage >, itk::STAPLEImageFilter< TInputImage, TOutputImage >, itk::Statistics::ImageClassifierFilter< TSample, TInputImage, TOutputImage >, itk::StochasticFractalDimensionImageFilter< TInputImage, TMaskImage, TOutputImage >, itk::StreamingImageFilter< TInputImage, TOutputImage >, itk::Testing::ComparisonImageFilter<

TInputImage, TOutputImage >, itk::ThresholdMaximumConnectedComponentsImageFilter< TInputImage, TOutputImage >, itk::TileImageFilter< TInputImage, TOutputImage >, itk::TobogganImageFilter< TInputImage, TOutputImage >, itk::UnsharpMaskImageFilter< TInputImage, TOutputImage, TInternalPrecision >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, TFunction1, TFunction2 >, itk::VectorConfidenceConnectedImageFilter< TInputImage, TOutputImage >, itk::VectorGradientMagnitudeImageFilter< TInputImage, TRealType, TOutputImage >, itk::VectorNeighborhoodOperatorImageFilter< TInputImage, TOutputImage >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage, TBinaryPriorImage >, itk::VotingBinaryImageFilter< TInputImage, TOutputImage >, itk::WarpImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::WarpVectorImageFilter< TInputImage, TOutputImage, TDisplacementField >, itk::ZeroCrossingBasedEdgeDetectionImageFilter< TInputImage, TOutputImage >, itk::ZeroCrossingImageFilter< TInputImage, TOutputImage >, itk::MaskedFFTNormalizedCorrelationImageFilter< TInputImage, TOutputImage >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Function::BinaryThresholdAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::BinaryAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MaximumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MeanAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MedianAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::MinimumAccumulator< TInputImage::PixelType > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::StandardDeviationAccumulator< TInputImage::PixelType, TAccumulate > >, itk::ProjectionImageFilter< TInputImage, TOutputImage, Functor::SumAccumulator< TInputImage::PixelType, TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::greater< TOutputImage::PixelType > >, itk::ReconstructionImageFilter< TInputImage, TOutputImage, std::less< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::greater< TInputImage::PixelType >, std::greater< TOutputImage::PixelType > >, itk::ValuedRegionalExtremaImageFilter< TInputImage, TOutputImage, std::less< TInputImage::PixelType >, std::less< TOutputImage::PixelType > >, itk::VoronoiSegmentationImageFilterBase< TInputImage, TOutputImage >

See itk::ImageToImageFilter for additional documentation.

### Neighborhood Iterator on Vector Image

### Synopsis

NeighborhoodIterator on a VectorImage.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkNeighborhoodIterator.h"
```

(continues on next page)

(continued from previous page)

```
using VectorImageType = itk::VectorImage<unsigned char, 2>;

int
main(int, char *[])
{
  // Create an image
  VectorImageType::Pointer image = VectorImageType::New();

  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(10);

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->SetNumberOfComponentsPerPixel(3);
  image->Allocate();

  // Create the neighborhood iterator
  VectorImageType::SizeType radius;
  radius[0] = 1;
  radius[1] = 1;

  itk::NeighborhoodIterator<VectorImageType> iterator(radius, image, image->
↪GetLargestPossibleRegion());

  while (!iterator.IsAtEnd())
  {
    iterator.GetCenterPixel();

    ++iterator;
  }


  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 3>
**class VectorImage** : **public** itk::ImageBase<*VImageDimension*>
Templated n-dimensional vector image class.

This class differs from Image in that it is intended to represent multiple images. Each pixel represents *k* measurements, each of datatype *TPixel*. The memory organization of the resulting image is as follows: ... Pi0 Pi1 Pi2 Pi3 P(i+1)0 P(i+1)1 P(i+1)2 P(i+1)3 P(i+2)0 ... where Pi0 represents the 0th measurement of the pixel at index i.

Conceptually, a `VectorImage< TPixel, 3 >` is the same as a `Image< VariableLengthVector< TPixel >, 3 >`. The difference lies in the memory organization. The latter results in a fragmented organization with each location in the Image holding a pointer to an `VariableLengthVector` holding the actual pixel. The former stores the *k* pixels instead of a pointer reference, which apart from avoiding fragmentation

of memory also avoids storing a 8 bytes of pointer reference for each pixel. The parameter *k* can be set using `SetVectorLength`.

The API of the class is such that it returns a pixeltype VariableLengthVector< TPixel > when queried, with the data internally pointing to the buffer. (the container does not manage the memory). Similarly SetPixel calls can be made with VariableLengthVector< TPixel >.

The API of this class is similar to Image.

**Caveats:** When using Iterators on this image, you cannot use the it.Value(). You must use Set/Get() methods instead.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**See** DefaultVectorPixelAccessor

**See** DefaultVectorPixelAccessorFunctor

**See** VectorImageToImagePixelAccessor

**See** VectorImageToImageAdaptor

**See** Image

**See** ImportImageContainer

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cast Vector Image To Another Type
- Create Vector Image
- Neighborhood Iterator On Vector Image

See itk::VectorImage for additional documentation.

template<typename **TImage**, typename **TBoundaryCondition** = ZeroFluxNeumannBoundaryCondition<*TImage*>>
**class NeighborhoodIterator** : **public** itk::*ConstNeighborhoodIterator*<*TImage*, *TBoundaryCondition*>

Defines iteration of a local N-dimensional neighborhood of pixels across an itk::Image.

This class is a loose extension of the Standard Template Library (STL) bi-directional iterator concept to *masks* of pixel neighborhoods within itk::Image objects. This NeighborhoodIterator base class defines simple forward and reverse iteration of an N-dimensional neighborhood mask across an image. Elements within the mask can be accessed like elements within an array.

NeighborhoodIterators are designed to encapsulate some of the complexity of working with image neighborhoods, complexity that would otherwise have to be managed at the algorithmic level. Use NeighborhoodIterators to simplify writing algorithms that perform geometrically localized operations on images (for example, convolution and morphological operations).

To motivate the discussion of NeighborhoodIterators and their use in Itk, consider the following code that takes directional derivatives at each point in an image.

```
itk::NeighborhoodInnerProduct<ImageType> innerProduct;

itk::DerivativeOperator<ImageType> operator;
 operator->SetOrder(1);
 operator->SetDirection(0);
```

(continues on next page)

(continued from previous page)

```
 operator->CreateDirectional();

itk::NeighborhoodIterator<ImageType>
  iterator(operator->GetRadius(), myImage, myImage->GetRequestedRegion());

iterator.SetToBegin();
while ( ! iterator.IsAtEnd() )
{
  std::cout << "Derivative at index " << iterator.GetIndex() << is <<
    innerProduct(iterator, operator) << std::endl;
  ++iterator;
}
```

Most of the work for the programmer in the code above is in setting up for the iteration. There are three steps. First an inner product function object is created which will be used to effect convolution with the derivative kernel. Setting up the derivative kernel, DerivativeOperator, involves setting the order and direction of the derivative. Finally, we create an iterator over the RequestedRegion of the itk::Image (see Image) using the radius of the derivative kernel as the size.

Itk iterators only loosely follow STL conventions. Notice that instead of asking myImage for myImage.begin() and myImage.end(), iterator.SetToBegin() and iterator.IsAtEnd() are called. Itk iterators are typically more complex objects than traditional, pointer-style STL iterators, and the increased overhead required to conform to the complete STL API is not always justified.

The API for creating and manipulating a NeighborhoodIterator mimics that of the itk::ImageIterators. Like the itk::ImageIterator, a ConstNeighborhoodIterator is defined on a region of interest in an itk::Image. Iteration is constrained within that region of interest.

A NeighborhoodIterator is constructed as a container of pointers (offsets) to a geometric neighborhood of image pixels. As the central pixel position in the mask is moved around the image, the neighboring pixel pointers (offsets) are moved accordingly.

A *pixel neighborhood* is defined as a central pixel location and an N-dimensional radius extending outward from that location.

Pixels in a neighborhood can be accessed through a NeighborhoodIterator like elements in an array. For example, a 2D neighborhood with radius 2x1 has indices:

```
0  1  2  3  4
5  6  7  8  9
10 11 12 13 14
```

Now suppose a NeighborhoodIterator with the above dimensions is constructed and positioned over a neighborhood of values in an Image:

```
1.2 1.3 1.8 1.4 1.1
1.8 1.1 0.7 1.0 1.0
2.1 1.9 1.7 1.4 2.0
```

Shown below is some sample pixel access code and the values that it returns.

```
SizeValueType c = (SizeValueType) (iterator.Size() / 2); // get offset of center␣
↪pixel
SizeValueType s = iterator.GetStride(1);            // y-dimension step size

std::cout << iterator.GetPixel(7)      << std::endl;
std::cout << iterator.GetCenterPixel() << std::endl;
```

(continues on next page)

(continued from previous page)

```
std::cout << iterator.GetPixel(c)      << std::endl;
std::cout << iterator.GetPixel(c-1)    << std::endl;
std::cout << iterator.GetPixel(c-s)    << std::endl;
std::cout << iterator.GetPixel(c-s-1)  << std::endl;
std::cout << *iterator[c]              << std::endl;
```

Results:

```
0.7
0.7
0.7
1.1
1.8
1.3
0.7
```

Use of GetPixel() is preferred over the *iterator[] form, and can be used without loss of efficiency in most cases. Some variations (subclasses) of NeighborhoodIterators may exist which do not support the latter API. Corresponding SetPixel() methods exist to modify pixel values in non-const NeighborhoodIterators.

NeighborhoodIterators are "bidirectional iterators". They move only in two directions through the data set. These directions correspond to the layout of the image data in memory and not to spatial directions of the N-dimensional itk::Image. Iteration always proceeds along the fastest increasing dimension (as defined by the layout of the image data). For itk::Image this is the first dimension specified (i.e. for 3-dimensional (x,y,z) NeighborhoodIterator proceeds along the x-dimension) (For random access iteration through N-dimensional indices, use RandomAccessNeighborhoodIterator).

Each subclass of a ConstNeighborhoodIterator may also define its own mechanism for iteration through an image. In general, the Iterator does not directly keep track of its spatial location in the image, but uses a set of internal loop variables and offsets to trigger wraps at itk::Image region boundaries, and to identify the end of the itk::Image region.

**See** DerivativeOperator

**See** NeighborhoodInnerProduct

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**See** ShapedImageNeighborhoodRange

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Iterate Region In Image With Neighborhood
- Neighborhood Iterator On Vector Image

Subclassed by itk::ConstShapedNeighborhoodIterator< TImage, TBoundaryCondition >

See itk::NeighborhoodIterator for additional documentation.

### Observe an Event

### Synopsis

This example demonstrates how to observe and event that is invoked by a filter.

### Results

Output:

```
Progress: 0
Progress: 0.00994873
Progress: 0.0198975
Progress: 0.0298462
Progress: 0.0397949
Progress: 0.0497437
Progress: 0.0596924
Progress: 0.0696411
Progress: 0.0795898
Progress: 0.0895386
Progress: 0.0994873
Progress: 0.109436
Progress: 0.119385
Progress: 0.129333
Progress: 0.139282
Progress: 0.149231
Progress: 0.15918
Progress: 0.169128
Progress: 0.179077
Progress: 0.189026
Progress: 0.198975
Progress: 0.208923
Progress: 0.218872
Progress: 0.228821
Progress: 0.23877
Progress: 0.248718
Progress: 0.258667
Progress: 0.268616
Progress: 0.278564
Progress: 0.288513
Progress: 0.298462
Progress: 0.308411
Progress: 0.318359
Progress: 0.328308
Progress: 0.338257
Progress: 0.348206
Progress: 0.358154
Progress: 0.368103
Progress: 0.378052
Progress: 0.388
Progress: 0.397949
Progress: 0.407898
Progress: 0.417847
Progress: 0.427795
Progress: 0.437744
Progress: 0.447693
```

(continues on next page)

(continued from previous page)

```
Progress: 0.457642
Progress: 0.46759
Progress: 0.477539
Progress: 0.487488
Progress: 0.497437
Progress: 0.507385
Progress: 0.517334
Progress: 0.527283
Progress: 0.537231
Progress: 0.54718
Progress: 0.557129
Progress: 0.567078
Progress: 0.577026
Progress: 0.586975
Progress: 0.596924
Progress: 0.606873
Progress: 0.616821
Progress: 0.62677
Progress: 0.636719
Progress: 0.646667
Progress: 0.656616
Progress: 0.666565
Progress: 0.676514
Progress: 0.686462
Progress: 0.696411
Progress: 0.70636
Progress: 0.716309
Progress: 0.726257
Progress: 0.736206
Progress: 0.746155
Progress: 0.756104
Progress: 0.766052
Progress: 0.776001
Progress: 0.78595
Progress: 0.795898
Progress: 0.805847
Progress: 0.815796
Progress: 0.825745
Progress: 0.835693
Progress: 0.845642
Progress: 0.855591
Progress: 0.86554
Progress: 0.875488
Progress: 0.885437
Progress: 0.895386
Progress: 0.905334
Progress: 0.915283
Progress: 0.925232
Progress: 0.935181
Progress: 0.945129
Progress: 0.955078
Progress: 0.965027
Progress: 0.974976
Progress: 0.984924
Progress: 0.994873
Progress: 1
```

### Code

### Python

```
#!/usr/bin/env python

import itk

Dimension = 2
PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

source = itk.GaussianImageSource[ImageType].New()
size = itk.Size[Dimension]()
size.Fill(128)
source.SetSize(size)

sigma = itk.FixedArray[itk.D, Dimension]()
sigma.Fill(45.0)
source.SetSigma(sigma)


def myCommand():
    print("Progress: " + str(source.GetProgress()))


source.AddObserver(itk.ProgressEvent(), myCommand)

source.Update()
```

### C++

```
#include "itkImage.h"
#include "itkGaussianImageSource.h"
#include "itkCommand.h"

class MyCommand : public itk::Command
{
public:
  itkNewMacro(MyCommand);

  void
  Execute(itk::Object * caller, const itk::EventObject & event) override
  {
    Execute((const itk::Object *)caller, event);
  }

  void
  Execute(const itk::Object * caller, const itk::EventObject & event) override
  {
    if (!itk::ProgressEvent().CheckEvent(&event))
    {
      return;
    }
```

(continues on next page)

(continued from previous page)

```
    const auto * processObject = dynamic_cast<const itk::ProcessObject *>(caller);
    if (!processObject)
    {
      return;
    }
    std::cout << "Progress: " << processObject->GetProgress() << std::endl;
  }
};


int
main(int, char *[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using SourceType = itk::GaussianImageSource<ImageType>;
  SourceType::Pointer source = SourceType::New();

  ImageType::SizeType size;
  size.Fill(128);
  source->SetSize(size);

  SourceType::ArrayType sigma;
  sigma.Fill(45.0);
  source->SetSigma(sigma);

  MyCommand::Pointer myCommand = MyCommand::New();
  source->AddObserver(itk::ProgressEvent(), myCommand);

  source->Update();

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

**class Command** : **public** itk::*Object*

Superclass for callback/observer methods.

Command is an implementation of the command design pattern that is used in callbacks (such as StartMethod(), ProgressMethod(), and EndMethod()) in ITK. itk::Object implements a Subject/Observer pattern. When a subject needs to notify a observer, it does so using a itk::Command. The Execute method is called to run the command.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Observe An Event

Subclassed by itk::CommandIterationUpdate< TOptimizer >, itk::CommandIterationUpdatev4< TOptimizer >, itk::CommandVnlIterationUpdate< TOptimizer >, itk::CStyleCommand, itk::FunctionCommand,

itk::MemberCommand< T >, itk::ReceptorMemberCommand< T >, itk::SimpleConstMemberCommand< T >, itk::SimpleMemberCommand< T >, itk::WatershedMiniPipelineProgressCommand

See itk::Command for additional documentation.

### Pass Image to Function

### Synopsis

Pass an image to a function.

### Code

### C++

```
#include <itkImage.h>

#include <iostream>

using ImageType = itk::Image<float, 2>;
static void
myStandardPointer(const ImageType *)
{}

static void
mySmartPointer(const ImageType::Pointer)
{}

template <typename TImage>
static void
TemplateSmartPointer(const typename TImage::Pointer)
{}

template <typename TImage>
static void
TemplateStandardPointer(const TImage *)
{}

int
main(int, char *[])
{
  ImageType::Pointer  image = ImageType::New();
  itk::Index<2>       corner = { { 0, 0 } };
  itk::Size<2>        size = { { 10, 10 } };
  itk::ImageRegion<2> region(corner, size);
  image->SetRegions(region);
  image->Allocate();

  // A function that accepts a standard pointer can be passed either a standard or a␣
↪smart pointer
  myStandardPointer(image.GetPointer());
  myStandardPointer(image);

  // A function that accepts a smart pointer can be passed either a standard or a␣
↪smart pointer
```

(continues on next page)

(continued from previous page)

```
  mySmartPointer(image.GetPointer());
  mySmartPointer(image);

  return 0;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception

- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Permute Sequence of Indices

### Synopsis

Permute a sequence of indices.

### Results

Output:

```
1 0 4 3 2

After shuffle
4 1 0 2 3
```

### Code

### C++

```
#include <itkImageRandomNonRepeatingConstIteratorWithIndex.h>

int
main(int, char *[])
{
  itk::RandomPermutation rp(5);

  std::cout << std::endl;

  for (unsigned int i = 0; i < 5; i++)
  {
    std::cout << rp[i] << " ";
  }
  std::cout << std::endl << std::endl;

  rp.Shuffle();
  std::cout << "After shuffle" << std::endl;
  for (unsigned int i = 0; i < 5; i++)
  {
    std::cout << rp[i] << " ";
  }
```

(continues on next page)

(continued from previous page)

```
  std::cout << std::endl;


  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRandomNonRepeatingConstIteratorWithIndex** : **public** itk::ImageConstIteratorWithIndex<*TImage*>

A multi-dimensional image iterator that visits a random set of pixels within an image region. All pixels in the image will be visited before any are repeated. A priority image may be passed to the iterator which will cause it to select certain sets of pixels (those with lower priority values) before others.

This class was contributed by Rupert Brooks, McGill Centre for Intelligent Machines, Montreal, Canada. It is heavily based on the ImageRandomIterator class.

ImageRandomNonRepeatingConstIteratorWithIndex is a multi-dimensional iterator class that is templated over image type. ImageRandomNonRepeatingConstIteratorWithIndex is constrained to walk only within the specified region. When first instantiated, it creates (and stores) a random permutation of the image pixels. It then visits each pixel in the order specified by the permutation. Thus, iterator++ followed by iterator will end up leaving the iterator pointing at the same pixel. Furthermore, iterating from beginning to end will cover each pixel in the region exactly once.

This iterator can be passed an image the same size as the region, which specifies a priority for the pixels. Within areas of this priority image that have the same value, the pixel selection will be random. Otherwise the pixel selection will be in the order of the priority image. In the extreme, this allows the order of the pixel selection to be completely specified.

ImageRandomNonRepeatingConstIteratorWithIndex assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**Author** Rupert Brooks, McGill Centre for Intelligent Machines. Canada

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomNonRepeatingConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Random Select Pixel From Region Without Replacing
- Permute Sequence Of Indices

Subclassed by itk::ImageRandomNonRepeatingIteratorWithIndex< TImage >

See itk::ImageRandomNonRepeatingConstIteratorWithIndex for additional documentation.

### Pi Constant

### Synopsis

This will print a float that represents the constant Pi.

### Results

Output:

```
3.14159
```

### Code

### C++

```
#include "itkMath.h"

int
main(int /*argc*/, char * /*argv*/[])
{
  std::cout << itk::Math::pi << std::endl;
  return EXIT_SUCCESS;
}
```

### Produce Image Programmatically

### Synopsis

Produce an image programmitically.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#ifndef __itkImageSource_hxx
#define __itkImageSource_hxx

#include "ImageSource.h"
#include "itkObjectFactory.h"
#include "itkImageRegionIterator.h"
#include "itkImageRegionConstIterator.h"
```

(continues on next page)

(continued from previous page)

```
namespace itk
{

template <class TImage>
void
ImageFilter<TImage>::GenerateData()
{
  double a = 2.1;
  itkExceptionMacro("Here is a variable: " << a << " and then more text.");
}

} // namespace itk


#endif
```

## Classes demonstrated

template<typename **TOutputImage**>
**class ImageSource** : **public** itk::*ProcessObject*, **private** itk::ImageSourceCommon

Base class for all process objects that output image data.

ImageSource is the base class for all process objects that output image data. Specifically, this class defines the GetOutput() method that returns a pointer to the output image. The class also defines some internal private data members that are used to manage streaming of data.

Memory management in an ImageSource is slightly different than a standard ProcessObject. ProcessObject's always release the bulk data associated with their output prior to GenerateData() being called. ImageSources default to not releasing the bulk data incase that particular memory block is large enough to hold the new output values. This avoids unnecessary deallocation/allocation sequences. ImageSource's can be forced to use a memory management model similar to the default ProcessObject behaviour by calling ProcessObject::ReleaseDataBeforeUpdateFlagOn(). A user may want to set this flag to limit peak memory usage during a pipeline update.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Produce Image Programmatically

Subclassed by itk::ImageToImageFilter< TDisplacementField, TOutputImage >, itk::ImageToImageFilter< TFeatureImage, TOutputImage >, itk::ImageToImageFilter< TInputImage1, TOutputImage >, itk::ImageToImageFilter< TLabelImage, TOutputImage >, itk::ImageToImageFilter< TLabelMap, TOutputImage >, itk::GenerateImageSource< TOutputImage >, itk::ImageFileReader< TOutputImage, ConvertPixelTraits >, itk::ImageSeriesReader< TOutputImage >, itk::ImageToImageFilter< TInputImage, TOutputImage >, itk::LandmarkDisplacementFieldSource< TOutputImage >, itk::PathToImageFilter< TInputPath, TOutputImage >, itk::PointSetToImageFilter< TInputPointSet, TOutputImage >, itk::RandomImageSource< TOutputImage >, itk::SpatialObjectToImageFilter< TInputSpatialObject, TOutputImage >, itk::Testing::ExtractSliceImageFilter< TInputImage, TOutputImage >, itk::Testing::StretchIntensityImageFilter< TInputImage, TOutputImage >, itk::TransformToDisplacementFieldFilter< TOutputImage, TParametersValueType >, itk::TriangleMeshToBinaryImageFilter< TInputMesh, TOutputImage >, itk::VTKImageImport< TOutputImage >

See itk::ImageSource for additional documentation.

### Random Selection of Pixels From Region

### Synopsis

Randomly select pixels from a region of an image.

### Results

Output:

```
[4, 2]
[1, 1]
[1, 3]
[4, 0]
[3, 0]
[4, 3]
[1, 0]
[1, 1]
[4, 2]
[4, 2]
[1, 2]
[2, 1]
[1, 2]
[2, 3]
[2, 3]
[4, 1]
[4, 2]
[3, 1]
[2, 2]
[2, 2]
[1, 1]
[2, 2]
[1, 1]
[1, 1]
[1, 0]
[3, 1]
[2, 1]
[4, 0]
[2, 2]
[4, 1]
[0, 2]
[2, 1]
[0, 2]
[1, 0]
[1, 0]
[2, 3]
[1, 1]
[1, 3]
[0, 1]
[3, 0]
[4, 2]
[4, 0]
[3, 1]
```

(continues on next page)

(continued from previous page)

```
[0, 1]
[0, 3]
[0, 0]
[1, 2]
[3, 2]
[2, 0]
[2, 2]
[1, 1]
[3, 2]
[1, 1]
[2, 1]
[2, 1]
[4, 2]
[1, 3]
[4, 1]
[0, 3]
[1, 2]
[4, 3]
[0, 0]
[2, 1]
[1, 0]
[4, 3]
[4, 3]
[2, 3]
[2, 3]
[0, 3]
[1, 2]
[1, 3]
[2, 0]
[1, 0]
[2, 3]
[3, 3]
[3, 2]
[0, 0]
[3, 3]
[1, 2]
[1, 0]
[3, 0]
[3, 3]
[4, 0]
[1, 2]
[0, 2]
[1, 3]
[3, 3]
[1, 1]
[0, 3]
[0, 0]
[0, 2]
[0, 2]
[2, 2]
[2, 2]
[4, 1]
[2, 2]
[1, 1]
[1, 0]
[4, 1]
[1, 0]
```

(continues on next page)

(continued from previous page)

```
[3, 1]
[4, 3]
[0, 0]
[3, 2]
[1, 2]
[1, 2]
[2, 3]
[4, 2]
[2, 2]
[3, 0]
[0, 3]
[2, 2]
[1, 1]
[2, 3]
[4, 0]
[0, 3]
[4, 2]
[3, 2]
[3, 3]
[1, 2]
[4, 3]
[0, 0]
[1, 3]
[0, 0]
[4, 2]
[0, 2]
[0, 2]
[0, 3]
[4, 2]
[4, 2]
[4, 2]
[3, 2]
[4, 0]
[0, 0]
[0, 1]
[3, 0]
[0, 0]
[0, 0]
[0, 0]
[0, 0]
[4, 2]
[1, 2]
[0, 3]
[3, 3]
[4, 2]
[2, 2]
[2, 2]
[3, 3]
[1, 2]
[2, 1]
[0, 2]
[1, 2]
[4, 0]
[4, 3]
[3, 3]
[1, 2]
[0, 1]
```

(continues on next page)

(continued from previous page)

```
[3, 2]
[0, 2]
[4, 1]
[1, 2]
[3, 0]
[4, 3]
[1, 2]
[2, 2]
[2, 3]
[0, 0]
[2, 3]
[4, 1]
[4, 2]
[4, 2]
[1, 2]
[3, 3]
[1, 3]
[0, 2]
[3, 2]
[3, 0]
[2, 3]
[4, 2]
[2, 3]
[4, 2]
[4, 0]
[2, 0]
[1, 0]
[1, 2]
[1, 0]
[3, 2]
[0, 0]
[3, 1]
[4, 3]
[0, 3]
[0, 2]
[2, 0]
[4, 2]
[3, 2]
[1, 0]
[1, 0]
[4, 0]
[1, 2]
[0, 2]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRandomConstIteratorWithIndex.h"

int
main(int, char *[])
```

(continues on next page)

(continued from previous page)

```
{
  using ImageType = itk::Image<unsigned char, 2>;
  ImageType::Pointer image = ImageType::New();

  ImageType::SizeType regionSize;
  regionSize[0] = 5;
  regionSize[1] = 4;

  ImageType::IndexType regionIndex;
  regionIndex[0] = 0;
  regionIndex[1] = 0;

  ImageType::RegionType region;
  region.SetSize(regionSize);
  region.SetIndex(regionIndex);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRandomConstIteratorWithIndex<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  imageIterator.SetNumberOfSamples(200);
  imageIterator.GoToBegin();

  while (!imageIterator.IsAtEnd())
  {
    std::cout << imageIterator.GetIndex() << std::endl;

    ++imageIterator;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TImage**>
**class ImageRandomConstIteratorWithIndex** : **public** itk::ImageConstIteratorWithIndex<*TImage*>

A multi-dimensional image iterator that visits a random set of pixels within an image region.

ImageRandomConstIteratorWithIndex is a multi-dimensional iterator class that is templated over image type. ImageRandomConstIteratorWithIndex is constrained to walk only within the specified region. It samples random pixel positions at each increment or decrement.

ImageRandomConstIteratorWithIndex assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

The operator++ method provides a simple syntax for walking around a region of a multidimensional image. operator++ performs a jump to a random position within the specified image region. This is designed to facilitate the extraction of random samples from the image.

This is the typical use of this iterator in a loop:

```
ImageRandomConstIteratorWithIndex<ImageType> it( image, image->
↪GetRequestedRegion() );

it.SetNumberOfSamples(200);
it.GoToBegin();
while( !it.IsAtEnd() )
{
  it.Get();
  ++it;  // here it jumps to another random position inside the region
 }
```

or

```
ImageRandomConstIteratorWithIndex<ImageType> it( image, image->
↪GetRequestedRegion() );

it.SetNumberOfSamples(200);
it.GoToEnd();
while( !it.IsAtBegin() )
{
  it.Get();
  --it;  // here it jumps to another random position inside the region
 }
```

**Warning** Incrementing the iterator (++it) followed by a decrement (it) or vice versa does not in general return the iterator to the same position.

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Random Selection Of Pixels From Region

Subclassed by itk::ImageRandomIteratorWithIndex< TImage >

See itk::ImageRandomConstIteratorWithIndex for additional documentation.

### Random Select Pixel From Region Without Replacing

### Synopsis

Randomly select pixels from a region of an image without replacement.

### Results

Output:

```
[1, 2]
[1, 1]
[0, 2]
[2, 2]
[2, 1]
[2, 0]
[0, 0]
[0, 1]
[1, 0]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageRandomNonRepeatingConstIteratorWithIndex.h"

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned char, 2>;
  ImageType::Pointer image = ImageType::New();

  ImageType::SizeType regionSize;
  regionSize.Fill(3);

  ImageType::IndexType regionIndex;
  regionIndex.Fill(0);

  ImageType::RegionType region(regionIndex, regionSize);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRandomNonRepeatingConstIteratorWithIndex<ImageType> imageIterator(image,␣
↪image->GetLargestPossibleRegion());
  imageIterator.SetNumberOfSamples(region.GetNumberOfPixels());

  imageIterator.GoToBegin();
  while (!imageIterator.IsAtEnd())
  {
    std::cout << imageIterator.GetIndex() << std::endl;

    ++imageIterator;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageRandomNonRepeatingConstIteratorWithIndex** : **public** itk::ImageConstIteratorWithIndex<*TImage*>

A multi-dimensional image iterator that visits a random set of pixels within an image region. All pixels in the image will be visited before any are repeated. A priority image may be passed to the iterator which will cause it to select certain sets of pixels (those with lower priority values) before others.

This class was contributed by Rupert Brooks, McGill Centre for Intelligent Machines, Montreal, Canada. It is heavily based on the ImageRandomIterator class.

ImageRandomNonRepeatingConstIteratorWithIndex is a multi-dimensional iterator class that is templated over image type. ImageRandomNonRepeatingConstIteratorWithIndex is constrained to walk only within the specified region. When first instantiated, it creates (and stores) a random permutation of the image pixels. It then visits each pixel in the order specified by the permutation. Thus, iterator++ followed by iterator will end up leaving the iterator pointing at the same pixel. Furthermore, iterating from beginning to end will cover each pixel in the region exactly once.

This iterator can be passed an image the same size as the region, which specifies a priority for the pixels. Within areas of this priority image that have the same value, the pixel selection will be random. Otherwise the pixel selection will be in the order of the priority image. In the extreme, this allows the order of the pixel selection to be completely specified.

ImageRandomNonRepeatingConstIteratorWithIndex assumes a particular layout of the image data. The is arranged in a 1D array as if it were [][][][slice][row][col] with Index[0] = col, Index[1] = row, Index[2] = slice, etc.

**MORE INFORMATION** For a complete description of the ITK Image Iterators and their API, please see the Iterators chapter in the ITK Software Guide. The ITK Software Guide is available in print and as a free .pdf download from https://www.itk.org.

**Author** Rupert Brooks, McGill Centre for Intelligent Machines. Canada

**See** ImageConstIterator

**See** ConditionalConstIterator

**See** ConstNeighborhoodIterator

**See** ConstShapedNeighborhoodIterator

**See** ConstSliceIterator

**See** CorrespondenceDataStructureIterator

**See** FloodFilledFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalConstIterator

**See** FloodFilledImageFunctionConditionalIterator

**See** FloodFilledSpatialFunctionConditionalConstIterator

**See** FloodFilledSpatialFunctionConditionalIterator

**See** ImageConstIterator

**See** ImageConstIteratorWithIndex

**See** ImageIterator

**See** ImageIteratorWithIndex

**See** ImageLinearConstIteratorWithIndex

**See** ImageLinearIteratorWithIndex

**See** ImageRandomNonRepeatingConstIteratorWithIndex

**See** ImageRandomIteratorWithIndex

**See** ImageRegionConstIterator

**See** ImageRegionConstIteratorWithIndex

**See** ImageRegionExclusionConstIteratorWithIndex

**See** ImageRegionExclusionIteratorWithIndex

**See** ImageRegionIterator

**See** ImageRegionIteratorWithIndex

**See** ImageRegionReverseConstIterator

**See** ImageRegionReverseIterator

**See** ImageReverseConstIterator

**See** ImageReverseIterator

**See** ImageSliceConstIteratorWithIndex

**See** ImageSliceIteratorWithIndex

**See** NeighborhoodIterator

**See** PathConstIterator

**See** PathIterator

**See** ShapedNeighborhoodIterator

**See** SliceIterator

**See** ImageConstIteratorWithIndex

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Random Select Pixel From Region Without Replacing
- Permute Sequence Of Indices

Subclassed by itk::ImageRandomNonRepeatingIteratorWithIndex< TImage >

See itk::ImageRandomNonRepeatingConstIteratorWithIndex for additional documentation.

### Read a PointSet

### Synopsis

Read a PointSet

### Results

Print all points

### Code

### C++

```
#include "itkVTKPolyDataReader.h"

#include "itkImage.h"
#include "itkPointSet.h"

#include <string>
#include <iostream>

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cout << "Usage: " << argv[0] << " Input(.vtk)" << std::endl;
    return EXIT_FAILURE;
  }

  std::string InputFilename = argv[1];
  std::cout << "Input file: " << InputFilename << std::endl;

  // using PointSetType = itk::PointSet<double, 3 >;
  // PointSetType::Pointer pointsSet = PointSetType::New();
  // using PointType = PointSetType::PointType;

  constexpr unsigned int Dimension = 3;

  using CoordType = float;
  using MeshType = itk::Mesh<CoordType, Dimension>;

  using ReaderType = itk::VTKPolyDataReader<MeshType>;
  ReaderType::Pointer polyDataReader = ReaderType::New();

  using PointType = ReaderType::PointType;
  using VectorType = ReaderType::VectorType;

  polyDataReader->SetFileName(InputFilename.c_str());

  try
  {
    polyDataReader->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  std::cout << "polyDataReader:" << std::endl;
  std::cout << polyDataReader << std::endl;

  MeshType::Pointer mesh = polyDataReader->GetOutput();

  PointType point;

  std::cout << "Testing itk::VTKPolyDataReader" << std::endl;

  unsigned int numberOfPoints = mesh->GetNumberOfPoints();
  unsigned int numberOfCells = mesh->GetNumberOfCells();

  std::cout << "numberOfPoints= " << numberOfPoints << std::endl;
  std::cout << "numberOfCells= " << numberOfCells << std::endl;

  if (!numberOfPoints)
  {
    std::cerr << "ERROR: numberOfPoints= " << numberOfPoints << std::endl;
    return EXIT_FAILURE;
  }

  if (!numberOfCells)
  {
    std::cerr << "ERROR: numberOfCells= " << numberOfCells << std::endl;
    return EXIT_FAILURE;
  }

  // Retrieve points
  for (unsigned int i = 0; i < numberOfPoints; i++)
  {
    PointType pp;
    bool      pointExists = mesh->GetPoint(i, &pp);

    if (pointExists)
    {
      std::cout << "Point is = " << pp << std::endl;
    }
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class PointSet** : **public** itk::DataObject

A superclass of the N-dimensional mesh structure; supports point (geometric coordinate and attribute) definition.

PointSet is a superclass of the N-dimensional mesh structure (itk::Mesh). It provides the portion of the mesh definition for geometric coordinates (and associated attribute or pixel information). The defined API provides operations on points but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier to access the points. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

PointSet has two template parameters. The first is the pixel type, or the type of data stored (optionally) with the points. The second is the "MeshTraits" structure controlling type information characterizing the point set. Most

users will be happy with the defaults, and will not have to worry about this second argument.

Template parameters for PointSet:

TPixelType = The type stored as data for the point.

TMeshTraits = Type information structure for the point set.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a PointSet
- Read a PointSet
- Write a PointSet
- Bounding Box Of A Point Set

Subclassed by itk::Mesh< TPixelType, VDimension, TMeshTraits >

See itk::PointSet for additional documentation.

### Read Write Vector Image

### Synopsis

This example illustrates how to read and write an image of pixel type Vector.

### Results



Fig. 46: Input image



Fig. 47: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Read Write Vector Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()
```

(continues on next page)

(continued from previous page)

```
VectorDimension = 4

PixelType = itk.Vector[itk.F, VectorDimension]

ImageDimension = 3

ImageType = itk.Image[PixelType, ImageDimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(reader.GetOutput())
writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"

int
main(int argc, char * argv[])
{
  // Verify the number of parameters in the command line
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile  outputImageFile " << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int VectorDimension = 4;

  using PixelType = itk::Vector<float, VectorDimension>;

  constexpr unsigned int ImageDimension = 3;

  using ImageType = itk::Image<PixelType, ImageDimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(reader->GetOutput());

  try
  {
    writer->Update();
```

(continues on next page)

(continued from previous page)

```
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, . . .

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function

- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Re-Run Pipeline With Changing Largest Possible Region

### Synopsis

Re-run a pipeline where the LargestPossibleRegion in the pipeline is expected to change on consecutive runs. The pipeline does not detect this condition, and it will throw an exception,

```
Requested region is (at least partially) outside the largest possible region.
```

In this case, an *UpdateLargestPossibleRegion()* call is required instead of *Update()*.

### Results

Output:

```
Initial LargestPossibleRegion: ImageRegion (0x7fff11257330)
Dimension: 3
Index: [0, 0, 0]
Size: [256, 256, 3]


Trying to call Update() after shrinking the LargestPossibleRegion...

Error:
itk::InvalidRequestedRegionError (0x28cd880)
Location: "unknown"
File: /home/matt/bin/ITKSphinxExamples-Clang-RelWithDebInfo/ITK/Modules/Core/Common/
↪src/itkDataObject.cxx
Line: 411
Description: Requested region is (at least partially) outside the largest possible␣
↪region.


Trying to call UpdateLargestPossibleRegion() after shrinking the␣
↪LargestPossibleRegion...

Shrunk LargestPossibleRegion: ImageRegion (0x7fff11257330)
  Dimension: 3
```

(continues on next page)

(continued from previous page)

```
  Index: [0, 0, 0]
  Size: [256, 256, 2]
```

### Code

### C++

```
#include "itkImageSeriesReader.h"
#include "itkNumericSeriesFileNames.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <seriesFormat> <startIndex> <endIndex>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * seriesFormat = argv[1];
  unsigned int startIndex = std::stoi(argv[2]);
  unsigned int endIndex = std::stoi(argv[3]);

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using NameGeneratorType = itk::NumericSeriesFileNames;
  NameGeneratorType::Pointer nameGenerator = NameGeneratorType::New();
  nameGenerator->SetSeriesFormat(seriesFormat);
  nameGenerator->SetStartIndex(startIndex);
  nameGenerator->SetEndIndex(endIndex);
  std::vector<std::string> fileNames = nameGenerator->GetFileNames();

  using ReaderType = itk::ImageSeriesReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileNames(fileNames);

  try
  {
    reader->Update();

    ImageType::ConstPointer     output = reader->GetOutput();
    const ImageType::RegionType largestRegion = output->GetLargestPossibleRegion();
    std::cout << "Initial LargestPossibleRegion: ";
    std::cout << largestRegion << std::endl;
    ;
  }
  catch (itk::ExceptionObject & error)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  fileNames.pop_back();

  try
  {
    std::cout << "\nTrying to call Update() "
              << "after shrinking the LargestPossibleRegion...\n"
              << std::endl;
    reader->SetFileNames(fileNames);
    reader->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    // Print out the error that occurs
    std::cout << "Error: " << error << std::endl;
  }

  try
  {
    std::cout << "Trying to call UpdateLargestPossibleRegion() "
              << "after shrinking the LargestPossibleRegion...\n"
              << std::endl;
    reader->SetFileNames(fileNames);
    reader->UpdateLargestPossibleRegion();

    ImageType::ConstPointer     output = reader->GetOutput();
    const ImageType::RegionType largestRegion = output->GetLargestPossibleRegion();
    std::cout << "Shrunk LargestPossibleRegion: ";
    std::cout << largestRegion << std::endl;
    ;
  }
  catch (itk::ExceptionObject & error)
  {
    std::cout << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class ProcessObject** : **public** itk::*Object*

The base class for all process objects (source, filters, mappers) in the Insight data processing pipeline.

ProcessObject is an abstract object that specifies behavior and interface of network process objects (sources, filters, mappers). Source objects are creators of visualization data; filters input, process, and output image data; and mappers transform data into another form (like transforming coordinates or writing data to a file).

A major role of ProcessObject is to define the inputs and outputs of a filter. More than one input and/or output may exist for a given filter. Some classes (e.g., source objects or mapper objects) will not use inputs (the source) or outputs (mappers). In this case, the inputs or outputs is just ignored.

VOCABULARY:

- named entry - an entry indexed by a DataObjectIdentifierType or string.
- index entry - an entry indexed by an integer, which also always has a string identifier.
- define an input/output - adds a named entry or a indexed entry.
- required input - a precondition that the inputs is set before updating.
- set the value - set the value of an input or output, and automatically define the entry if it does not exist.

The inputs and outputs are referenced by name and optionally by an integer index. The **Primary** input and the **Primary** output play a special role: they drive the pipeline.

**Note**

- The Primary Input is always defined internally, and is handled as a special case for many methods.
- Some inputs can be defined as required. Either explicitly by name or the older ITKv3 style where a certain number of index inputs are required.

In addition to the reference by name, it is possible to access the inputs and outputs with an index. The index by default is mapped internally to a name with '_' followed by the index number. This default name can be changed with the AddRequiredInputName method. The indexed input or output 0 is mapped to the Primary input or output. The name of the Primary input or output defaults to "Primary", but this can be changed with SetPrimaryInputName and SetPrimaryOutputName.

For complicated filters which have optional, or varied required inputs, named input access is preferred. However, indexed input access provides constant time access to input and output DataObjects, and so are more efficient. A name can also be associated with an indexed input. Neither type of input or output should be accessed in a tight loop.

ProcessObject invokes the following events: Command::StartEvent, Command::EndEvent These are convenience events you can use for any purpose (e.g., debugging info, highlighting/notifying user interface, etc.) See Command and LightObject for information on using AddObserver.

Another event Command::ProgressEvent can be observed. Some filters invoke this event periodically during their execution (with the progress, parameter, the fraction of work done). The use is similar to that of StartEvent and EndEvent. Filters may also check their AbortGenerateData flag to determine whether to prematurely end their execution.

An important feature of subclasses of ProcessObject is that it is possible to control the memory-management model (i.e., retain output versus delete output data). The ReleaseDataFlag enables the deletion of the output data once the downstream process object finishes processing the data (please see text). The ReleaseDataBeforeUpdateFlag enables the deletion of the ProcessObject's output data from a previous update if that output data is slated to be regenerated by the pipeline process. Setting this flag can control peak memory usage during a subsequent pipeline update. For a ProcessObject, the ReleaseDataFlag defaults to false and the ReleaseDataBeforeUpdateFlag defaults to true. Some subclasses of ProcessObject, for example ImageSource, use a default setting of false for the ReleaseDataBeforeUpdateFlag.

Subclasses of ProcessObject may override 4 of the methods of this class to control how a given filter may interact with the pipeline (dataflow). These methods are: GenerateOutputInformation(), EnlargeOutputRequestedRegion(), GenerateInputRequestedRegion(), and GenerateOutputRequestedRegion(). By overriding these methods, a filter can deviate from the base assumptions of the pipeline execution model.

Subclassed by itk::ImageSource< Functor::MakeJoin< TInputImage1, TInputImage2 >::ImageType >, itk::ImageSource< Image< CovariantVector< TDataType, TInputImage::ImageDimension >, TInputImage::ImageDimension > >, itk::ImageSource< Image< DiffusionTensor3D< TTensorPixelType >, 3 > >, itk::ImageSource< Image< IdentifierType, TInputImage::ImageDimension > >, itk::ImageSource< Image< TLabelsType, TInputVectorImage::ImageDimension > >, itk::ImageSource< Image< TOutputPixelType, 2 > >, itk::ImageSource< Image< TOutputPixelType, TInputImage::ImageDimension > >,

itk::ImageSource< Image< TPixel, 3 > >, itk::ImageSource< Image< TPixel, VImageDimension > >, itk::ImageSource< ImageType >, itk::ImageSource< TClassifiedImage >, itk::ImageSource< TDisplacementField >, itk::ImageSource< TEigenValueImage >, itk::ImageSource< TImage >, itk::ImageSource< TImageType >, itk::ImageSource< TInputImage >, itk::ImageSource< TInputImage1 >, itk::ImageSource< TIntensityImage >, itk::ImageSource< TLabelImage >, itk::ImageSource< TLevelSet >, itk::ImageSource< TOutputImageType >, itk::ImageSource< TSparseOutputImage >, itk::ImageSource< TSparseOutputImageType >, itk::ImageSource< VectorImage< TProbabilityPrecisionType, TInputImage::ImageDimension > >, itk::DCMTKSeriesFileNames, itk::GDCMSeriesFileNames, itk::HistogramThresholdCalculator< THistogram, TOutput >, itk::ImageFileWriter< TInputImage >, itk::ImageRegistrationMethod< TFixedImage, TMovingImage >, itk::ImageRegistrationMethodv4< TFixedImage, TMovingImage, TOutputTransform, TVirtualImage, TPointSet >, itk::ImageSeriesWriter< TInputImage, TOutputImage >, itk::ImageSource< TOutputImage >, itk::ImageToSpatialObjectRegistrationMethod< TFixedImage, TMovingSpatialObject >, itk::ImageToVTKImageFilter< TInputImage >, itk::MeshFileWriter< TInputMesh >, itk::MeshSource< TOutputMesh >, itk::MultiResolutionImageRegistrationMethod< TFixedImage, TMovingImage >, itk::PathSource< TOutputPath >, itk::PointSetToImageRegistrationMethod< TFixedPointSet, TMovingImage >, itk::PointSetToPointSetRegistrationMethod< TFixedPointSet, TMovingPointSet >, itk::PointSetToSpatialObjectDemonsRegistration< TFixedPointSet, TMovingSpatialObject >, itk::Statistics::CovarianceSampleFilter< TSample >, itk::Statistics::HistogramToRunLengthFeaturesFilter< THistogram >, itk::Statistics::HistogramToTextureFeaturesFilter< THistogram >, itk::Statistics::ImageToListSampleFilter< TImage, TMaskImage >, itk::Statistics::MeanSampleFilter< TSample >, itk::Statistics::SampleClassifierFilter< TSample >, itk::Statistics::SampleToHistogramFilter< TSample, THistogram >, itk::Statistics::SampleToSubsampleFilter< TSample >, itk::Statistics::ScalarImageToCooccurrenceListSampleFilter< TImage >, itk::Statistics::ScalarImageToCooccurrenceMatrixFilter< TImageType, THistogramFrequencyContainer >, itk::Statistics::ScalarImageToRunLengthFeaturesFilter< TImageType, THistogramFrequencyContainer >, itk::Statistics::ScalarImageToRunLengthMatrixFilter< TImageType, THistogramFrequencyContainer >, itk::Statistics::ScalarImageToTextureFeaturesFilter< TImageType, THistogramFrequencyContainer >, itk::Statistics::StandardDeviationPerComponentSampleFilter< TSample >, itk::StreamingProcessObject, itk::TemporalProcessObject, itk::VTKImageExportBase, itk::watershed::BoundaryResolver< TPixelType, TDimension >, itk::watershed::EquivalenceRelabeler< TScalar, TImageDimension >, itk::watershed::Relabeler< TScalar, TImageDimension >, itk::watershed::Segmenter< TInputImage >, itk::watershed::SegmentTreeGenerator< TScalar >, itk::MeshSource< TDisplacements >, itk::MeshSource< TFeatures >, itk::MeshSource< TOutput >, itk::MeshSource< TOutputPointSet >, itk::MeshSource< VoronoiDiagram2D< TCoordType > >, itk::PathSource< OrthogonallyCorrected2DParametricPath >, itk::PathSource< PolyLineParametricPath< 2 > >, itk::PathSource< TOutputChainCodePath >, itk::PathSource< TOutputFourierSeriesPath >, itk::watershed::Relabeler< ScalarType, Self::ImageDimension >, itk::watershed::Segmenter< InputImageType >, itk::watershed::SegmentTreeGenerator< ScalarType >

See itk::ProcessObject for additional documentation.

template<typename **TOutputImage**>
**class ImageSeriesReader** : **public** itk::*ImageSource<TOutputImage>*

Data source that reads image data from a series of disk files.

This class builds an n-dimension image from multiple n-1 dimension image files. The files stored in a vector of strings are read using the ImageFileReader. File format may vary between the files, but the image data must have the same Size for all dimensions.

**See** GDCMSeriesFileNames

**See** NumericSeriesFileNames

See itk::ImageSeriesReader for additional documentation.

### Return Object From Function

### Synopsis

Return an object from a function.

### Results

Output:

```
ImageRegion (0x7ff7f950ba70)
Dimension: 2
Index: [0, 0]
Size: [10, 10]
```

### Code

### C++

```
#include "itkImage.h"

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
}

ImageType::Pointer
ReturnSmartPointer()
{
  ImageType::Pointer  image = ImageType::New();
  itk::Index<2>       corner = { { 0, 0 } };
  itk::Size<2>        size = { { 10, 10 } };
  itk::ImageRegion<2> region(corner, size);
  image->SetRegions(region);
  image->Allocate();

  return image;
}

ImageType *
ReturnPointer()
{
  ImageType::Pointer  image = ImageType::New();
  itk::Index<2>       corner = { { 0, 0 } };
  itk::Size<2>        size = { { 10, 10 } };
  itk::ImageRegion<2> region(corner, size);
  image->SetRegions(region);
  image->Allocate();

  return image;
}

int
```

(continues on next page)

(continued from previous page)

```
main(int, char *[])
{
  {
    // This is how it should be done.
    ImageType::Pointer smartPointer = ReturnSmartPointer();
    std::cout << smartPointer->GetLargestPossibleRegion() << std::endl;
  }

  {
    ImageType * pointer = ReturnPointer();
    // This crashes the program because the smart pointer created in the function␣
↪goes out of scope and gets deleted
    // because it is returned as a normal pointer.
    // std::cout << pointer->GetLargestPossibleRegion() << std::endl;
    pointer = nullptr; // Here to silence warning
  }

  {
    ImageType * pointer = ReturnSmartPointer();
    // This crashes the program because though the function returned a ::Pointer, it␣
↪was not stored
    // anywhere so the reference count was not increased, so it got deleted.
    // std::cout << pointer->GetLargestPossibleRegion() << std::endl;
    pointer = nullptr; // Here to silence warning
  }

  {
    // I thought this might also work, but it does not (crash).
    // My reasoning was that even though you don't return a smart pointer, you assign␣
↪the object to a smart
    // pointer at return time, so it still has a reference count of 1.
    // ImageType::Pointer smartPointer = ReturnPointer(); // this line causes a
↪'glibc error'
    // std::cout << smartPointer->GetLargestPossibleRegion() << std::endl;
  }

  return 0;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Set the Default Number of Threads

### Synopsis

Set the default number of threads for multi-threading.

The default number of threads can also be set via the environment variable, *ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS*.

### Results

```
Filter's default number of threads: 3
```

### Code

#### C++

```
#include "itkMultiThreaderBase.h"
#include "itkMedianImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <NumberOfThreads>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const auto numberOfThreads = static_cast<unsigned int>(std::atoi(argv[1]));

  itk::MultiThreaderBase::SetGlobalDefaultNumberOfThreads(numberOfThreads);

  constexpr unsigned int Dimension = 2;
  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;
  using FilterType = itk::MedianImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();

  const auto filterDefaultThreads = filter->GetMultiThreader()->
↪GetGlobalDefaultNumberOfThreads();
  std::cout << "Filter's default number of threads: " << filterDefaultThreads << std::
↪endl;

  if (filterDefaultThreads != numberOfThreads)
  {
    std::cerr << "Filter does not have expected default number of threads." << std::
↪endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python
import itk

parser = argparse.ArgumentParser(description="Set the Default Number Of Threads.")
parser.add_argument("number_of_threads", type=int)
args = parser.parse_args()

# Create a MultiThreaderBase instance to get access to its static methods
threader = itk.MultiThreaderBase.New()
threader.SetGlobalDefaultNumberOfThreads(args.number_of_threads)

filt = itk.MedianImageFilter.New()
filter_default_threads = filt.GetMultiThreader().GetGlobalDefaultNumberOfThreads()

print("Filter's default number of threads: {}".format(filter_default_threads))

assert filter_default_threads == args.number_of_threads
```

### Classes demonstrated

**class MultiThreaderBase** : **public** itk::*Object*
A class for performing multithreaded execution.

Multithreaders are a class hierarchy that provides support for multithreaded execution by abstracting away platform-specific details. This class can be used to execute a single method on multiple threads or to parallelize an operation over a given image region or array.

Subclassed by itk::PoolMultiThreader, itk::TBBMultiThreader

See itk::MultiThreaderBase for additional documentation.

### Set Pixel Value in One Image

### Synopsis

This example demonstrates how to set the value of individual pixels in an image.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
```

(continues on next page)

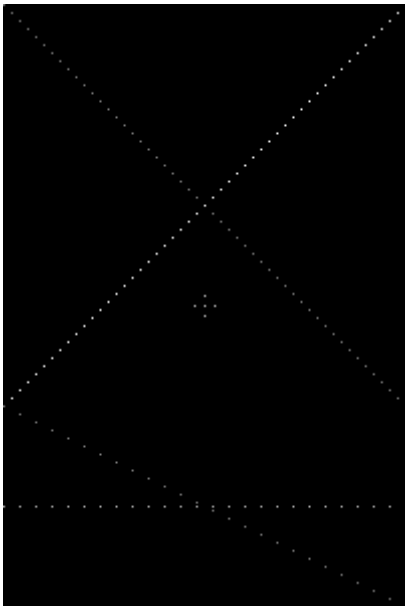


Fig. 48: Output image

(continued from previous page)

```
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <OutputFileName>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::RegionType region;

  ImageType::IndexType start;
  start.Fill(0);

  region.SetIndex(start);

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);

  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(itk::NumericTraits<PixelType>::Zero);

  ImageType::IndexType pixelIndex;
```

(continues on next page)

(continued from previous page)

```
  for (ImageType::IndexValueType r = 0; r < 50; r++)
  {
    pixelIndex[0] = 4 * r;
    pixelIndex[1] = 4 * r;

    image->SetPixel(pixelIndex, 128);

    pixelIndex[0] = 4 * r;
    pixelIndex[1] = 200 - 4 * r;

    image->SetPixel(pixelIndex, 255);
  }

  for (ImageType::IndexValueType r = 0; r < 25; r++)
  {
    pixelIndex[0] = 8 * r;
    pixelIndex[1] = 200 + 4 * r;

    image->SetPixel(pixelIndex, 128);

    pixelIndex[0] = 8 * r;
    pixelIndex[1] = 250;

    image->SetPixel(pixelIndex, 180);
  }

  pixelIndex[0] = 95;
  pixelIndex[1] = 150;

  image->SetPixel(pixelIndex, 200);

  pixelIndex[0] = 100;
  pixelIndex[1] = 150;

  image->SetPixel(pixelIndex, 200);

  pixelIndex[0] = 105;
  pixelIndex[1] = 150;

  image->SetPixel(pixelIndex, 200);

  pixelIndex[0] = 100;
  pixelIndex[1] = 155;

  image->SetPixel(pixelIndex, 200);

  pixelIndex[0] = 100;
  pixelIndex[1] = 145;

  image->SetPixel(pixelIndex, 200);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(image);
  writer->SetFileName(argv[1]);

  try
```

(continues on next page)

(continued from previous page)

```
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python
import itk
import argparse

parser = argparse.ArgumentParser(description="Set Pixel Value In One Image.")
parser.add_argument("output_image")
args = parser.parse_args()

Dimension = 2

PixelType = itk.UC

ImageType = itk.Image[PixelType, Dimension]

region = itk.ImageRegion[Dimension]()

start = itk.Index[Dimension]()
start.Fill(0)

region.SetIndex(start)

size = itk.Size[Dimension]()
size[0] = 200
size[1] = 300

region.SetSize(size)

image = ImageType.New()
image.SetRegions(region)
image.Allocate()
image.FillBuffer(itk.NumericTraits[PixelType].ZeroValue())

pixelIndex = itk.Index[Dimension]()

for r in range(50):
    pixelIndex[0] = 4 * r
    pixelIndex[1] = 4 * r

    image.SetPixel(pixelIndex, 128)
```

(continues on next page)

(continued from previous page)

```
    pixelIndex[0] = 4 * r
    pixelIndex[1] = 200 - 4 * r

    image.SetPixel(pixelIndex, 255)

for r in range(25):
    pixelIndex[0] = 8 * r
    pixelIndex[1] = 200 + 4 * r

    image.SetPixel(pixelIndex, 128)

    pixelIndex[0] = 8 * r
    pixelIndex[1] = 250

    image.SetPixel(pixelIndex, 180)

pixelIndex[0] = 95
pixelIndex[1] = 150

image.SetPixel(pixelIndex, 200)

pixelIndex[0] = 100
pixelIndex[1] = 150

image.SetPixel(pixelIndex, 200)

pixelIndex[0] = 105
pixelIndex[1] = 150

image.SetPixel(pixelIndex, 200)

pixelIndex[0] = 100
pixelIndex[1] = 155

image.SetPixel(pixelIndex, 200)

pixelIndex[0] = 100
pixelIndex[1] = 145

image.SetPixel(pixelIndex, 200)

itk.imwrite(image, args.output_image)
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself.

ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Sort ITK Index

### Synopsis

Sort itk::Index.

### Results

### Code

### C++

```
#include "itkIndex.h"
#include <map>

int
main(int, char *[])
{
  using IndexType = itk::Index<2>;
  IndexType                  index = { { 3, 4 } };
  std::map<IndexType, float> myMap;
  myMap[index] = 42;
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Store Non-Pixel Data In Image

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Store non-pixel associated data in an image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include <itkImage.h>
#include <itkMetaDataDictionary.h>
#include <itkMetaDataObject.h>
#include <itkImageRegionIterator.h>
#include <itkImageFileWriter.h>
#include <itkImageFileReader.h>

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  // Create an image
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  // Store some data in it
  itk::MetaDataDictionary dictionary;
  itk::EncapsulateMetaData<float>(dictionary, "ASimpleFloat", 1.2);
  image->SetMetaDataDictionary(dictionary);

  // View all of the data
  dictionary.Print(std::cout);

  // View the data individually
  auto itr = dictionary.Begin();

  while (itr != dictionary.End())
  {
    std::cout << "Key   = " << itr->first << std::endl;
    std::cout << "Value = ";
    itr->second->Print(std::cout);
    std::cout << std::endl;
    ++itr;
  }

  // Write the image (and the data) to a file
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("test.mhd");
  writer->SetInput(image);
  writer->Update();

  // Read the image (and data) from the file
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName("test.mhd");

  // Display the data
  std::cout << "Data read from file:" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  reader->GetMetaDataDictionary().Print(std::cout);

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(20);
    ++imageIterator;
  }
}
```

## Classes demonstrated

**class MetaDataDictionary**

Provides a mechanism for storing a collection of arbitrary data types.

The MetaDataDictionary, along with the MetaDataObject derived template classes, is designed to provide a mechanism for storing a collection of arbitrary data types. The main motivation for such a collection is to associate arbitrary data elements with itk DataObjects.

**Author** Hans J. Johnson

The MetaDataDictionary implements shallow copying with copy on write behavior. When a copy of this class is created, the new copy will be shared with the old copy via C++11 shared pointers. When a non-constant operation is done, if the dictionary is not unique to this object, then a deep copy is performed. This make is very cheap to create multiple copies of the same dictionary if they are never modified.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Store Non-Pixel Data In Image

See itk::MetaDataDictionary for additional documentation.

### Stream a Pipeline

### Synopsis

Stream a pipeline. The PipelineMonitorImageFilter shows how the image is processed in three sub-regions. Note that the StreamingImageFilter comes at the end of the pipeline and that no calls to *Update()* occur on any intermediate filters.

### Results

```
The output LargestPossibleRegion is: ImageRegion (0x8dc420)
  Dimension: 2
  Index: [0, 0]
  Size: [3, 3]


Updated RequestedRegions's:
  ImageRegion (0x8dce80)
  Dimension: 2
  Index: [0, 0]
  Size: [3, 1]

  ImageRegion (0x8dcea8)
  Dimension: 2
  Index: [0, 1]
  Size: [3, 1]

  ImageRegion (0x8dced0)
  Dimension: 2
  Index: [0, 2]
  Size: [3, 1]
```

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.10.0"):
    print("ITK 4.10.0 is required.")
    sys.exit(1)

parser = argparse.ArgumentParser(description="Stream A Pipeline.")
parser.add_argument("number_of_splits", type=int)
args = parser.parse_args()
```

(continues on next page)

(continued from previous page)

```
Dimension = 2
PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

source = itk.RandomImageSource[ImageType].New()
size = itk.Size[Dimension]()
size.Fill(args.number_of_splits)
source.SetSize(size)

monitorFilter = itk.PipelineMonitorImageFilter[ImageType].New()
monitorFilter.SetInput(source.GetOutput())

streamingFilter = itk.StreamingImageFilter[ImageType, ImageType].New()
streamingFilter.SetInput(monitorFilter.GetOutput())
streamingFilter.SetNumberOfStreamDivisions(args.number_of_splits)

streamingFilter.Update()

print(
    "The output LargestPossibleRegion is: "
    + str(streamingFilter.GetOutput().GetLargestPossibleRegion())
)

print("")

updatedRequestedRegions = monitorFilter.GetUpdatedRequestedRegions()
print("Updated ApplyAFilterOnlyToASpecifiedImageRegion's:")
for ii in range(len(updatedRequestedRegions)):
    print("  " + str(updatedRequestedRegions[ii]))
```

### C++

```
#include "itkPipelineMonitorImageFilter.h"
#include "itkRandomImageSource.h"
#include "itkStreamingImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << argv[0] << " <NumberOfSplits>" << std::endl;
    return EXIT_FAILURE;
  }
  int numberOfSplits = std::stoi(argv[1]);

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using SourceType = itk::RandomImageSource<ImageType>;
  SourceType::Pointer source = SourceType::New();
  ImageType::SizeType size;
  size.Fill(numberOfSplits);
```

(continues on next page)

(continued from previous page)

```
  source->SetSize(size);

  using MonitorFilterType = itk::PipelineMonitorImageFilter<ImageType>;
  MonitorFilterType::Pointer monitorFilter = MonitorFilterType::New();
  monitorFilter->SetInput(source->GetOutput());
  // If ITK was built with the Debug CMake configuration, the filter
  // automatically outputs status information to the console
  monitorFilter->DebugOn();

  using StreamingFilterType = itk::StreamingImageFilter<ImageType, ImageType>;
  StreamingFilterType::Pointer streamingFilter = StreamingFilterType::New();
  streamingFilter->SetInput(monitorFilter->GetOutput());
  streamingFilter->SetNumberOfStreamDivisions(numberOfSplits);

  try
  {
    streamingFilter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  std::cout << "The output LargestPossibleRegion is: " << streamingFilter->
↪GetOutput()->GetLargestPossibleRegion()
            << std::endl;
  std::cout << std::endl;

  const MonitorFilterType::RegionVectorType updatedRequestedRegions = monitorFilter->
↪GetUpdatedRequestedRegions();
  std::cout << "Updated RequestedRegions's:" << std::endl;
  for (const auto & updatedRequestedRegion : updatedRequestedRegions)
  {
    std::cout << "  " << updatedRequestedRegion << std::endl;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class StreamingImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Pipeline object to control data streaming for large data processing.

StreamingImageFilter is a pipeline object that allows the user to control how data is pulled through the pipeline. To generate its OutputRequestedRegion, this filter will divide the output into several pieces (controlled by SetNumberOfStreamDivisions), and call the upstream pipeline for each piece, tiling the individual outputs into one large output. This reduces the memory footprint for the application since each filter does not have to process the entire dataset at once. This filter will produce the entire output as one image, but the upstream filters will do their processing in pieces.

See itk::StreamingImageFilter for additional documentation.

template<typename **TImageType**>

**class PipelineMonitorImageFilter** : **public** itk::*ImageToImageFilter<TImageType, TImageType>*

Enables monitoring, recording and debugging of the pipeline execution and information exchange.

This filter is useful for testing, debugging, and understanding the pipeline. When DebugOn is enabled and compiled in Debug mode, many itkDebug messages are printed. This filter also features, several Verify methods which check the recorded information, for certain conditions, which should occur when well behaved filters are executed.

There are two meta verify methods that should primarily be used depending on the expected capabilities of the pipeline:

- **VerifyAllInputCanStream**
- **VerifyAllInputCanNotStream**

During the pipeline execution this filter records a variety of information to aid if verifying correct pipeline behavior:

- **NumberOfUpdate** the number of times GenerateData was executed
- **OutputRequestedRegions** an array of the output of this filter's requested region
- **InputRequestedRegions** an array of the input image's requested region after PropagateRequestedRegion
- **UpdatedBufferedRegions** an array of the input image's buffered region after upstream GenerateData
- **UpdatedRequestedRegions** an array of the input image's requested region after upstream GenerateData

The following are recorded from the input image after the input's output information is generated:

- **UpdatedOutputOrigin**
- **UpdatedOutputDirection**
- **UpdatedOutputSpacing**
- **UpdatedOutputLargestPossibleRegion**

This filter always runs in-place so it has no per-pixel overhead.

See itk::PipelineMonitorImageFilter for additional documentation.

### Throw Exception

### Synopsis

Throw an exception.

### Results

**Note:** No output is printed, this example simply displays functionality.

### Code

### C++

```
#include "itkImage.h"
#include <itkCastImageFilter.h>

// TODO: Incomplete example
int
main(int, char *[])
{
  // Setup types
  using ImageType = itk::Image<unsigned char, 2>;
  ImageType::Pointer image = ImageType::New();

  // Create and the filter
  using FilterType = itk::CastImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VImageDimension** = 2>
**class Image** : **public** itk::ImageBase<*VImageDimension*>

Templated n-dimensional image class.

Images are templated over a pixel type (modeling the dependent variables), and a dimension (number of independent variables). The container for the pixel data is the ImportImageContainer.

Within the pixel container, images are modelled as arrays, defined by a start index and a size.

The superclass of Image, ImageBase, defines the geometry of the image in terms of where the image sits in physical space, how the image is oriented in physical space, the size of a pixel, and the extent of the image itself. ImageBase provides the methods to convert between the index and physical space coordinate frames.

Pixels can be accessed directly using the SetPixel() and GetPixel() methods or can be accessed via iterators that define the region of the image they traverse.

The pixel type may be one of the native types; a Insight-defined class type such as Vector; or a user-defined type. Note that depending on the type of pixel that you use, the process objects (i.e., those filters processing data objects) may not operate on the image and/or pixel type. This becomes apparent at compile-time because operator overloading (for the pixel type) is not supported.

The data in an image is arranged in a 1D array as [][][][slice][row][col] with the column index varying most rapidly. The Index type reverses the order so that with Index[0] = col, Index[1] = row, Index[2] = slice, ...

**See** ImageBase

**See** ImageContainerInterface

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Set Pixel Value In One Image
- Get Image Size
- Sort ITK Index
- Return Object From Function
- Create Another Instance Of An Image
- Pass Image To Function
- Deep Copy Image
- Throw Exception
- Get Or Set Member Variable Of ITK Class
- Mini Pipeline
- Check If Module Is Present
- Display Image

Subclassed by itk::GPUImage< TPixel, VImageDimension >

See itk::Image for additional documentation.

### Trace Memory Between Points

### Synopsis

Trace the memory charge between the execution of two pieces of code. Note that the memory must be used for it to be counted.

### Results

Results:

```
We are measuring Memory in units of kB.

** Start **
Mean: 0
Total: 0

** After allocation **
Mean: 1056
Total: 2112

** After deallocation **
Mean: 60
Total: 180
```

### Code

### C++

```
#include "itkMemoryProbe.h"
#include <iostream>

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << " <Size of the array>" << std::endl;
    return EXIT_FAILURE;
  }

  const auto N = static_cast<unsigned int>(std::stoi(argv[1]));

  if (N == 0)
  {
    std::cerr << "Size of the array should be non null" << std::endl;
    return EXIT_FAILURE;
  }

  itk::MemoryProbe memoryProbe;

  memoryProbe.Start();
  std::cout << "We are measuring " << memoryProbe.GetType();
  std::cout << " in units of " << memoryProbe.GetUnit() << ".\n" << std::endl;
  memoryProbe.Stop();

  // Expect zeros here
  std::cout << "** Start **" << std::endl;
  std::cout << "Mean: " << memoryProbe.GetMean() << std::endl;
  std::cout << "Total: " << memoryProbe.GetTotal() << std::endl;
  std::cout << std::endl;

  memoryProbe.Start();
  auto * a = new char[N];
  // The memory must be used for it to be counted.
  for (unsigned int i = 0; i < N; ++i)
  {
    a[i] = static_cast<char>(i * i);
  }
  memoryProbe.Stop();

  std::cout << "** After allocation **" << std::endl;
  std::cout << "Mean: " << memoryProbe.GetMean() << std::endl;
  std::cout << "Total: " << memoryProbe.GetTotal() << std::endl;
  std::cout << std::endl;

  memoryProbe.Start();
  delete[] a;
  memoryProbe.Stop();

  std::cout << "** After deallocation **" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  std::cout << "Mean: " << memoryProbe.GetMean() << std::endl;
  std::cout << "Total: " << memoryProbe.GetTotal() << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class MemoryProbe** : **public** itk::ResourceProbe<OffsetValueType, double>
Computes the memory allocated between two points in code.

This class allows the user to trace the memory charge between the execution of two pieces of code. It can be started and stopped in order to evaluate the execution over multiple passes. The values of memory are taken from GetProcessMemoryInfo() for Windows, the SMAPS file for Linux and getrusage() otherwise.

See itk::MemoryProbe for additional documentation.

### Try Catch Exception

### Synopsis

Try and catch an *itk::ExceptionObject*. The exception is printed to console output. This can provide more information when a program crashes, including where the exception occurred and its description. Exceptions in ITK are usually only thrown during calls to *->Update()*, so only the *->Update()* call is added to the *try* block.

### Results

```
ExceptionObject caught !

itk::ImageFileReaderException (0x20cece0)
Location: "void itk::ImageFileReader<TOutputImage, ConvertPixelTraits>::
↪GenerateOutputInformation() [with TOutputImage = itk::Image<double, 2u>,␣
↪ConvertPixelTraits = itk::DefaultConvertPixelTraits<double>]"
File: /path/to/ITK/Modules/IO/ImageBase/include/itkImageFileReader.hxx
Line: 143
Description:  Could not create IO object for file nofile.png
The file doesn't exist.
Filename = nofile.png
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"

int
main(int, char *[])
{
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Dimension = 2;
  using PixelType = double;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName("nofile.png");

  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;

    // Since the goal of the example is to catch the exception,
    // we declare this a success.
    return EXIT_SUCCESS;
  }

  // Since the goal of the example is to catch the exception,
  // the example fails if it is not caught.
  return EXIT_FAILURE;
}
```

### Use ParallelizeImageRegion

### Synopsis

This example demonstrates how to use MultiThreaderBase::ParallelizeImageRegion to apply a non-trivial operation for all pixels in an image, in parallel. To perform a multi-threaded operation like this prior ITK 5.0 required the creation of a class.

### Results

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkWatershedImageFilter.h"
#include "itkImageRegionIterator.h"

constexpr unsigned int Dimension = 2;
using InputPixelType = unsigned char;
using InputImageType = itk::Image<InputPixelType, Dimension>;
using OutputPixelType = itk::RGBAPixel<unsigned char>;
```

(continues on next page)

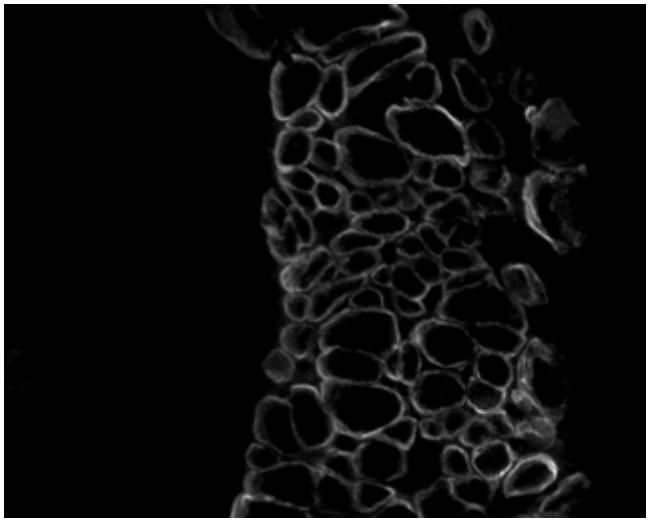


Fig. 49: Segmentation and custom colorization of an image using MultiThreaderBase::ParallelizeImageRegion.

(continued from previous page)

```
using OutputImageType = itk::Image<OutputPixelType, Dimension>;
using LabeledImageType = itk::Image<itk::IdentifierType, Dimension>;

OutputImageType::Pointer
segmentationAndCustomColorization(InputImageType::Pointer inImage)
{
  using WatershedFilterType = itk::WatershedImageFilter<InputImageType>;
  WatershedFilterType::Pointer watershed = WatershedFilterType::New();
  watershed->SetThreshold(0.05);
  watershed->SetLevel(0.3);
  watershed->SetInput(inImage);
  watershed->Update();

  LabeledImageType::Pointer image = watershed->GetOutput();
  image->DisconnectPipeline();

  OutputImageType::Pointer outImage = OutputImageType::New();
  outImage->CopyInformation(image);
  outImage->SetRegions(image->GetBufferedRegion());
  outImage->Allocate(true);

  itk::MultiThreaderBase::Pointer mt = itk::MultiThreaderBase::New();
  // ParallelizeImageRegion invokes the provided lambda function in parallel
  // each invocation will contain a piece of the overall region
  mt->ParallelizeImageRegion<Dimension>(
    image->GetBufferedRegion(),
    // the lambda will have access to outer variables 'image' and 'outImage'
    // it will have parameter 'region', which needs to be processed
    [image, outImage](const LabeledImageType::RegionType & region) {
      itk::ImageRegionConstIterator<LabeledImageType> iIt(image, region);
      itk::ImageRegionIterator<OutputImageType>       oIt(outImage, region);
      for (; !iIt.IsAtEnd(); ++iIt, ++oIt)
      {
        LabeledImageType::IndexType ind = iIt.GetIndex();
        OutputPixelType             p;
```

(continues on next page)

(continued from previous page)

```
        p.SetAlpha(iIt.Get());
        static_assert(Dimension <= 3, "Dimension has to be 2 or 3");
        for (unsigned d = 0; d < Dimension; d++)
        {
          p.SetElement(d, ind[d]);
        }
        oIt.Set(p);
      }
    },
    nullptr); // we don't have a filter whose progress needs to be updated

  return outImage;
}

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << " <OutputFileName>" << std::endl;
    return EXIT_FAILURE;
  }

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->UseCompressionOn();

  try
  {
    OutputImageType::Pointer outImage = segmentationAndCustomColorization(reader->
↪GetOutput());
    writer->SetInput(outImage);
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class MultiThreaderBase** : **public** itk::*Object*
A class for performing multithreaded execution.

Multithreaders are a class hierarchy that provides support for multithreaded execution by abstracting away platform-specific details. This class can be used to execute a single method on multiple threads or to parallelize an operation over a given image region or array.

Subclassed by itk::PoolMultiThreader, itk::TBBMultiThreader

See itk::MultiThreaderBase for additional documentation.

### Variable Length Vector

### Synopsis

This computes a variable length vector.

### Results

Output:

```
[1, 2]
1 2
VectorToVariableLengthVector()
variableLengthVector: [1, 2]
VariableLengthVectorToVector()
fixedLengthVector: [1, 2]
```

### Code

### C++

```
#include <itkVariableLengthVector.h>
#include <itkVector.h>

void
VectorToVariableLengthVector();
void
VariableLengthVectorToVector();

int
main(int, char *[])
{
  using VectorType = itk::VariableLengthVector<double>;
  VectorType v;
  v.SetSize(2);
  v[0] = 1;
  v[1] = 2;
  std::cout << v << std::endl;
```

(continues on next page)

(continued from previous page)

```
  for (unsigned int i = 0; i < v.Size(); i++)
  {
    std::cout << v[i] << " ";
  }
  std::cout << std::endl;

  VectorToVariableLengthVector();
  VariableLengthVectorToVector();
  return EXIT_SUCCESS;
}

void
VectorToVariableLengthVector()
{
  std::cout << "VectorToVariableLengthVector()" << std::endl;

  using FixedVectorType = itk::Vector<double, 2>;
  FixedVectorType fixedLengthVector;
  fixedLengthVector[0] = 1;
  fixedLengthVector[1] = 2;

  using VariableVectorType = itk::VariableLengthVector<double>;
  VariableVectorType variableLengthVector;

  // This works
  variableLengthVector.SetSize(fixedLengthVector.Size());
  variableLengthVector.SetData(fixedLengthVector.GetDataPointer());

  // This crashes with both true and false
  // variableLengthVector.SetData(fixedLengthVector.GetDataPointer(), 2, true);

  std::cout << "variableLengthVector: " << variableLengthVector << std::endl;
}

void
VariableLengthVectorToVector()
{
  std::cout << "VariableLengthVectorToVector()" << std::endl;
  using VariableVectorType = itk::VariableLengthVector<double>;
  VariableVectorType variableLengthVector;
  variableLengthVector.SetSize(2);

  variableLengthVector[0] = 1;
  variableLengthVector[1] = 2;

  using FixedVectorType = itk::Vector<double, 2>;
  FixedVectorType fixedLengthVector;

  for (unsigned int i = 0; i < FixedVectorType::GetVectorDimension(); i++)
  {
    fixedLengthVector[i] = variableLengthVector[i];
  }

  // This function doesn't exist!
  // fixedLengthVector.SetData(variableLengthVector.GetDataPointer());
  std::cout << "fixedLengthVector: " << fixedLengthVector << std::endl;
}
```

### Classes demonstrated

template<typename **TValue**>
**class VariableLengthVector**

Represents an array whose length can be defined at run-time.

This class is templated over the data type. This data-type is meant to be a scalar, such as float, double etc. . .

**Note** ITK itself provides several classes that can serve as `Arrays`.

- FixedArray - Compile time fixed length arrays that's intended to represent an enumerated collection of `n` entities.
- Array - Run time resizeable array that is intended to hold a collection of `n` entities
- Vector - Compile time fixed length array that is intended to hold a collection of `n` data types. A vector usually has a mathematical meaning. It should only be used when mathematical operations such as addition, multiplication by a scalar, product etc make sense.
- VariableLengthVector - Run time array that is intended to hold a collection of scalar data types. Again, it should be used only when mathematical operations on it are relevant. If not, use an Array.
- Point - Represents the spatial coordinates of a spatial location. Operators on Point reflect geometrical concepts.

**For the reasons listed above, you cannot instantiate**

```
VariableLengthVector< bool >
```

.

Design Considerations: We do not derive from `vnl_vector` to avoid being limited by the explicit template instantiations of vnl_vector and other hacks that vnl folks have been forced to use.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**See** CovariantVector

**See** SymmetricSecondRankTensor

**See** RGBPixel

**See** DiffusionTensor3D

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Variable Length Vector

**Invariant** If `m_LetArrayManageMemory` is true, `m_Data` is deletable (whether it's null or pointing to something with no elements. i.e. `m_NumElements` may be 0 and yet `m_Data` may be not null.)

#### Policies

The following Policies will be used by `itk::VariableLengthVector::SetSize`

See itk::VariableLengthVector for additional documentation.

### Vector Dot Product

### Synopsis

Dot product of Vectors

### Results

Output:

```
u :[-1, 1, -1]
v :[1, 2, 3]
DotProduct( u, v ) = -2
u - DotProduct( u, v ) * v = [1, 5, 5]
```

### Code

### C++

```
#include "itkVector.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using VectorType = itk::Vector<CoordType, Dimension>;

  VectorType u;
  u[0] = -1.;
  u[1] = 1.;
  u[2] = -1.;

  VectorType v;
  v[0] = 1.;
  v[1] = 2.;
  v[2] = 3.;

  std::cout << "u :" << u << std::endl;
  std::cout << "v :" << v << std::endl;
  std::cout << "DotProduct( u, v ) = " << u * v << std::endl;
  std::cout << "u - ( u * v ) * v = " << u - (u * v) * v << std::endl;

  return EXIT_SUCCESS;
}
```

### Python

```python
#!/usr/bin/env python

import itk
from numpy.core import double

Dimension = 3
CoordType = itk.D

VectorType = itk.Vector[CoordType, Dimension]

u = VectorType()
u[0] = -1.0
u[1] = 1.0
u[2] = -1.0

v = VectorType()
v[0] = 1.0
v[1] = 2.0
v[2] = 3.0

print("u: " + str(u))
print("v: " + str(v))
print("DotProduct( u, v ) = " + str(u * v))
print("DotProduct( u, v ) = " + str(u - double(u * v) * v))
```

### Classes demonstrated

template<typename **T**, unsigned int **NVectorDimension** = 3>
**class Vector** : **public** itk::*FixedArray*<*T*, *NVectorDimension*>

A templated class holding a n-Dimensional vector.

Vector is a templated class that holds a single vector (i.e., an array of values). Vector can be used as the data type held at each pixel in an Image or at each vertex of an Mesh. The template parameter T can be any data type that behaves like a primitive (or atomic) data type (int, short, float, complex). The NVectorDimension defines the number of components in the vector array.

Vector is not a dynamically extendible array like std::vector. It is intended to be used like a mathematical vector.

If you wish a simpler pixel types, you can use Scalar, which represents a single data value at a pixel. There is also the more complex type ScalarVector, which supports (for a given pixel) a single scalar value plus an array of vector values. (The scalar and vectors can be of different data type.)

**See** Image

**See** Mesh

**See** Point

**See** CovariantVector

**See** Matrix

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Create a vector
- Dot product (inner product) of two vectors

See itk::Vector for additional documentation.

### Watch a Filter

### Synopsis

This example demonstrates how to watch what is going on inside of a filter.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkSimpleFilterWatcher.h"
#include "itkThresholdImageFilter.h"
#include "itkImageRegionIterator.h"

template <class TImage>
void
CreateImage(typename TImage::Pointer image)
{
  using ImageType = TImage;

  // Create an image with 2 connected components
  typename ImageType::RegionType region;
  typename ImageType::IndexType  start;
  start.Fill(0);

  typename ImageType::SizeType size;
  size.Fill(100);

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(255);
    ++imageIterator;
  }
}
int
main(int, char *[])
```

(continues on next page)

(continued from previous page)

```
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::Pointer image = ImageType::New();
  CreateImage<ImageType>(image);

  using ThresholdImageFilterType = itk::ThresholdImageFilter<ImageType>;

  ThresholdImageFilterType::Pointer thresholdFilter = ThresholdImageFilterType::New();
  thresholdFilter->SetInput(image);
  thresholdFilter->ThresholdBelow(100);
  thresholdFilter->SetOutsideValue(0);

  itk::SimpleFilterWatcher watcher(thresholdFilter, "ThresholdFilter");

  thresholdFilter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class SimpleFilterWatcher**

Simple mechanism for monitoring the pipeline events of a filter and reporting these events to std::cout.

SimpleFilterWatcher provides a simple mechanism for monitoring the execution of filter. SimpleFilterWatcher is a stack-based object which takes a pointer to a ProcessObject at constructor time. SimpleFilterWatcher creates a series of commands that are registered as observers to the specified ProcessObject. The events monitored are:

```
StartEvent
EndEvent
ProgressEvent
IterationEvent
AbortEvent
```

The callbacks routines registered for these events emit a simple message to std::cout.

Example of use:

using FilterType = itk::BinaryThresholdImageFilter<ImageType>; FilterType::Pointer thresholdFilter = FilterType::New();

SimpleFilterWatcher watcher(thresholdFilter, “Threshold”);

The second argument to the constructor to SimpleFilterWatcher is an optional string that is prepended to the event messages. This allows the user to associate the emitted messages to a particular filter/variable.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Watch A filter

Subclassed by itk::XMLFilterWatcher

See itk::SimpleFilterWatcher for additional documentation.

### Write a PointSet

### Synopsis

Write a PointSet

### Results

Stores all points

### Code

### C++

```
#include "itkPointSet.h"

int
main(int argc, char * argv[])
{
  // Verify the number of parameters in the command line
  if (argc < 1)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " outputFile " << std::endl;
    return EXIT_FAILURE;
  }

  std::string outputFilename = argv[1];

  // Store points
  typedef itk::PointSet<double, 3> PointSetType;
  PointSetType::Pointer            pointsSet = PointSetType::New();
  typedef PointSetType::PointType  PointType;


  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class PointSet** : **public** itk::DataObject

A superclass of the N-dimensional mesh structure; supports point (geometric coordinate and attribute) definition.

PointSet is a superclass of the N-dimensional mesh structure (itk::Mesh). It provides the portion of the mesh definition for geometric coordinates (and associated attribute or pixel information). The defined API provides operations on points but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier to access the points. See DefaultStaticMeshTraits for the set of type

definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

PointSet has two template parameters. The first is the pixel type, or the type of data stored (optionally) with the points. The second is the "MeshTraits" structure controlling type information characterizing the point set. Most users will be happy with the defaults, and will not have to worry about this second argument.

Template parameters for PointSet:

TPixelType = The type stored as data for the point.

TMeshTraits = Type information structure for the point set.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create a PointSet
- Read a PointSet
- Write a PointSet
- Bounding Box Of A Point Set

Subclassed by itk::Mesh< TPixelType, VDimension, TMeshTraits >

See itk::PointSet for additional documentation.

### 3.2.2 ImageAdaptors

#### Add Constant to Every Pixel Without Duplicating Memory

#### Synopsis

Add a constant to every pixel without duplicating the image in memory (an accessor).

#### Results

Output:

```
addPixelAccessor.SetValue(5)
image->GetPixel[0, 0]: 20 adaptor->GetPixel[0, 0]: 25
addPixelAccessor.SetValue(100)
image->GetPixel[0, 0]: 20 adaptor->GetPixel[0, 0]: 120
```

### Code

### C++

```
#include "itkImage.h"
#include "itkAddPixelAccessor.h"
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"

using ImageType = itk::Image<unsigned int, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using AddPixelAccessorType = itk::Accessor::AddPixelAccessor<ImageType::PixelType>;
  using ImageAdaptorType = itk::ImageAdaptor<ImageType, AddPixelAccessorType>;

  ImageAdaptorType::Pointer adaptor = ImageAdaptorType::New();
  AddPixelAccessorType      addPixelAccessor;


  adaptor->SetImage(image);

  ImageType::IndexType index;
  index[0] = 0;
  index[1] = 0;

  addPixelAccessor.SetValue(5);
  adaptor->SetPixelAccessor(addPixelAccessor);
  std::cout << "addPixelAccessor.SetValue(5)" << std::endl;
  std::cout << "\timage->GetPixel" << index << ": " << image->GetPixel(index) << "␣
↪adaptor->GetPixel" << index << ": "
            << adaptor->GetPixel(index) << std::endl;

  addPixelAccessor.SetValue(100);
  adaptor->SetPixelAccessor(addPixelAccessor);
  std::cout << "addPixelAccessor.SetValue(100)" << std::endl;
  std::cout << "\timage->GetPixel" << index << ": " << image->GetPixel(index) << "␣
↪adaptor->GetPixel" << index << ": "
            << adaptor->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
```

(continues on next page)

(continued from previous page)

```
  size.Fill(10);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(20);
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TPixel**>
**class AddPixelAccessor**

Simulates the effect of adding a constant value to all pixels.

This class is intended to be used as parameter of an ImageAdaptor to make an image appear as having pixels with intensity values increased by a constant amount.

**See** ImageAdaptor

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Constant To Every Pixel Without Duplicating Memory

See itk::Accessor::AddPixelAccessor for additional documentation.

### Extract Channel of Image With Multiple Components

### Synopsis

Extract a component/channel of an itkImage with pixels with multiple components.

### Results

Output:

```
[1, 2, 3]
1
```

### Code

#### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"
#include "itkNthElementImageAdaptor.h"

using VectorImageType = itk::Image<itk::CovariantVector<float, 3>, 2>;

static void
CreateImage(VectorImageType::Pointer image);

int
main(int, char *[])
{
  VectorImageType::Pointer image = VectorImageType::New();
  CreateImage(image);

  itk::Index<2> index;
  index.Fill(0);

  std::cout << image->GetPixel(index) << std::endl;

  using ImageAdaptorType = itk::NthElementImageAdaptor<VectorImageType, float>;

  ImageAdaptorType::Pointer adaptor = ImageAdaptorType::New();

  adaptor->SelectNthElement(0);
  adaptor->SetImage(image);

  std::cout << adaptor->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(VectorImageType::Pointer image)
{
  VectorImageType::IndexType start;
  start.Fill(0);

  VectorImageType::SizeType size;
  size.Fill(2);

  VectorImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
```

(continues on next page)

(continued from previous page)

```
  image->SetRegions(region);
  image->Allocate();

  // Create a pixel and fill it with (1,2,3), and set every pixel of the image to␣
↪this pixel.

  itk::ImageRegionIterator<VectorImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  itk::CovariantVector<float, 3>            vectorPixel;
  vectorPixel[0] = 1;
  vectorPixel[1] = 2;
  vectorPixel[2] = 3;

  image->FillBuffer(vectorPixel);
}
```

### Classes demonstrated

template<typename **TImage**, typename **TOutputPixelType**>
**class NthElementImageAdaptor** : **public** itk::*ImageAdaptor*<*TImage*, PixelAccessor>

Presents an image as being composed of the N-th element of its pixels.

It assumes that the pixels are of container type and have in their API an operator[]( unsigned int ) defined.

Additional casting is performed according to the input and output image types following C++ default casting rules.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Channel Of Image With Multiple Components
- Process Nth Component Of Vector Image

See itk::NthElementImageAdaptor for additional documentation.

### Present Image After Operation

### Synopsis

Present an image by first performing an operation.

### Results

Output:

```
[1, 2, 3]
1
```

### Code

#### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"

using ScalarImageType = itk::Image<float, 2>;
using VectorImageType = itk::Image<itk::CovariantVector<float, 3>, 2>;


static void
CreateImage(VectorImageType::Pointer image);

class VectorPixelAccessor
{
public:
  using InternalType = itk::CovariantVector<float, 3>;
  using ExternalType = float;

  void
  operator=(const VectorPixelAccessor & vpa)
  {
    m_Index = vpa.m_Index;
  }
  ExternalType
  Get(const InternalType & input) const
  {
    return static_cast<ExternalType>(input[m_Index]);
  }
  void
  SetIndex(unsigned int index)
  {
    m_Index = index;
  }

private:
  unsigned int m_Index;
};

int
main(int, char *[])
{
  VectorImageType::Pointer image = VectorImageType::New();
  CreateImage(image);

  itk::Index<2> index;
  index.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  std::cout << image->GetPixel(index) << std::endl;

  using ImageAdaptorType = itk::ImageAdaptor<VectorImageType, VectorPixelAccessor>;

  ImageAdaptorType::Pointer adaptor = ImageAdaptorType::New();

  VectorPixelAccessor accessor;
  accessor.SetIndex(0);
  adaptor->SetPixelAccessor(accessor);
  adaptor->SetImage(image);

  std::cout << adaptor->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(VectorImageType::Pointer image)
{
  VectorImageType::IndexType start;
  start.Fill(0);

  VectorImageType::SizeType size;
  size.Fill(2);

  VectorImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<VectorImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  itk::CovariantVector<float, 3>            vec;
  vec[0] = 1;
  vec[1] = 2;
  vec[2] = 3;

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(vec);

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TImage**, typename **TAccessor**>
**class ImageAdaptor** : **public** itk::ImageBase<*TImage*::ImageDimension>
Give access to partial aspects of voxels from an Image.

ImageAdaptors are templated over the ImageType and over a functor that will specify what part of the pixel can be accessed

The basic aspects of this class are the types it defines.

Image adaptors can be used as intermediate classes that allow the sending of an image to a filter, specifying what part of the image pixels the filter will act on.

The TAccessor class should implement the Get and Set methods These two will specify how data can be put and get from parts of each pixel. It should define the types ExternalType and InternalType too.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Present Image After Operation

Subclassed by itk::NthElementImageAdaptor< TImage, TOutputPixelType >

See itk::ImageAdaptor for additional documentation.

### Process Nth Component of Vector Image

### Synopsis

Process the nth component/element/channel of a vector image.

### Results

### Code

### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"
#include "itkNthElementImageAdaptor.h"
#include "itkBinomialBlurImageFilter.h"

using VectorImageType = itk::Image<itk::CovariantVector<float, 3>, 2>;

static void
CreateImage(VectorImageType::Pointer image);

int
main(int, char *[])
{
  VectorImageType::Pointer image = VectorImageType::New();
  CreateImage(image);
```

(continues on next page)

(continued from previous page)

```
  using ImageAdaptorType = itk::NthElementImageAdaptor<VectorImageType, float>;

  ImageAdaptorType::Pointer adaptor = ImageAdaptorType::New();

  adaptor->SelectNthElement(0);
  adaptor->SetImage(image);

  using BlurFilterType = itk::BinomialBlurImageFilter<ImageAdaptorType, itk::Image
↪<float, 2>>;
  BlurFilterType::Pointer blurFilter = BlurFilterType::New();
  blurFilter->SetInput(adaptor);
  blurFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(VectorImageType::Pointer image)
{
  VectorImageType::IndexType start;
  start.Fill(0);

  VectorImageType::SizeType size;
  size.Fill(2);

  VectorImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<VectorImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  itk::CovariantVector<float, 3>            vec;
  vec[0] = 1;
  vec[1] = 2;
  vec[2] = 3;

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(vec);

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TImage**, typename **TOutputPixelType**>
**class NthElementImageAdaptor** : **public** itk::*ImageAdaptor*<*TImage*, PixelAccessor>
Presents an image as being composed of the N-th element of its pixels.

It assumes that the pixels are of container type and have in their API an operator[]( unsigned int ) defined.

Additional casting is performed according to the input and output image types following C++ default casting rules.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Channel Of Image With Multiple Components
- Process Nth Component Of Vector Image

See itk::NthElementImageAdaptor for additional documentation.

### View Component Vector Image as Scalar Image

### Synopsis

View a component of a vector image as if it were a scalar image

### Results

Output:

```
1
```

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkImage.h"
#include "itkVectorImageToImageAdaptor.h"
#include "itkImageRegionIterator.h"

using ScalarImageType = itk::Image<float, 2>;
using VectorImageType = itk::VectorImage<float, 2>;

void
CreateImage(VectorImageType::Pointer image);

int
main(int, char *[])
{
  VectorImageType::Pointer image = VectorImageType::New();
```

(continues on next page)

(continued from previous page)

```
  CreateImage(image);

  using ImageAdaptorType = itk::VectorImageToImageAdaptor<float, 2>;
  ImageAdaptorType::Pointer adaptor = ImageAdaptorType::New();
  adaptor->SetExtractComponentIndex(0);
  adaptor->SetImage(image);

  itk::Index<2> index;
  index[0] = 0;
  index[1] = 0;

  std::cout << adaptor->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(VectorImageType::Pointer image)
{
  VectorImageType::IndexType start;
  start.Fill(0);

  VectorImageType::SizeType size;
  size.Fill(2);

  VectorImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->SetNumberOfComponentsPerPixel(2);
  image->Allocate();

  using VariableVectorType = itk::VariableLengthVector<double>;
  VariableVectorType variableLengthVector;
  variableLengthVector.SetSize(2);

  itk::ImageRegionIterator<VectorImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  variableLengthVector[0] = 1;
  variableLengthVector[1] = 2;

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(variableLengthVector);

    ++imageIterator;
  }
}
```

#### Classes demonstrated

template<typename **TPixelType**, unsigned int **Dimension**>
**class VectorImageToImageAdaptor** : **public** itk::*ImageAdaptor*<*VectorImage*<*TPixelType*, *Dimension*>, Accessor::Vector
Presents a VectorImage and extracts a component from it into an image.

The class is expected to be templated over a pixel type and dimension. These are the pixel types and dimension of the VectorImage.

The component to extract is set with SetExtractComponentIdx() method.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- View Component Vector Image As Scalar Image

See itk::VectorImageToImageAdaptor for additional documentation.

### 3.2.3 ImageFunction

#### Compute Median of Image at Pixel

#### Synopsis

Computes the median of an image at a pixil(in a regular neighborhood).

#### Results

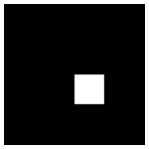


Fig. 50: input.png

Output:

```
Median at [10, 10] is 0
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkMedianImageFunction.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  using MedianImageFunctionType = itk::MedianImageFunction<UnsignedCharImageType>;
  MedianImageFunctionType::Pointer medianImageFunction = MedianImageFunctionType::
↪New();
  medianImageFunction->SetInputImage(image);

  itk::Index<2> index;
  index.Fill(10);
  std::cout << "Median at " << index << " is " << static_cast<int>
↪(medianImageFunction->EvaluateAtIndex(index))
            << std::endl;
  return EXIT_SUCCESS;
}

void
CreateImage(UnsignedCharImageType::Pointer image)
{
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] >= 50 && imageIterator.GetIndex()[1] >= 50 &&␣
↪imageIterator.GetIndex()[0] <= 70 &&
        imageIterator.GetIndex()[1] <= 70)
    {
      imageIterator.Set(255);
```

(continues on next page)

(continued from previous page)

```
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TCoordRep** = float>
**class MedianImageFunction** : **public** itk::ImageFunction<*TInputImage*, *TInputImage*::PixelType, *TCoordRep*>
Calculate the median value in the neighborhood of a pixel.

Calculate the median pixel value over the standard 8, 26, etc. connected neighborhood. This calculation uses a ZeroFluxNeumannBoundaryCondition.

If called with a ContinuousIndex or Point, the calculation is performed at the nearest neighbor.

This class is templated over the input image type and the coordinate representation type (e.g. float or double ).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Median Of Image At Pixel

See itk::MedianImageFunction for additional documentation.

### Linearly Interpolate Position in Image

### Synopsis

Linearly interpolate a position in an image.

### Results

Output:

```
Value at 1.3: 13
```

### Code

### C++

```
#include "itkImage.h"
#include "itkContinuousIndex.h"
#include "itkLinearInterpolateImageFunction.h"

using ImageType = itk::Image<unsigned char, 1>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  itk::ContinuousIndex<double, 1> pixel;
  pixel[0] = 1.3;

  itk::LinearInterpolateImageFunction<ImageType, double>::Pointer interpolator =
    itk::LinearInterpolateImageFunction<ImageType, double>::New();
  interpolator->SetInputImage(image);

  std::cout << "Value at 1.3: " << interpolator->EvaluateAtContinuousIndex(pixel) <<␣
↪std::endl;

  return EXIT_SUCCESS;
}


void
CreateImage(ImageType::Pointer image)
{
  // Create a 1D image
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;

  ImageType::SizeType size;
  size[0] = 10;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  for (unsigned int i = 0; i < 10; i++)
  {
    ImageType::IndexType pixelIndex;
    pixelIndex[0] = i;

    image->SetPixel(pixelIndex, i * 10);
  }
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TCoordRep** = double>
**class LinearInterpolateImageFunction** : **public** itk::InterpolateImageFunction<*TInputImage*, *TCoordRep*>
Linearly interpolate an image at specified positions.

LinearInterpolateImageFunction linearly interpolates image intensity at a non-integer pixel position. This class is templated over the input image type and the coordinate representation type (e.g. float or double).

This function works for N-dimensional images.

This function works for images with scalar and vector pixel types, and for images of type VectorImage.

**See** VectorLinearInterpolateImageFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Linearly Interpolate Position In Image

See itk::LinearInterpolateImageFunction for additional documentation.

### Multiply Kernel With an Image at Location

### Synopsis

This will multiply a kernel with an image at a particular location.

### Results

Output:

```
Sum on border: 4
Sum in center: 9
Sum outside: 0
```

### Code

### C++

```
#include "itkImage.h"
#include "itkNeighborhoodOperatorImageFunction.h"
#include "itkNeighborhoodOperator.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
```

(continues on next page)

(continued from previous page)

```
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  itk::Neighborhood<float, 2> neighborhood;
  neighborhood.SetRadius(1);
  for (unsigned int i = 0; i < neighborhood.GetSize()[0] * neighborhood.GetSize()[1];␣
↪++i)
  {
    neighborhood[i] = 1;
  }

  using NeighborhoodOperatorImageFunctionType = itk::NeighborhoodOperatorImageFunction
↪<UnsignedCharImageType, float>;
  NeighborhoodOperatorImageFunctionType::Pointer neighborhoodOperatorImageFunction =
    NeighborhoodOperatorImageFunctionType::New();
  neighborhoodOperatorImageFunction->SetOperator(neighborhood);
  neighborhoodOperatorImageFunction->SetInputImage(image);

  {
    itk::Index<2> index;
    index.Fill(20);

    float output = neighborhoodOperatorImageFunction->EvaluateAtIndex(index);
    std::cout << "Sum on border: " << output << std::endl;
  }

  {
    itk::Index<2> index;
    index.Fill(35);

    float output = neighborhoodOperatorImageFunction->EvaluateAtIndex(index);
    std::cout << "Sum in center: " << output << std::endl;
  }

  {
    itk::Index<2> index;
    index.Fill(7);

    float output = neighborhoodOperatorImageFunction->EvaluateAtIndex(index);
    std::cout << "Sum outside: " << output << std::endl;
  }

  return EXIT_SUCCESS;
}

void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image with 2 connected components
  UnsignedCharImageType::IndexType start;
  start.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  UnsignedCharImageType::SizeType size;
  size.Fill(100);

  UnsignedCharImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 80; c++)
    {
      UnsignedCharImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 1);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutput**>
**class NeighborhoodOperatorImageFunction** : **public** itk::ImageFunction<*TInputImage*, *TOutput*>
Compute the convolution of a neighborhood operator with the image at a specific location in space, i.e. point, index or continuous index. This class is templated over the input image type.

**See** NeighborhoodOperator

**See** ImageFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiply Kernel With An Image At Location

See itk::NeighborhoodOperatorImageFunction for additional documentation.

### Resample Segmented Image

### Synopsis

Resample (stretch or compress) a label image resulting form segmentation.

For more on why label image interpolation is necessary and how it works, see the associated Insight Journal article.

### Results

Fig. 51: Input label image

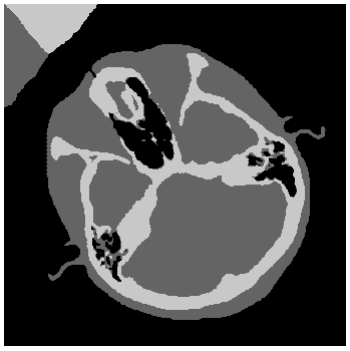


Fig. 52: Resample with label image gaussian interpolation.

Fig. 53: Resample with nearest neighbor interpolation.

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse
```

(continues on next page)

(continued from previous page)

```
parser = argparse.ArgumentParser(description="Resample Segmented Image.")
parser.add_argument("input_image")
parser.add_argument("spacing_fraction", type=float)
parser.add_argument("sigma_fraction", type=float)
parser.add_argument("output_image_file_label_image_interpolator")
parser.add_argument("output_image_file_nearest_neighbor_interpolator")
args = parser.parse_args()

input_image = itk.imread(args.input_image)

resize_filter = itk.ResampleImageFilter.New(input_image)

input_spacing = itk.spacing(input_image)
output_spacing = [s * args.spacing_fraction for s in input_spacing]
resize_filter.SetOutputSpacing(output_spacing)

input_size = itk.size(input_image)
output_size = [
    int(s * input_spacing[dim] / args.spacing_fraction)
    for dim, s in enumerate(input_size)
]
resize_filter.SetSize(output_size)

gaussian_interpolator = itk.LabelImageGaussianInterpolateImageFunction.New(input_
↪image)
sigma = [s * args.sigma_fraction for s in output_spacing]
gaussian_interpolator.SetSigma(sigma)
gaussian_interpolator.SetAlpha(3.0)
resize_filter.SetInterpolator(gaussian_interpolator)

itk.imwrite(resize_filter, args.output_image_file_label_image_interpolator)

nearest_neighbor_interpolator = itk.NearestNeighborInterpolateImageFunction.New(
    input_image
)
resize_filter.SetInterpolator(nearest_neighbor_interpolator)

itk.imwrite(resize_filter, args.output_image_file_nearest_neighbor_interpolator)
```

### C++

```
#include <iostream>
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelImageGaussianInterpolateImageFunction.h"
#include "itkNearestNeighborInterpolateImageFunction.h"
#include "itkResampleImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Usage: " << argv[0]
              << " inputImageFile spacingFraction sigmaFraction␣
↪outputImageFileLabelImageInterpolator "
                 "outputImageFileNearestNeighborInterpolator"
              << std::endl;

    return EXIT_FAILURE;
  }
  const char * const inputImageFile = argv[1];
  const double       spacingFraction = std::stod(argv[2]);
  const double       sigmaFraction = std::stod(argv[3]);
  const char * const outputImageFileLabelImageInterpolator = argv[4];
  const char * const outputImageFileNearestNeighborInterpolator = argv[5];

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImageFile);
  reader->Update();

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  ResampleFilterType::Pointer resizeFilter = ResampleFilterType::New();
  resizeFilter->SetInput(reader->GetOutput());

  //     Compute and set the output size
  //
  //     The computation must be so that the following holds:
  //
  //     new width         old x spacing
  //     ----------   =   ---------------
  //     old width         new x spacing
  //
  //
  //     new height         old y spacing
  //    ------------  =   ---------------
  //     old height         new y spacing
  //
  //     So either we specify new height and width and compute new spacings
  //     or we specify new spacing and compute new height and width
  //     and computations that follows need to be modified a little (as it is

  const ImageType::SpacingType inputSpacing{ reader->GetOutput()->GetSpacing() };
  ImageType::SpacingType       outputSpacing;
  for (unsigned int dim = 0; dim < Dimension; ++dim)
  {
    outputSpacing[dim] = inputSpacing[dim] * spacingFraction;
  }
  resizeFilter->SetOutputSpacing(outputSpacing);

  const ImageType::RegionType inputRegion{ reader->GetOutput()->
↪GetLargestPossibleRegion() };
  const ImageType::SizeType   inputSize{ inputRegion.GetSize() };
  ImageType::SizeType         outputSize;
  for (unsigned int dim = 0; dim < Dimension; ++dim)
```

(continues on next page)

(continued from previous page)

```
  {
    outputSize[dim] = inputSize[dim] * inputSpacing[dim] / spacingFraction;
  }
  resizeFilter->SetSize(outputSize);

  using GaussianInterpolatorType = itk::LabelImageGaussianInterpolateImageFunction
↪<ImageType, double>;
  GaussianInterpolatorType::Pointer   gaussianInterpolator = GaussianInterpolatorType:
↪:New();
  GaussianInterpolatorType::ArrayType sigma;
  for (unsigned int dim = 0; dim < Dimension; ++dim)
  {
    sigma[dim] = outputSpacing[dim] * sigmaFraction;
  }
  gaussianInterpolator->SetSigma(sigma);
  gaussianInterpolator->SetAlpha(3.0);
  resizeFilter->SetInterpolator(gaussianInterpolator);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(resizeFilter->GetOutput());
  writer->SetFileName(outputImageFileLabelImageInterpolator);
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  using NearestNeighborInterpolatorType = itk::NearestNeighborInterpolateImageFunction
↪<ImageType, double>;
  NearestNeighborInterpolatorType::Pointer nearestNeighborInterpolator =␣
↪NearestNeighborInterpolatorType::New();
  resizeFilter->SetInterpolator(nearestNeighborInterpolator);

  writer->SetFileName(outputImageFileNearestNeighborInterpolator);
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TCoordRep** = double, typename **TPixelCompare** = std::less<**typename** itk::*Num*
**class LabelImageGaussianInterpolateImageFunction** : **public** itk::GaussianInterpolateImageFunction<*TInputImag*

Interpolation function for multi-label images that implicitly smooths each unique value in the image corresponding to each label set element and returns the corresponding label set element with the largest weight.

This filter is an alternative to nearest neighbor interpolation for multi-label images. Given a multi-label image `I` with label set `L`, this function returns a label at the non-voxel position `I(x)`, based on the following rule

$$I(x) = \arg\max_{l \in L} (G_\sigma * I_l)(x)$$

Where $I_l$ is the `l-th` binary component of the multilabel image. In other words, each label in the multi-label image is convolved with a Gaussian, and the label for which the response is largest is returned. For sigma=0, this is just nearest neighbor interpolation.

This class defines an N-dimensional Gaussian interpolation function for label using the vnl error function. The two parameters associated with this function are:

- `Sigma` - a scalar array of size ImageDimension determining the width of the interpolation function.
- `Alpha` - a scalar specifying the cutoff distance over which the function is calculated.

**Note** The input image can be of any type, but the number of unique intensity values in the image will determine the amount of memory needed to complete each interpolation.

**Author** Paul Yushkevich

**Author** Nick Tustison

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Resample Segmented Image

See itk::LabelImageGaussianInterpolateImageFunction for additional documentation.

### 3.2.4 Mesh

#### Add Points and Edges

#### Synopsis

How to add points and edges to a mesh.

### Results

Output:

```
Points = 4
[-1, -1, 0]
[1, -1, 0]
[1, 1, 0]
[1, 1, 1]
line first point id: 2
line second point id: 3
```

### Code

### C++

```
#include "itkMesh.h"
#include "itkLineCell.h"

constexpr unsigned int Dimension = 3;
using MeshType = itk::Mesh<float, Dimension>;

MeshType::Pointer
CreatePointOnlyMesh();
void
CreateMeshWithEdges();

int
main(int, char *[])
{
  // CreatePointOnlyMesh();
  CreateMeshWithEdges();

  return 0;
}


MeshType::Pointer
CreatePointOnlyMesh()
{
  MeshType::Pointer mesh = MeshType::New();

  // Create points
  MeshType::PointType p0, p1, p2, p3;

  p0[0] = -1.0;
  p0[1] = -1.0;
  p0[2] = 0.0; // first  point ( -1, -1, 0 )
  p1[0] = 1.0;
  p1[1] = -1.0;
  p1[2] = 0.0; // second point (  1, -1, 0 )
  p2[0] = 1.0;
  p2[1] = 1.0;
  p2[2] = 0.0; // third  point (  1,  1, 0 )
  p3[0] = 1.0;
```

(continues on next page)

(continued from previous page)

```
  p3[1] = 1.0;
  p3[2] = 1.0; // third  point (  1,  1, 1 )

  mesh->SetPoint(0, p0);
  mesh->SetPoint(1, p1);
  mesh->SetPoint(2, p2);
  mesh->SetPoint(3, p3);

  std::cout << "Points = " << mesh->GetNumberOfPoints() << std::endl;

  // Access points
  using PointsIterator = MeshType::PointsContainer::Iterator;

  PointsIterator pointIterator = mesh->GetPoints()->Begin();

  PointsIterator end = mesh->GetPoints()->End();
  while (pointIterator != end)
  {
    MeshType::PointType p = pointIterator.Value(); // access the point
    std::cout << p << std::endl;                   // print the point
    ++pointIterator;                               // advance to next point
  }

  return mesh;
}

void
CreateMeshWithEdges()
{
  MeshType::Pointer mesh = CreatePointOnlyMesh();

  using CellAutoPointer = MeshType::CellType::CellAutoPointer;
  using LineType = itk::LineCell<MeshType::CellType>;

  // Create a link to the previous point in the column (below the current point)
  CellAutoPointer colline;
  colline.TakeOwnership(new LineType);

  // unsigned int pointId0 = 0;
  // unsigned int pointId1 = 1;

  unsigned int pointId0 = 2;
  unsigned int pointId1 = 3;

  colline->SetPointId(0, pointId0); // line between points 0 and 1
  colline->SetPointId(1, pointId1);
  // std::cout << "Linked point: " << MeshIndex << " and " << MeshIndex - 1 << std::
↪endl;
  mesh->SetCell(0, colline);

  using CellIterator = MeshType::CellsContainer::Iterator;
  CellIterator cellIterator = mesh->GetCells()->Begin();
  CellIterator CellsEnd = mesh->GetCells()->End();

  while (cellIterator != CellsEnd)
  {
    MeshType::CellType * cellptr = cellIterator.Value();
```

(continues on next page)

(continued from previous page)

```
    auto *               line = dynamic_cast<LineType *>(cellptr);

    itk::IdentifierType * linePoint0 = line->PointIdsBegin();
    // itk::IdentifierType* linePoint1 = line->PointIdsEnd();
    itk::IdentifierType * linePoint1 = linePoint0 + 1;
    std::cout << "line first point id: " << *linePoint0 << std::endl;
    std::cout << "line second point id: " << *linePoint1 << std::endl;

    ++cellIterator;
  }
}
```

## Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class Mesh** : **public** itk::*PointSet*<*TPixelType*, *VDimension*, *TMeshTraits*>

Implements the N-dimensional mesh structure.

Mesh is an adaptive, evolving structure. Typically points and cells are created, with the cells referring to their defining points. If additional topological information is required, then BuildCellLinks() is called and links from the points back to the cells that use them are created. This allows implicit topological information about the faces and edges of the cells to be determined. (For example, a "face" neighbor to a cell can be determined by intersection the sets of cells that use the points defining the face. This is an inherent assumption on the manifold relationship of the cells in the mesh.) In some cases, either because the mesh is non-manifold, because we wish to explicitly store information with the faces and edges of the mesh, or because performance requirements demand that boundaries are explicitly represented (the set intersection does not need to be performed); then Mesh can be further extended by adding explicit boundary assignments.

**Overview** Mesh implements the N-dimensional mesh structure for ITK. It provides an API to perform operations on points, cells, boundaries, etc., but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier types that will be used to access the mesh. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

One of the most important parts of using this mesh is how to create cells to insert into it. The cells for the mesh take two template parameters. The first is the pixel type, and should correspond exactly to that type given to the mesh. The second is a "CellTraits" which holds a sub-set of the "MeshTraits" structure definitions, and is also a member of them. Any cell which is to be inserted to a mesh should have MeshTraits::CellTraits as its second template parameter.

**Usage** Mesh has three template parameters. The first is the pixel type, or the type of data stored (optionally) with points, cells, and/or boundaries. The second is the geometric dimension of the points defining the mesh. This also limits the maximum topological dimension of the cells that can be inserted. The third template parameter is the "MeshTraits" structure controlling type information for the mesh. Most users will be happy with the defaults, and will not have to worry about this third argument.

Template parameters for Mesh:

TPixelType = The type stored as data for an entity (cell, point, or boundary).

TMeshTraits = Type information structure for the mesh.

**References** No reference information is available.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Add Points And Edges
- Convert Mesh To Unstructered Grid
- Working With Point And Cell Data

Subclassed by itk::SimplexMesh< TPixelType, VDimension, TMeshTraits >

See itk::Mesh for additional documentation.

### Calculate Area and Volume of Simplex Mesh

### Synopsis

Calculate the area and volume of an itk::SimplexMesh.

### Results

Output:

```
Ideal Volume: 523.599
Mesh Volume: 520.812
Ideal Surface Area: 314.159
Mesh Surface Area: 313.1
```

### Code

### C++

```
#define _USE_MATH_DEFINES // needed for Visual Studio (before #include <cmath>)
#include <itkSimplexMesh.h>
#include <itkRegularSphereMeshSource.h>
#include <itkTriangleMeshToSimplexMeshFilter.h>
#include <itkSimplexMeshVolumeCalculator.h>

using TMesh = itk::Mesh<float, 3>;
using TSimplex = itk::SimplexMesh<float, 3>;
using TSphere = itk::RegularSphereMeshSource<TMesh>;
using TConvert = itk::TriangleMeshToSimplexMeshFilter<TMesh, TSimplex>;
using TVolume = itk::SimplexMeshVolumeCalculator<TSimplex>;

int
main(int, char *[])
{

  // Create a spherical mesh with known radius and resolution.
  TSphere::Pointer    source = TSphere::New();
  TSphere::VectorType scale;
  scale.Fill(5.0);
  source->SetScale(scale);
  source->SetResolution(5);
  source->Update();
```

(continues on next page)

(continued from previous page)

```
  // Ensure that all cells of the mesh are triangles.
  for (TMesh::CellsContainerIterator it = source->GetOutput()->GetCells()->Begin();
       it != source->GetOutput()->GetCells()->End();
       ++it)
  {
    TMesh::CellAutoPointer cell;
    source->GetOutput()->GetCell(it->Index(), cell);
    if (3 != cell->GetNumberOfPoints())
    {
      std::cerr << "ERROR: All cells must be trianglar." << std::endl;
      return EXIT_FAILURE;
    }
  }

  // Convert the triangle mesh to a simplex mesh.
  TConvert::Pointer convert = TConvert::New();
  convert->SetInput(source->GetOutput());
  convert->Update();

  // Calculate the volume and area of the simplex mesh.
  TVolume::Pointer volume = TVolume::New();
  volume->SetSimplexMesh(convert->GetOutput());
  volume->Compute();

  // Compare with the volume and area of an ideal sphere.
  std::cout << "Ideal Volume: " << 4.0 / 3.0 * M_PI * pow(5.0, 3) << std::endl;
  std::cout << "Mesh Volume: " << volume->GetVolume() << std::endl;
  std::cout << "Ideal Surface Area: " << 4.0 * M_PI * pow(5.0, 2) << std::endl;
  std::cout << "Mesh Surface Area: " << volume->GetArea() << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**>
**class SimplexMeshVolumeCalculator** : **public** itk::*Object*

Adapted from itkSimplexMeshToTriangleFilter to calculate the volume of a simplex mesh using the barycenters and normals. call Compute() to calculate the volume and GetVolume() to get the value. For an example see itkDeformableSimplexMesh3DFilter.cxx (Thomas Boettger. Division Medical and Biological Informatics, German Cancer Research Center, Heidelberg.)

The original implementation has been replaced with an algorithm based on the discrete form of the divergence theorem. The general assumption here is that the model is of closed surface. For more details see the following reference (Alyassin A.M. et al, "Evaluation of new algorithms for the interactive measurement of

surface area and volume", Med Phys 21(6) 1994.).

**Author** Leila Baghdadi MICe, Hospital for Sick Children, Toronto, Canada.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Calculate Area And Volume Of Simplex Mesh

See itk::SimplexMeshVolumeCalculator for additional documentation.

### Convert Mesh to Unstructered Grid

### Synopsis

Convert an itk::Mesh to a vtkUnstructuredGrid.

### Results

An Output.vtu file will be generated alone with the following output.

Output:

```
Unstructered grid has 3 cells.
```

### Code

### C++

```
// ITK
#include "itkLineCell.h"
#include "itkMesh.h"
#include "itkTriangleCell.h"
#include "itkQuadrilateralCell.h"

// VTK
#include "vtkVersion.h"
#include <vtkCellArray.h>
#include <vtkSmartPointer.h>
#include <vtkUnstructuredGrid.h>
#include <vtkXMLUnstructuredGridWriter.h>

using MeshType = itk::Mesh<float, 3>;

// Functions
static MeshType::Pointer
CreateMeshWithEdges();
static void
ConvertMeshToUnstructuredGrid(MeshType::Pointer, vtkUnstructuredGrid *);

class VisitVTKCellsClass
{
  vtkCellArray * m_Cells;
  int *          m_LastCell;
  int *          m_TypeArray;

public:
  using CellInterfaceType = itk::CellInterface<MeshType::PixelType, MeshType::
↪CellTraits>;

  using floatLineCell = itk::LineCell<CellInterfaceType>;
```

(continues on next page)

(continued from previous page)

```
  using floatTriangleCell = itk::TriangleCell<CellInterfaceType>;
  using floatQuadrilateralCell = itk::QuadrilateralCell<CellInterfaceType>;

  // Set the vtkCellArray that will be constructed
  void
  SetCellArray(vtkCellArray * a)
  {
    m_Cells = a;
  }

  // Set the cell counter pointer
  void
  SetCellCounter(int * i)
  {
    m_LastCell = i;
  }

  // Set the type array for storing the vtk cell types
  void
  SetTypeArray(int * i)
  {
    m_TypeArray = i;
  }

  // Visit a triangle and create the VTK_TRIANGLE cell
  void
  Visit(unsigned long, floatTriangleCell * t)
  {
    m_Cells->InsertNextCell(3, (vtkIdType *)t->PointIdsBegin());
    m_TypeArray[*m_LastCell] = VTK_TRIANGLE;
    (*m_LastCell)++;
  }

  // Visit a triangle and create the VTK_QUAD cell
  void
  Visit(unsigned long, floatQuadrilateralCell * t)
  {
    m_Cells->InsertNextCell(4, (vtkIdType *)t->PointIdsBegin());
    m_TypeArray[*m_LastCell] = VTK_QUAD;
    (*m_LastCell)++;
  }

  // Visit a line and create the VTK_LINE cell
  void
  Visit(unsigned long, floatLineCell * t)
  {
    m_Cells->InsertNextCell(2, (vtkIdType *)t->PointIdsBegin());
    m_TypeArray[*m_LastCell] = VTK_LINE;
    (*m_LastCell)++;
  }
};

int
main(int, char *[])
{
  MeshType::Pointer mesh = CreateMeshWithEdges();
```

(continues on next page)

(continued from previous page)

```
  vtkSmartPointer<vtkUnstructuredGrid> unstructuredGrid = vtkSmartPointer
↪<vtkUnstructuredGrid>::New();
  ConvertMeshToUnstructuredGrid(mesh, unstructuredGrid);

  // Write file
  vtkSmartPointer<vtkXMLUnstructuredGridWriter> writer = vtkSmartPointer
↪<vtkXMLUnstructuredGridWriter>::New();
  writer->SetFileName("output.vtu");
#if VTK_MAJOR_VERSION <= 5
  writer->SetInputConnection(unstructuredGrid->GetProducerPort());
#else
  writer->SetInputData(unstructuredGrid);
#endif
  writer->Write();

  return EXIT_SUCCESS;
}

MeshType::Pointer
CreateMeshWithEdges()
{

  MeshType::Pointer mesh = MeshType::New();

  // Create 4 points and add them to the mesh
  MeshType::PointType p0, p1, p2, p3;

  p0[0] = -1.0;
  p0[1] = -1.0;
  p0[2] = 0.0;
  p1[0] = 1.0;
  p1[1] = -1.0;
  p1[2] = 0.0;
  p2[0] = 1.0;
  p2[1] = 1.0;
  p2[2] = 0.0;
  p3[0] = 1.0;
  p3[1] = 1.0;
  p3[2] = 1.0;

  mesh->SetPoint(0, p0);
  mesh->SetPoint(1, p1);
  mesh->SetPoint(2, p2);
  mesh->SetPoint(3, p3);

  // Create three lines and add them to the mesh
  using CellAutoPointer = MeshType::CellType::CellAutoPointer;
  using LineType = itk::LineCell<MeshType::CellType>;

  CellAutoPointer line0;
  line0.TakeOwnership(new LineType);
  line0->SetPointId(0, 0); // line between points 0 and 1
  line0->SetPointId(1, 1);
  mesh->SetCell(0, line0);

  CellAutoPointer line1;
  line1.TakeOwnership(new LineType);
```

(continues on next page)

(continued from previous page)

```
  line1->SetPointId(0, 1); // line between points 1 and 2
  line1->SetPointId(1, 2);
  mesh->SetCell(1, line1);

  CellAutoPointer line2;
  line2.TakeOwnership(new LineType);
  line2->SetPointId(0, 2); // line between points 2 and 3
  line2->SetPointId(1, 3);
  mesh->SetCell(2, line2);

  return mesh;
}

void
ConvertMeshToUnstructuredGrid(MeshType::Pointer mesh, vtkUnstructuredGrid *␣
↪unstructuredGrid)
{
  // Get the number of points in the mesh
  int numPoints = mesh->GetNumberOfPoints();
  if (numPoints == 0)
  {
    mesh->Print(std::cerr);
    std::cerr << "no points in Grid " << std::endl;
    exit(-1);
  }

  // Create the vtkPoints object and set the number of points
  vtkPoints * vpoints = vtkPoints::New();
  vpoints->SetNumberOfPoints(numPoints);
  // Iterate over all the points in the itk mesh filling in
  // the vtkPoints object as we go
  MeshType::PointsContainer::Pointer points = mesh->GetPoints();

  // In ITK the point container is not necessarily a vector, but in VTK it is
  vtkIdType                VTKId = 0;
  std::map<vtkIdType, int> IndexMap;

  for (MeshType::PointsContainer::Iterator i = points->Begin(); i != points->End();␣
↪++i, VTKId++)
  {
    // Get the point index from the point container iterator
    IndexMap[VTKId] = i->Index();

    // Set the vtk point at the index with the the coord array from itk
    // itk returns a const pointer, but vtk is not const correct, so
    // we have to use a const cast to get rid of the const
    vpoints->SetPoint(VTKId, const_cast<float *>(i->Value().GetDataPointer()));
  }

  // Set the points on the vtk grid
  unstructuredGrid->SetPoints(vpoints);

  // Setup some VTK things
  int    vtkCellCount = 0; // running counter for current cell being inserted into vtk
  int    numCells = mesh->GetNumberOfCells();
  auto * types = new int[numCells]; // type array for vtk
  // create vtk cells and estimate the size
```

(continues on next page)

(continued from previous page)

```
  vtkCellArray * cells = vtkCellArray::New();
  cells->EstimateSize(numCells, 4);

  // Setup the line visitor
  using LineVisitor = itk::CellInterfaceVisitorImplementation<
    float,
    MeshType::CellTraits,
    itk::LineCell<itk::CellInterface<MeshType::PixelType, MeshType::CellTraits>>,
    VisitVTKCellsClass>;
  LineVisitor::Pointer lv = LineVisitor::New();
  lv->SetTypeArray(types);
  lv->SetCellCounter(&vtkCellCount);
  lv->SetCellArray(cells);

  // Setup the triangle visitor
  using TriangleVisitor = itk::CellInterfaceVisitorImplementation<
    float,
    MeshType::CellTraits,
    itk::TriangleCell<itk::CellInterface<MeshType::PixelType, MeshType::CellTraits>>,
    VisitVTKCellsClass>;
  TriangleVisitor::Pointer tv = TriangleVisitor::New();
  tv->SetTypeArray(types);
  tv->SetCellCounter(&vtkCellCount);
  tv->SetCellArray(cells);

  // Setup the quadrilateral visitor
  using QuadrilateralVisitor = itk::CellInterfaceVisitorImplementation<
    float,
    MeshType::CellTraits,
    itk::QuadrilateralCell<itk::CellInterface<MeshType::PixelType, MeshType::
↪CellTraits>>,
    VisitVTKCellsClass>;
  QuadrilateralVisitor::Pointer qv = QuadrilateralVisitor::New();
  qv->SetTypeArray(types);
  qv->SetCellCounter(&vtkCellCount);
  qv->SetCellArray(cells);

  // Add the visitors to a multivisitor

  MeshType::CellType::MultiVisitor::Pointer mv = MeshType::CellType::MultiVisitor::
↪New();

  mv->AddVisitor(tv);
  mv->AddVisitor(qv);
  mv->AddVisitor(lv);

  // Now ask the mesh to accept the multivisitor which
  // will Call Visit for each cell in the mesh that matches the
  // cell types of the visitors added to the MultiVisitor
  mesh->Accept(mv);

  // Now set the cells on the vtk grid with the type array and cell array
  unstructuredGrid->SetCells(types, cells);
  std::cout << "Unstructured grid has " << unstructuredGrid->GetNumberOfCells() << "␣
↪cells." << std::endl;

  // Clean up vtk objects
```

(continues on next page)

(continued from previous page)

```
  cells->Delete();
  vpoints->Delete();
}
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class Mesh** : **public** itk::*PointSet*<*TPixelType*, *VDimension*, *TMeshTraits*>

Implements the N-dimensional mesh structure.

Mesh is an adaptive, evolving structure. Typically points and cells are created, with the cells referring to their defining points. If additional topological information is required, then BuildCellLinks() is called and links from the points back to the cells that use them are created. This allows implicit topological information about the faces and edges of the cells to be determined. (For example, a "face" neighbor to a cell can be determined by intersection the sets of cells that use the points defining the face. This is an inherent assumption on the manifold relationship of the cells in the mesh.) In some cases, either because the mesh is non-manifold, because we wish to explicitly store information with the faces and edges of the mesh, or because performance requirements demand that boundaries are explicitly represented (the set intersection does not need to be performed); then Mesh can be further extended by adding explicit boundary assignments.

**Overview** Mesh implements the N-dimensional mesh structure for ITK. It provides an API to perform operations on points, cells, boundaries, etc., but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier types that will be used to access the mesh. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

One of the most important parts of using this mesh is how to create cells to insert into it. The cells for the mesh take two template parameters. The first is the pixel type, and should correspond exactly to that type given to the mesh. The second is a "CellTraits" which holds a sub-set of the "MeshTraits" structure definitions, and is also a member of them. Any cell which is to be inserted to a mesh should have MeshTraits::CellTraits as its second template parameter.

**Usage** Mesh has three template parameters. The first is the pixel type, or the type of data stored (optionally) with points, cells, and/or boundaries. The second is the geometric dimension of the points defining the mesh. This also limits the maximum topological dimension of the cells that can be inserted. The third template parameter is the "MeshTraits" structure controlling type information for the mesh. Most users will be happy with the defaults, and will not have to worry about this third argument.

Template parameters for Mesh:

TPixelType = The type stored as data for an entity (cell, point, or boundary).

TMeshTraits = Type information structure for the mesh.

**References** No reference information is available.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Points And Edges
- Convert Mesh To Unstructered Grid
- Working With Point And Cell Data

Subclassed by itk::SimplexMesh< TPixelType, VDimension, TMeshTraits >

See itk::Mesh for additional documentation.

### Convert Triangle Mesh to Binary Image

### Synopsis

Convert a triangular itk::Mesh to binary itk::Image

### Results

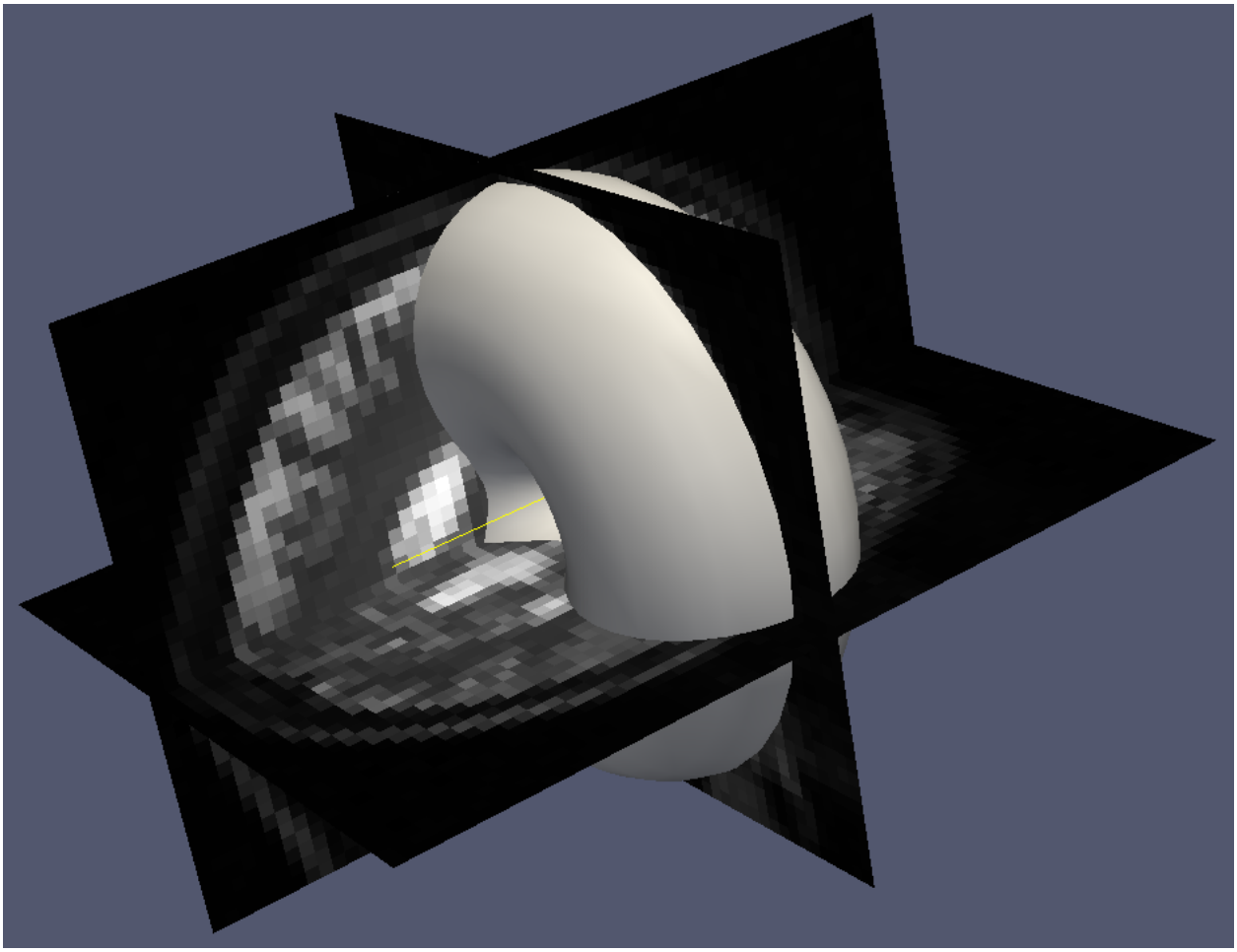


Fig. 54: Input 3D Image and Mesh

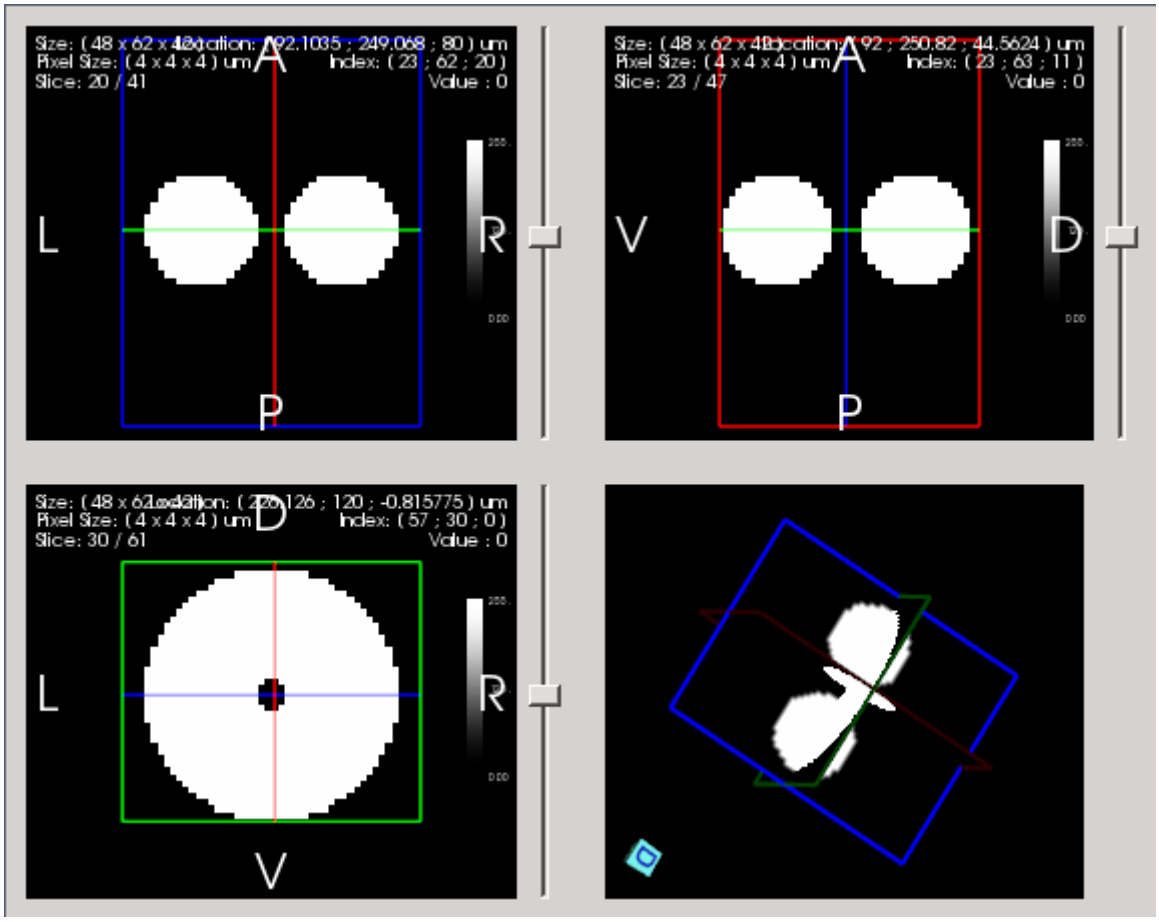



Fig. 55: Output image

### Code

### C++

```
#include "itkMesh.h"
#include "itkMeshFileReader.h"

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkCastImageFilter.h"
#include "itkTriangleMeshToBinaryImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputImageName> <InputMeshName> <OutputImageName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputImageName = argv[1];
  const char * inputMeshName = argv[2];
  const char * outputImageName = argv[3];

  constexpr unsigned int Dimension = 3;
  using MeshPixelType = double;

  using MeshType = itk::Mesh<MeshPixelType, Dimension>;

  using MeshReaderType = itk::MeshFileReader<MeshType>;
  MeshReaderType::Pointer meshReader = MeshReaderType::New();
  meshReader->SetFileName(inputMeshName);

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using ImageReaderType = itk::ImageFileReader<InputImageType>;

  ImageReaderType::Pointer imageReader = ImageReaderType::New();
  imageReader->SetFileName(inputImageName);

  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using CastFilterType = itk::CastImageFilter<InputImageType, OutputImageType>;
  CastFilterType::Pointer cast = CastFilterType::New();
  cast->SetInput(imageReader->GetOutput());

  using FilterType = itk::TriangleMeshToBinaryImageFilter<MeshType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(meshReader->GetOutput());
  filter->SetInfoImage(cast->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  filter->SetInsideValue(itk::NumericTraits<OutputPixelType>::max());
  try
  {
    filter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputImageName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputImage**>
**class TriangleMeshToBinaryImageFilter** : **public** itk::*ImageSource*<*TOutputImage*>
3D Rasterization algorithm Courtesy of Dr David Gobbi of Atamai Inc.

**Author** Leila Baghdadi, MICe, Hospital for Sick Childern, Toronto, Canada,

See itk::TriangleMeshToBinaryImageFilter for additional documentation.

### Extract Iso Surface

### Synopsis

Extract the iso surface as one itk::Mesh from a 3D itk::Image

### Results

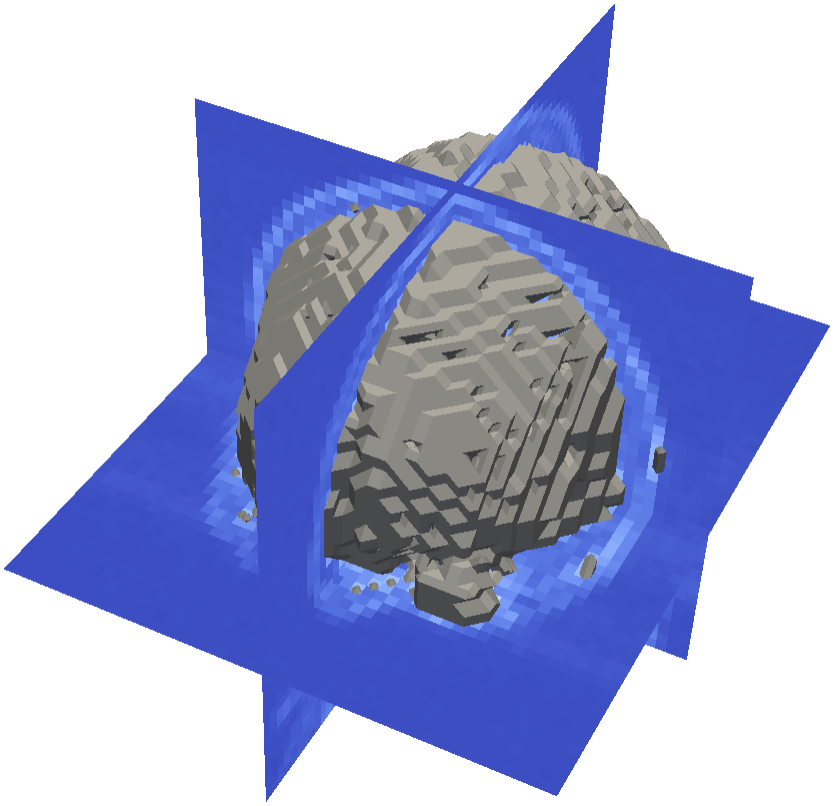


Fig. 56: Input 3D image with iso-surface mesh

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkMesh.h"
#include "itkBinaryThresholdImageFilter.h"
#include "itkBinaryMask3DMeshSource.h"
#include "itkMeshFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <Lower Threshold> <Upper␣
↪Threshold>";
```

(continues on next page)

(continued from previous page)

```
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  auto lowerThreshold = static_cast<PixelType>(std::stoi(argv[3]));
  auto upperThreshold = static_cast<PixelType>(std::stoi(argv[4]));

  using BinaryThresholdFilterType = itk::BinaryThresholdImageFilter<ImageType,␣
↪ImageType>;
  BinaryThresholdFilterType::Pointer threshold = BinaryThresholdFilterType::New();
  threshold->SetInput(reader->GetOutput());
  threshold->SetLowerThreshold(lowerThreshold);
  threshold->SetUpperThreshold(upperThreshold);
  threshold->SetOutsideValue(0);

  using MeshType = itk::Mesh<double, Dimension>;

  using FilterType = itk::BinaryMask3DMeshSource<ImageType, MeshType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(threshold->GetOutput());
  filter->SetObjectValue(255);

  using WriterType = itk::MeshFileWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputMesh**>
**class BinaryMask3DMeshSource** : **public** itk::ImageToMeshFilter<*TInputImage*, *TOutputMesh*>

This class tries to construct a 3D mesh surface based on a binary mask. It can be used to integrate a region-based segmentation method and a deformable model into one hybrid framework.

To construct a mesh, we need to construct elements in a voxel and combine those elements later to form the final mesh. Before go through every voxel in the 3D volume, we first construct 2 look up tables. The index of these 2 tables are the on-off combination of the 8 nodes that form the voxel. So both of these tables has the size of $2^8$ bytes. According to previous work, all those $2^8$ combination of the nodes can be grouped into 16 final combinations. In the first table, we record the final combination that can be transformed from the current combination. The entries of the second table are made up of the transforming sequence that is necessary for the current combination transform to one of the final combinations.

We then go through the 3D volume voxel by voxel, using those two tables we have defined to construct elements within each voxel. We then merge all these mesh elements into one 3D mesh.

**PARAMETERS** The ObjectValue parameter is used to identify the object. In most applications, pixels in the object region are assigned to "1", so the default value of ObjectValue is set to "1"

**REFERENCE** W. Lorensen and H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics 21, pp. 163-169, 1987.

**INPUT** The input should be a 3D binary image.

See itk::BinaryMask3DMeshSource for additional documentation.

### Translate One Mesh

### Synopsis

translate one itk::Mesh

### Results

### Code

### C++

```
#include "itkMesh.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"
#include "itkTranslationTransform.h"
#include "itkTransformMeshFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
```

(continues on next page)

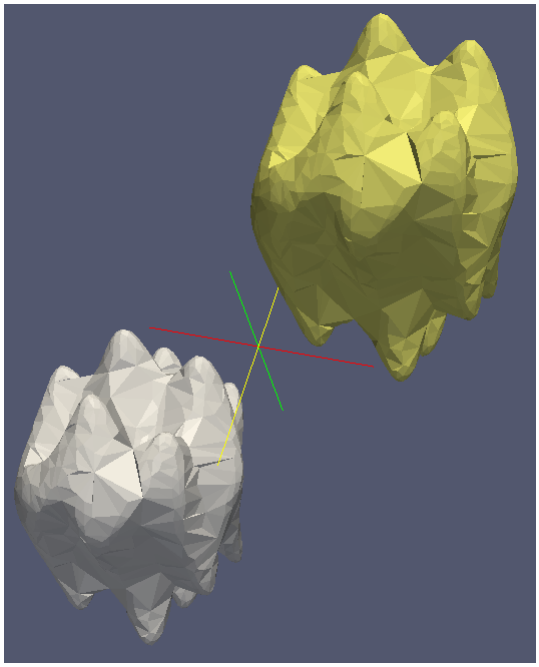


Fig. 57: Input Mesh (grey) and Output Mesh (yellow)

(continued from previous page)

```
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 3;

  using PixelType = double;
  using MeshType = itk::Mesh<PixelType, Dimension>;

  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using TransformType = itk::TranslationTransform<MeshType::PointType::CoordRepType,␣
↪Dimension>;
  TransformType::Pointer translation = TransformType::New();

  TransformType::OutputVectorType displacement;
  displacement.Fill(1.);

  translation->Translate(displacement);

  using FilterType = itk::TransformMeshFilter<MeshType, MeshType, TransformType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetTransform(translation);
```

(continues on next page)

(continued from previous page)

```
  using WriterType = itk::MeshFileWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputMesh**, typename **TTransform**>
**class TransformMeshFilter** : **public** itk::MeshToMeshFilter<*TInputMesh*, *TOutputMesh*>
TransformMeshFilter applies a transform to all the points of a mesh.

The additional content of the mesh is passed untouched. Including the connectivity and the additional information contained on cells and points.

Meshes that have added information like normal vector on the points, will have to take care of transforming this data by other means.

See itk::TransformMeshFilter for additional documentation.

### Working With Point and Cell Data

### Synopsis

Associate point and cell data with a mesh.

### Results

Output:

```
66
66
5

128
128
10
```

mesh.vtk is also created.

### Code

### C++

```
// Include the relevant header files.
#include "itkMesh.h"
#include "itkRegularSphereMeshSource.h"
#include "itkMeshFileWriter.h"

// We define the dimension and coordinate type...
constexpr unsigned int Dimension = 3;
using TCoordinate = float;

// ...and then type alias the mesh, sphere, and writer.
using TMesh = itk::Mesh<TCoordinate, Dimension>;
using TSphere = itk::RegularSphereMeshSource<TMesh>;
using TMeshWriter = itk::MeshFileWriter<TMesh>;

int
main(int, char *[])
{

  // Create the sphere source.
  TSphere::Pointer sphere = TSphere::New();
  sphere->Update();

  // We now assign it to a mesh pointer.
  TMesh::Pointer mesh = sphere->GetOutput();

  // It is necessary to disconnect the mesh from the pipeline;
  // otherwise, the point and cell data will be deallocated
  // when we call "Update()" on the writer later in the program.
  mesh->DisconnectPipeline();

  // Let's assign a value to each of the mesh's points...
  for (unsigned int i = 0; i < mesh->GetNumberOfPoints(); ++i)
    mesh->SetPointData(i, 5.0);

  // ...and assign a different value to each of the mesh's cells.
  for (unsigned int i = 0; i < mesh->GetNumberOfCells(); ++i)
    mesh->SetCellData(i, 10.0);

  // We'll print out some data about the points...
  std::cout << mesh->GetNumberOfPoints() << std::endl;                          // 66
  std::cout << mesh->GetPointData()->Size() << std::endl;                       // 66
  std::cout << mesh->GetPointData()->ElementAt(0) << std::endl << std::endl; // 5.0

  // ...and about the cells.
  std::cout << mesh->GetNumberOfCells() << std::endl;                         // 128
  std::cout << mesh->GetCellData()->Size() << std::endl;                      // 128
  std::cout << mesh->GetCellData()->ElementAt(0) << std::endl << std::endl; // 10.0

  // Finally, we'll write the data to file.  Note that the only mesh file
  // formats supported by ITK which support cell and point data are .vtk and .gii.
  TMeshWriter::Pointer meshWriter = TMeshWriter::New();
  meshWriter->SetFileName("mesh.vtk");
  meshWriter->SetInput(mesh);
```

(continues on next page)

(continued from previous page)

```
  meshWriter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixelType**, unsigned int **VDimension** = 3, typename **TMeshTraits** = DefaultStaticMeshTraits<*TPixelTyp*
**class Mesh** : **public** itk::*PointSet*<*TPixelType*, *VDimension*, *TMeshTraits*>

Implements the N-dimensional mesh structure.

Mesh is an adaptive, evolving structure. Typically points and cells are created, with the cells referring to their defining points. If additional topological information is required, then BuildCellLinks() is called and links from the points back to the cells that use them are created. This allows implicit topological information about the faces and edges of the cells to be determined. (For example, a "face" neighbor to a cell can be determined by intersection the sets of cells that use the points defining the face. This is an inherent assumption on the manifold relationship of the cells in the mesh.) In some cases, either because the mesh is non-manifold, because we wish to explicitly store information with the faces and edges of the mesh, or because performance requirements demand that boundaries are explicitly represented (the set intersection does not need to be performed); then Mesh can be further extended by adding explicit boundary assignments.

**Overview** Mesh implements the N-dimensional mesh structure for ITK. It provides an API to perform operations on points, cells, boundaries, etc., but does not tie down the underlying implementation and storage. A "MeshTraits" structure is used to define the container and identifier types that will be used to access the mesh. See DefaultStaticMeshTraits for the set of type definitions needed. All types that are defined in the "MeshTraits" structure will have duplicate type alias in the resulting mesh itself.

One of the most important parts of using this mesh is how to create cells to insert into it. The cells for the mesh take two template parameters. The first is the pixel type, and should correspond exactly to that type given to the mesh. The second is a "CellTraits" which holds a sub-set of the "MeshTraits" structure definitions, and is also a member of them. Any cell which is to be inserted to a mesh should have MeshTraits::CellTraits as its second template parameter.

**Usage** Mesh has three template parameters. The first is the pixel type, or the type of data stored (optionally) with points, cells, and/or boundaries. The second is the geometric dimension of the points defining the mesh. This also limits the maximum topological dimension of the cells that can be inserted. The third template parameter is the "MeshTraits" structure controlling type information for the mesh. Most users will be happy with the defaults, and will not have to worry about this third argument.

Template parameters for Mesh:

TPixelType = The type stored as data for an entity (cell, point, or boundary).

TMeshTraits = Type information structure for the mesh.

**References** No reference information is available.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Points And Edges
- Convert Mesh To Unstructered Grid
- Working With Point And Cell Data

Subclassed by itk::SimplexMesh< TPixelType, VDimension, TMeshTraits >

See itk::Mesh for additional documentation.

### Write Mesh to VTP

### Synopsis

Write an itk::Mesh to a vtp (vtkPolyData) file.

### Results

**Warning: Fix Errors** Example contains errors needed to be fixed for proper output.

Output:

```
Points = 4
[-1, -1, 0]
[1, -1, 0]
[1, 1, 0]
[1, 1, 1]
```

### Code

### C++

```
#include "itkMesh.h"
#include "itkLineCell.h"
#include "itkVTKPolyDataWriter.h"

constexpr unsigned int Dimension = 3;
using MeshType = itk::Mesh<float, Dimension>;

MeshType::Pointer
CreateMeshWithEdges();

int
main(int, char *[])
{

  MeshType::Pointer mesh = CreateMeshWithEdges();

  using WriterType = itk::VTKPolyDataWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(mesh);
  writer->SetFileName("test.vtk");
  writer->Update();

  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
MeshType::Pointer
CreateMeshWithEdges()
{

  MeshType::Pointer mesh = MeshType::New();

  // Create points
  MeshType::PointType p0, p1, p2, p3;

  p0[0] = -1.0;
  p0[1] = -1.0;
  p0[2] = 0.0; // first  point ( -1, -1, 0 )
  p1[0] = 1.0;
  p1[1] = -1.0;
  p1[2] = 0.0; // second point (  1, -1, 0 )
  p2[0] = 1.0;
  p2[1] = 1.0;
  p2[2] = 0.0; // third  point (  1,  1, 0 )
  p3[0] = 1.0;
  p3[1] = 1.0;
  p3[2] = 1.0; // third  point (  1,  1, 1 )

  mesh->SetPoint(0, p0);
  mesh->SetPoint(1, p1);
  mesh->SetPoint(2, p2);
  mesh->SetPoint(3, p3);

  std::cout << "Points = " << mesh->GetNumberOfPoints() << std::endl;

  // access points
  using PointsIterator = MeshType::PointsContainer::Iterator;

  PointsIterator pointIterator = mesh->GetPoints()->Begin();

  PointsIterator end = mesh->GetPoints()->End();
  while (pointIterator != end)
  {
    MeshType::PointType p = pointIterator.Value(); // access the point
    std::cout << p << std::endl;                   // print the point
    ++pointIterator;                               // advance to next point
  }

  using CellAutoPointer = MeshType::CellType::CellAutoPointer;
  using LineType = itk::LineCell<MeshType::CellType>;


  CellAutoPointer line0;
  line0.TakeOwnership(new LineType);
  line0->SetPointId(0, 0); // line between points 0 and 1
  line0->SetPointId(1, 1);
  mesh->SetCell(0, line0);

  CellAutoPointer line1;
  line1.TakeOwnership(new LineType);
  line1->SetPointId(0, 1); // line between points 1 and 2
```

(continues on next page)

(continued from previous page)

```
  line1->SetPointId(1, 2);
  mesh->SetCell(1, line1);

  CellAutoPointer line2;
  line2.TakeOwnership(new LineType);
  line2->SetPointId(0, 2); // line between points 2 and 3
  line2->SetPointId(1, 3);
  mesh->SetCell(2, line2);

  return mesh;
}
```

#### Classes demonstrated

template<typename **TInputMesh**>
**class VTKPolyDataWriter** : **public** itk::*Object*

Writes an itkMesh to a file in VTK file format.

Caveat: The input to itkVTKPolyDataWriter must be a triangle mesh. Use vtkTriangleFilter to convert your mesh to a triangle mesh.

This class may be deprecated in the future. The MeshFileWriter is preferred.

**See** MeshFileWriter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Write Mesh To VTP

See itk::VTKPolyDataWriter for additional documentation.

### 3.2.5 QuadEdgeMesh

#### Create Triangular Quad Edge Mesh

#### Synopsis

Create a triangular surface mesh using itk::QuadEdgeMesh

#### Results

#### Code

#### C++

```
#include "itkQuadEdgeMesh.h"
#include "itkMeshFileWriter.h"
```

(continues on next page)

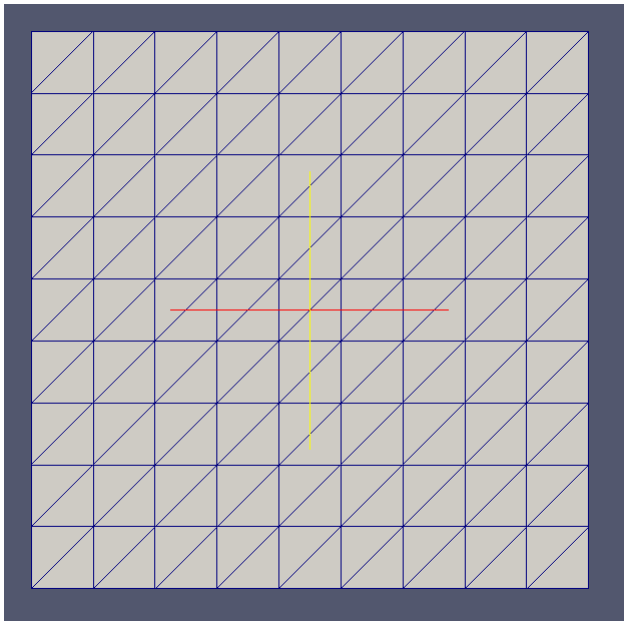


Fig. 58: Output mesh

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * outputFileName = argv[1];

  constexpr unsigned int Dimension = 3;

  using CoordType = double;
  using MeshType = itk::QuadEdgeMesh<CoordType, Dimension>;

  MeshType::Pointer mesh = MeshType::New();

  using PointsContainer = MeshType::PointsContainer;
  using PointsContainerPointer = MeshType::PointsContainerPointer;

  PointsContainerPointer points = PointsContainer::New();
  points->Reserve(100);

  using PointType = MeshType::PointType;
  PointType p;
  p[2] = 0.;
```

(continues on next page)

(continued from previous page)

```
  using PointIdentifier = MeshType::PointIdentifier;
  PointIdentifier k = 0;

  for (int i = 0; i < 10; i++)
  {
    p[0] = static_cast<CoordType>(i);

    for (int j = 0; j < 10; j++)
    {
      p[1] = static_cast<CoordType>(j);
      points->SetElement(k, p);
      k++;
    }
  }

  mesh->SetPoints(points);

  k = 0;

  for (int i = 0; i < 9; i++)
  {
    for (int j = 0; j < 9; j++)
    {
      mesh->AddFaceTriangle(k, k + 1, k + 11);
      mesh->AddFaceTriangle(k, k + 11, k + 10);
      k++;
    }
    k++;
  }

  using WriterType = itk::MeshFileWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(mesh);
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension**, typename **TTraits** = QuadEdgeMeshTraits<*TPixel*, *VDimension*, bool, bo
**class QuadEdgeMesh** : **public** itk::*Mesh*<*TPixel*, *VDimension*, *TTraits*>
Mesh class for 2D manifolds embedded in ND space.

This implementation was contributed as a paper to the Insight Journal https://www.insight-journal.org/browse/publication/122

**Author** Alexandre Gouaillard, Leonardo Florez-Valencia, Eric Boix

See itk::QuadEdgeMesh for additional documentation.

### Cut Mesh

### Synopsis

Given a bounding box and a mesh, first retrieve all vertices whose coordinates are in the bounding box; then retrieve all faces connected to these vertices.

### Results

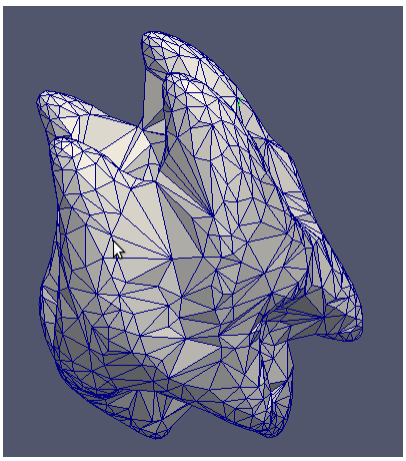


Fig. 59: Input mesh

### Code

### C++

```
#include "itkQuadEdgeMesh.h"
#include "itkQuadEdgeMeshPolygonCell.h"

#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"

int
```

(continues on next page)

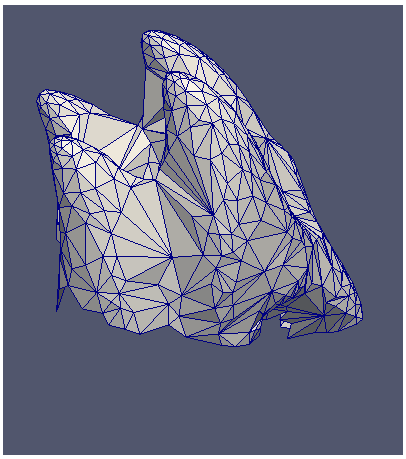


Fig. 60: Output mesh (cut)

(continued from previous page)

```
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 3;
  using CoordType = double;
  using MeshType = itk::QuadEdgeMesh<CoordType, Dimension>;
  using MeshPointer = MeshType::Pointer;
  using MeshPointsContainerPointer = MeshType::PointsContainerPointer;
  using MeshPointsContainerIterator = MeshType::PointsContainerIterator;
  using MeshPointType = MeshType::PointType;
  using MeshPointIdentifier = MeshType::PointIdentifier;
  using MeshCellIdentifier = MeshType::CellIdentifier;

  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  MeshPointer mesh = reader->GetOutput();

  MeshPointer output = MeshType::New();

  std::map<MeshPointIdentifier, MeshPointIdentifier> verticesMap;
  std::set<MeshCellIdentifier>                       facesSet;

  MeshPointsContainerPointer points = mesh->GetPoints();
```

(continues on next page)

(continued from previous page)

```
  MeshPointsContainerIterator pIt = points->Begin();
  MeshPointsContainerIterator pEnd = points->End();

  while (pIt != pEnd)
  {
    MeshPointType p = pIt->Value();

    if ((p[2] < 0.))
    {
      // here you do not want to use operator = to copy the coordinates from the
      // input mesh to the output one. Indeed the operator = does not only copy
      // the coordinates, but also the pointer to first QuadEdge of the 0-ring,
      // and it would be wrong.
      MeshPointType q;
      q.CastFrom(p);

      verticesMap[pIt->Index()] = output->AddPoint(q);

      // iterate on the 0-ring (vertex neighbors)
      MeshType::QEType * qe = p.GetEdge();
      MeshType::QEType * temp = qe;
      do
      {
        // insert the corresponding faces into std::set
        facesSet.insert(temp->GetLeft());

        temp = temp->GetOnext();
      } while (qe != temp);
    }
    ++pIt;
  }

  MeshType::CellsContainerPointer cells = mesh->GetCells();

  using PolygonType = itk::QuadEdgeMeshPolygonCell<MeshType::CellType>;

  // iterate on the faces to be added into resulting mesh
  for (auto fIt : facesSet)
  {
    auto *                    face = dynamic_cast<PolygonType *>(cells->
↪ElementAt(fIt));
    MeshType::PointIdentifier id[3];

    if (face)
    {
      MeshType::QEType * qe = face->GetEdgeRingEntry();
      MeshType::QEType * temp = qe;

      unsigned int k = 0;

      // iterate on the vertices of a given face
      do
      {
        // add the corresponding vertex into the output mesh if it has not been added␣
↪yet
        if (verticesMap.find(temp->GetOrigin()) == verticesMap.end())
        {
```

(continues on next page)

(continued from previous page)

```
          MeshType::PointType p = mesh->GetPoint(temp->GetOrigin());

          MeshType::PointType q;
          q.CastFrom(p);

          verticesMap[temp->GetOrigin()] = output->AddPoint(q);
        }
        id[k++] = verticesMap[temp->GetOrigin()];

        temp = temp->GetLnext();
      } while (qe != temp);

      // add the corresponding face into the output mesh
      output->AddFaceTriangle(id[0], id[1], id[2]);
    }
  }

  // save the corresponding mesh
  using MeshWriterType = itk::MeshFileWriter<MeshType>;
  MeshWriterType::Pointer writer = MeshWriterType::New();
  writer->SetInput(output);
  writer->SetFileName(outputFileName);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension**, typename **TTraits** = QuadEdgeMeshTraits<*TPixel*, *VDimension*, bool, bo
**class QuadEdgeMesh** : **public** itk::*Mesh*<*TPixel*, *VDimension*, *TTraits*>

Mesh class for 2D manifolds embedded in ND space.

This implementation was contributed as a paper to the Insight Journal https://www.insight-journal.org/browse/publication/122

**Author** Alexandre Gouaillard, Leonardo Florez-Valencia, Eric Boix

See itk::QuadEdgeMesh for additional documentation.

### Extract Vertex on Mesh Boundaries

### Synopsis

Print the vertices in order which lie on each boundary of a given mesh.

### Results

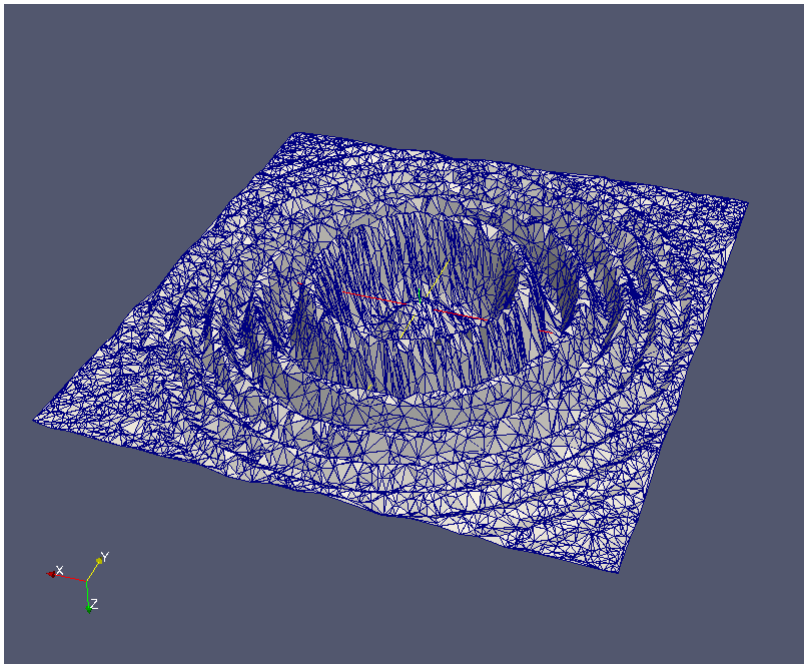



Fig. 61: Input mesh

Example output:

```
There are 1 borders on this mesh
0: 684 -> 3848 -> 955 -> 1643 -> 4340 -> 178 -> 1160 -> 6650 -> 3539 -> 523 ->
6052 -> 9 -> 7619 -> 2876 -> 4010 -> 102 -> 1451 -> 557 -> 5658 -> 3326 -> 4606
-> 1154 -> 4584 -> 5553 -> 6739 -> 1725 -> 7207 -> 3978 -> 3116 -> 7770 -> 2293
-> 3285 -> 2215 -> 6505 -> 7128 -> 7973 -> 6479 -> 3292 -> 3719 -> 3111 -> 6566
-> 1227 -> 3410 -> 3418 -> 7208 -> 5732 -> 7203 -> 645 -> 314 -> 7872 -> 6941
-> 5076 -> 1965 -> 2017 -> 3235 -> 3801 -> 4754 -> 1348 -> 2390 -> 7367 -> 6319
-> 5458 -> 7572 -> 151 -> 4095 -> 3873 -> 7336 -> 7260 -> 2112 -> 6373 -> 1664
-> 7247 -> 7661 -> 5790 -> 1698 -> 572 -> 5783 -> 3042 -> 5259 -> 7802 -> 6192
-> 894 -> 4545 -> 1298 -> 684
```

### Code

### C++

```
#include "itkQuadEdgeMesh.h"
#include "itkVTKPolyDataReader.h"
#include "itkQuadEdgeMeshBoundaryEdgesMeshFunction.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << argv[0] << " <InputFileName>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using MeshType = itk::QuadEdgeMesh<CoordType, Dimension>;
  using VTKReaderType = itk::VTKPolyDataReader<MeshType>;

  VTKReaderType::Pointer reader = VTKReaderType::New();
  reader->SetFileName(argv[1]);

  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << e.what() << std::endl;
    return EXIT_FAILURE;
  }

  MeshType::Pointer mesh = reader->GetOutput();

  using BoundaryExtractorType = itk::QuadEdgeMeshBoundaryEdgesMeshFunction<MeshType>;

  BoundaryExtractorType::Pointer extractor = BoundaryExtractorType::New();

  using MeshPointIdentifier = MeshType::PointIdentifier;

  using MeshQEType = MeshType::QEType;
  using MeshIteratorGeom = MeshQEType::IteratorGeom;

  using EdgeListType = MeshType::EdgeListType;
  using EdgeListPointer = MeshType::EdgeListPointerType;
  using EdgeListIterator = EdgeListType::iterator;

  EdgeListPointer list = extractor->Evaluate(*mesh);

  if (list->empty())
  {
    std::cerr << "There is no border on this mesh" << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }

  std::cout << "There are " << list->size() << " borders on this mesh" << std::endl;

  auto                   it = list->begin();
  const EdgeListIterator end = list->end();

  size_t i = 0;
  while (it != end)
  {
    std::cout << i << ": ";

    MeshIteratorGeom       eIt = (*it)->BeginGeomLnext();
    const MeshIteratorGeom eEnd = (*it)->EndGeomLnext();

    MeshPointIdentifier id = MeshType::m_NoPoint;

    while (eIt != eEnd)
    {
      MeshQEType * qe = eIt.Value();

      if (qe->GetOrigin() != id)
      {
        std::cout << qe->GetOrigin();
      }

      id = qe->GetDestination();

      std::cout << " -> " << id;
      ++eIt;
    }

    std::cout << std::endl;

    ++it;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TMesh**>
**class QuadEdgeMeshBoundaryEdgesMeshFunction** : **public** itk::FunctionBase<*TMesh*, *TMesh*::EdgeListPointerType>
Build a list of references to edges (as GeometricalQuadEdge::RawPointer) each one representing a different boundary component.

**Note** Each resulting edge has the surface on its right and is hence ready for a walk on with the help of BeginGeomLnext().

**Note** The size() of the resulting list is the number of boundary components.

See itk::QuadEdgeMeshBoundaryEdgesMeshFunction for additional documentation.

### Get List of Faces Around a Given Vertex

### Synopsis

Get the list of faces around a given vertex.

### Results

Results:

```
79
1961
2036
1960
1
```

These faces are visualized in the figures below.

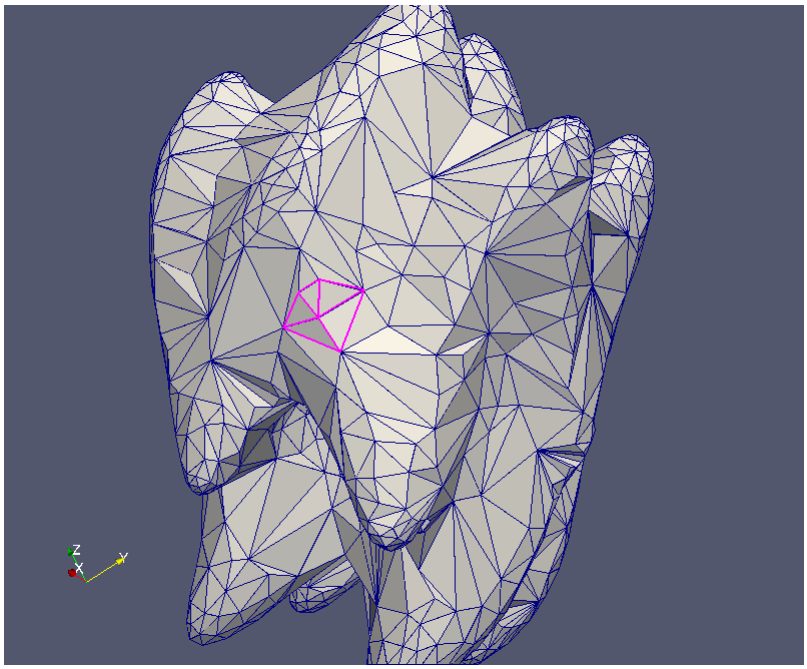


Fig. 62: The selected faces around the given vertex are highlighted in pink.

### Code

### C++

```
#include "itkMeshFileReader.h"
#include "itkQuadEdgeMesh.h"

int
main(int argc, char * argv[])
```

(continues on next page)

(continued from previous page)

```
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <VertexId>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 3;

  using PixelType = double;
  using MeshType = itk::QuadEdgeMesh<PixelType, Dimension>;

  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << e.what() << std::endl;
    return EXIT_FAILURE;
  }

  MeshType::Pointer mesh = reader->GetOutput();

  MeshType::PointIdentifier id = std::stoi(argv[2]);

  MeshType::QEType * qe = mesh->FindEdge(id);

  MeshType::QEType * temp = qe;
  do
  {
    std::cout << temp->GetLeft() << std::endl;
    temp = temp->GetOnext();
  } while (qe != temp);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension**, typename **TTraits** = QuadEdgeMeshTraits<*TPixel*, *VDimension*, bool, bo
**class QuadEdgeMesh** : **public** itk::*Mesh*<*TPixel*, *VDimension*, *TTraits*>

Mesh class for 2D manifolds embedded in ND space.

This implementation was contributed as a paper to the Insight Journal https://www.insight-journal.org/browse/publication/122

**Author** Alexandre Gouaillard, Leonardo Florez-Valencia, Eric Boix

See itk::QuadEdgeMesh for additional documentation.

### Print Vertex Neighbors

### Synopsis

Print the neighbors of a given vertex.

### Results

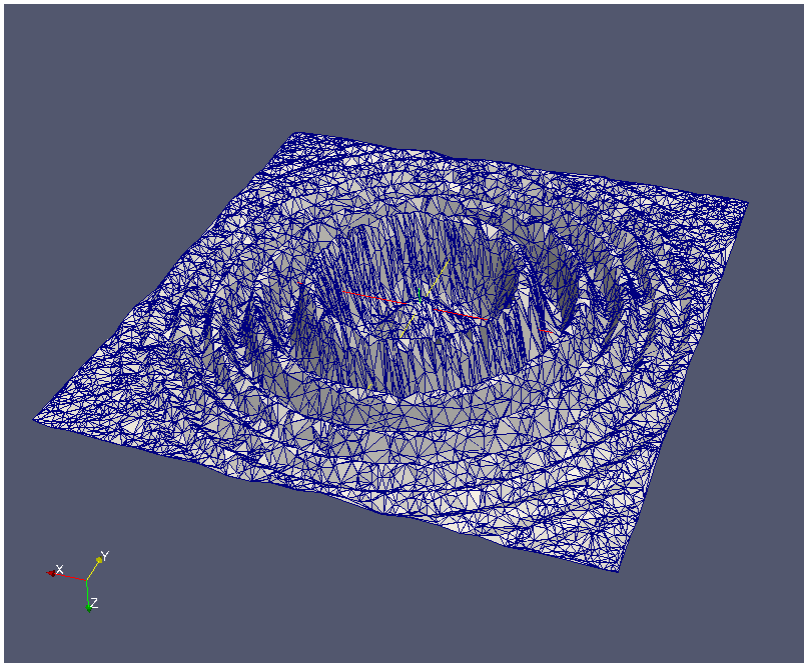



Fig. 63: Input mesh

Example output:

```
3435
6999
5422
2869
244
584
```

### Code

### C++

```
#include "itkMeshFileReader.h"
#include "itkQuadEdgeMesh.h"

int
main(int argc, char * argv[])
{
```

(continues on next page)

(continued from previous page)

```
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <VertexId>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 3;
  using CoordinateType = double;
  using MeshType = itk::QuadEdgeMesh<CoordinateType, Dimension>;
  using ReaderType = itk::MeshFileReader<MeshType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << e.what() << std::endl;
    return EXIT_FAILURE;
  }

  MeshType::Pointer mesh = reader->GetOutput();

  auto id = static_cast<MeshType::PointIdentifier>(std::stoi(argv[2]));

  MeshType::QEType * qe = mesh->FindEdge(id);
  if (qe == nullptr)
  {
    std::cerr << "Error: either this vertex does not exist, either this vertex is not␣
↪connected." << std::endl;
    return EXIT_FAILURE;
  }

  MeshType::QEType * qe2 = qe;

  do
  {
    std::cout << qe->GetDestination() << std::endl;
    qe = qe->GetOnext();
  } while (qe2 != qe);

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension**, typename **TTraits** = QuadEdgeMeshTraits<*TPixel*, *VDimension*, bool, bo
**class QuadEdgeMesh** : **public** itk::*Mesh*<*TPixel*, *VDimension*, *TTraits*>
Mesh class for 2D manifolds embedded in ND space.

This implementation was contributed as a paper to the Insight Journal https://www.insight-journal.org/browse/publication/122

**Author** Alexandre Gouaillard, Leonardo Florez-Valencia, Eric Boix

See itk::QuadEdgeMesh for additional documentation.

### 3.2.6 SpatialObjects

#### Blob

#### Synopsis

Blob.

#### Results

Output:

```
Bounds: [0, 0, 0, 0]
```

#### Code

#### C++

```
#include "itkBlobSpatialObject.h"

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  using BlobType = itk::BlobSpatialObject<2>;

  // Create a list of points
  BlobType::BlobPointListType points;
  for (unsigned int i = 0; i < 20; i++)
  {
    BlobType::BlobPointType point;
    point.SetPositionInObjectSpace(i, i);

    points.push_back(point);
  }

  BlobType::Pointer blob = BlobType::New();
  blob->SetPoints(points);

  BlobType::BoundingBoxType::BoundsArrayType bounds = blob->
↪GetMyBoundingBoxInWorldSpace()->GetBounds();
```

(continues on next page)

(continued from previous page)

```
  std::cout << "Bounds: " << bounds << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **TDimension** = 3>
**class BlobSpatialObject** : **public** itk::PointBasedSpatialObject<*TDimension*, SpatialObjectPoint<*TDimension*>>
Spatial object representing a potentially amorphous object.

The BlobSpatialObject is a discretized representation of a "blob", which can be taken to be an arbitrary, possibly amorphous shape. The representation is a list of the points (voxel centers) contained in the object. This can be thought of as an alternate way to represent a binary image.

**See** SpatialObjectPoint

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Blob

See itk::BlobSpatialObject for additional documentation.

### Contour Spatial Object

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Contour spatial object.

### Results

**Results Coming Soon!**

### Code

### C++

```
#include "itkSpatialObjectToImageFilter.h"
#include "itkContourSpatialObject.h"
#include "itkContourSpatialObjectPoint.h"
#include "itkImageFileWriter.h"
```

(continues on next page)

(continued from previous page)

```
#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int /*argc*/, char * /*argv*/[])
{
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;

  using ContourType = itk::ContourSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<ContourType,␣
↪ImageType>;


  // Create a list of points
  ContourType::ControlPointListType points;

  // Add some points
  ContourType::ControlPointType point;
  point.SetPositionInObjectSpace(0, 0);
  points.push_back(point);
  point.SetPositionInObjectSpace(0, 30);
  points.push_back(point);
  point.SetPositionInObjectSpace(30, 30);
  points.push_back(point);
  point.SetPositionInObjectSpace(0, 0);
  points.push_back(point);

  // Create a contour from the list of points
  ContourType::Pointer contour = ContourType::New();
  contour->SetControlPoints(points);

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();
  itk::Size<2>                            size;
  size.Fill(50);
  imageFilter->SetInsideValue(255); // white
  imageFilter->SetSize(size);
  imageFilter->SetInput(contour);
  imageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("contour.png");
  writer->SetInput(imageFilter->GetOutput());
  writer->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(imageFilter->GetOutput());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

template<unsigned int **TDimension** = 3>
**class ContourSpatialObject** : **public** itk::PointBasedSpatialObject<*TDimension*, ContourSpatialObjectPoint<*TDimension*

Representation of a Contour based on the spatial object classes.

The Contour is basically defined by a set of points which are inside this blob

**See** SpatialObjectPoint

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

\sphinxexample{Core/SpatialObjects/{{ ContourSpatialObject,Contour Spacial Object} \endsphinx

See itk::ContourSpatialObject for additional documentation.

### Convert Spatial Object to Image

### Synopsis

Convert a spatial object to an image.

### Results

Fig. 64: output.png

### Code

### C++

```
#include "itkSpatialObjectToImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << argv[0] << " outputimagefile " << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;

  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  //  The SpatialObjectToImageFilter requires that the user defines the grid
  //  parameters of the output image. This includes the number of pixels along
  //  each dimension, the pixel spacing, image direction and

  ImageType::SizeType size;
  size[0] = 50;
  size[1] = 50;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing[0] = 100.0 / size[0];
  spacing[1] = 100.0 / size[1];

  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 20;
  // ellipse->SetRadiusInObjectSpace(  size[0] * 0.2 * spacing[0] );
  ellipse->SetRadiusInObjectSpace(radiusArray);

  // Position the ellipse

  using TransformType = EllipseType::TransformType;

  TransformType::Pointer transform = TransformType::New();

  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = size[0] * spacing[0] / 2.0;
  translation[1] = size[1] * spacing[1] / 4.0;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);

  ellipse->SetDefaultOutsideValue(0);
```

(continues on next page)

(continued from previous page)

```
  imageFilter->SetUseObjectValue(true);

  imageFilter->SetOutsideValue(0);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName(argv[1]);
  writer->SetInput(imageFilter->GetOutput());

  try
  {
    imageFilter->Update();
    writer->Update();
  }
  catch (itk::ExceptionObject & excp)
  {
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputSpatialObject**, typename **TOutputImage**>
**class SpatialObjectToImageFilter** : **public** itk::*ImageSource<TOutputImage>*

Base class for filters that take a SpatialObject as input and produce an image as output. By default, if the user does not specify the size of the output image, the maximum size of the object's bounding box is used. The spacing of the image is given by the spacing of the input Spatial object.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert Spatial Object To Image

See itk::SpatialObjectToImageFilter for additional documentation.

### Create a Line Spatial Object

### Synopsis

Specify a piecewise-linear object by specifying points along the line.

### Results

Fig. 65: line.png

### Code

### C++

```
#include "itkSpatialObjectToImageFilter.h"
#include "itkLineSpatialObject.h"
#include "itkLineSpatialObjectPoint.h"
#include "itkImageFileWriter.h"

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;

  using LineType = itk::LineSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<LineType,␣
↪ImageType>;


  // Create a list of points
  std::vector<LineType::LinePointType> points;
  for (unsigned int i = 0; i < 20; i++)
  {
    LineType::LinePointType point;
    point.SetPositionInObjectSpace(10, i);

    LineType::LinePointType::CovariantVectorType normal;
    normal[0] = 0;
    normal[1] = 1;
    point.SetNormalInObjectSpace(normal, 0);
    points.push_back(point);
  }

  // Create a line from the list of points
  LineType::Pointer line = LineType::New();
  line->SetPoints(points);

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();
  itk::Size<2>                            size;
  size.Fill(50);
  imageFilter->SetInsideValue(255); // white
  imageFilter->SetSize(size);
```

(continues on next page)

(continued from previous page)

```
  imageFilter->SetInput(line);
  imageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("line.png");
  writer->SetInput(imageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **TDimension** = 3>
**class LineSpatialObject** : **public** itk::PointBasedSpatialObject<*TDimension*, *LineSpatialObjectPoint*<*TDimension*>>
Representation of a Line based on the spatial object classes.

The Line is basically defined by a set of points.

**See** LineSpatialObjectPoint

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Line Spatial Object

See itk::LineSpatialObject for additional documentation.

template<unsigned int **TPointDimension** = 3>
**class LineSpatialObjectPoint** : **public** itk::SpatialObjectPoint<*TPointDimension*>
Point used for a line definition.

This class contains all the functions necessary to define a point that can be used to build lines. This Class derives from SpatialObjectPoint. A LineSpatialObjectPoint has NDimension-1 normals.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Line Spatial Object

See itk::LineSpatialObjectPoint for additional documentation.

### Ellipse

### Synopsis

Create ellipse.

### Results

Fig. 66: Output.png

### Code

### C++

```
#include "itkSpatialObjectToImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << argv[0] << " outputimagefile " << std::endl;
    return EXIT_FAILURE;
  }

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;

  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  //  The SpatialObjectToImageFilter requires that the user defines the grid
  //  parameters of the output image. This includes the number of pixels along
  //  each dimension, the pixel spacing, image direction and

  ImageType::SizeType size;
  size[0] = 50;
  size[1] = 50;

  imageFilter->SetSize(size);
```

(continues on next page)

(continued from previous page)

```
ImageType::SpacingType spacing;
spacing[0] = 100.0 / size[0];
spacing[1] = 100.0 / size[1];

imageFilter->SetSpacing(spacing);

EllipseType::Pointer   ellipse = EllipseType::New();
EllipseType::ArrayType radiusArray;
radiusArray[0] = 10;
radiusArray[1] = 20;
// ellipse->SetRadiusInObjectSpace(  size[0] * 0.2 * spacing[0] );
ellipse->SetRadiusInObjectSpace(radiusArray);

// Position the ellipse

using TransformType = EllipseType::TransformType;

TransformType::Pointer transform = TransformType::New();

transform->SetIdentity();

TransformType::OutputVectorType translation;
translation[0] = size[0] * spacing[0] / 2.0;
translation[1] = size[1] * spacing[1] / 4.0;
transform->Translate(translation, false);

ellipse->SetObjectToParentTransform(transform);

imageFilter->SetInput(ellipse);

ellipse->SetDefaultInsideValue(255);

ellipse->SetDefaultOutsideValue(0);

imageFilter->SetUseObjectValue(true);

imageFilter->SetOutsideValue(0);

using WriterType = itk::ImageFileWriter<ImageType>;
WriterType::Pointer writer = WriterType::New();

writer->SetFileName(argv[1]);
writer->SetInput(imageFilter->GetOutput());

try
{
  imageFilter->Update();
  writer->Update();
}
catch (itk::ExceptionObject & excp)
{
  std::cerr << excp << std::endl;
  return EXIT_FAILURE;
}

return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

#### Classes demonstrated

template<unsigned int **TDimension** = 3>
**class EllipseSpatialObject** : **public** itk::SpatialObject<*TDimension*>

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Ellipse

See itk::EllipseSpatialObject for additional documentation.

### 3.2.7 TestKernel

#### Generate Random Image

#### Synopsis

This example produces an image with random pixel values.

#### Results

Fig. 67: Output image

#### Code

#### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << " <OutputFileName>" << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }
  const char * outputFileName = argv[1];

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::SizeType size;
  size.Fill(10);

  using RandomImageSourceType = itk::RandomImageSource<ImageType>;

  RandomImageSourceType::Pointer randomImageSource = RandomImageSourceType::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce reproducible results
  randomImageSource->SetSize(size);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(randomImageSource->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**>
**class RandomImageSource** : **public** itk::*ImageSource<TOutputImage>*

Generate an n-dimensional image of random pixel values.

RandomImageSource generates an image of random pixel values. This filter uses an inline random number generator since the library drand48, although thread-safe, is very slow in a threaded environment. The output image may be of any dimension. NOTE: To produce deterministic results, set the number of threads to 1.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Generate Random Image

See itk::RandomImageSource for additional documentation.

### 3.2.8 Transform

#### Apply Affine Transform From Homogeneous Matrix and Resample

#### Synopsis

Given one homogeneous matrix (here a 3x3 matrix), apply the corresponding transform to a given image and resample using a WindowedSincInterpolateImageFunction.

#### Results

Fig. 68: Input image

Fig. 69: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkAffineTransform.h"
#include "itkResampleImageFilter.h"
#include "itkWindowedSincInterpolateImageFunction.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <DefaultPixelValue>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using ScalarType = double;

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const auto   defaultValue = static_cast<ScalarType>(std::stod(argv[3]));

  using MatrixType = itk::Matrix<ScalarType, Dimension + 1, Dimension + 1>;
  MatrixType matrix;
  matrix[0][0] = std::cos(0.05);
  matrix[0][1] = std::sin(0.05);
  matrix[0][2] = 0.;

  matrix[1][0] = -matrix[0][1];
  matrix[1][1] = matrix[0][0];
  matrix[1][2] = 0.;

  matrix[2][0] = -10.;
  matrix[2][1] = -20.;
  matrix[2][2] = 1.;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  const auto input = itk::ReadImage<ImageType>(inputFileName);

  const ImageType::SizeType & size = input->GetLargestPossibleRegion().GetSize();

  using ResampleImageFilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  ResampleImageFilterType::Pointer resample = ResampleImageFilterType::New();
  resample->SetInput(input);
  resample->SetReferenceImage(input);
  resample->UseReferenceImageOn();
  resample->SetSize(size);
  resample->SetDefaultPixelValue(defaultValue);
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Radius = 3;
  using InterpolatorType = itk::WindowedSincInterpolateImageFunction<ImageType,␣
↪Radius>;
  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  resample->SetInterpolator(interpolator);

  using TransformType = itk::AffineTransform<ScalarType, Dimension>;
  TransformType::Pointer transform = TransformType::New();

  // get transform parameters from MatrixType
  TransformType::ParametersType parameters(Dimension * Dimension + Dimension);
  for (unsigned int i = 0; i < Dimension; i++)
  {
    for (unsigned int j = 0; j < Dimension; j++)
    {
      parameters[i * Dimension + j] = matrix[i][j];
    }
  }
  for (unsigned int i = 0; i < Dimension; i++)
  {
    parameters[i + Dimension * Dimension] = matrix[i][Dimension];
  }
  transform->SetParameters(parameters);

  resample->SetTransform(transform);

  try
  {
    itk::WriteImage(resample->GetOutput(), outputFileName);
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class AffineTransform** : **public** itk::MatrixOffsetTransformBase<*TParametersValueType*, *NDimensions*, *NDimensions*>
Affine transformation of a vector space (e.g. space coordinates)

This class allows the definition and manipulation of affine transformations of an n-dimensional affine space (and its associated vector space) onto itself. One common use is to define and manipulate Euclidean coordinate transformations in two and three dimensions, but other uses are possible as well.

An affine transformation is defined mathematically as a linear transformation plus a constant offset. If A is a constant n x n matrix and b is a constant n-vector, then y = Ax+b defines an affine transformation from the n-vector x to the n-vector y.

The difference between two points is a vector and transforms linearly, using the matrix only. That is, (y1-y2) =

A*(x1-x2).

The AffineTransform class determines whether to transform an object as a point or a vector by examining its type. An object of type Point transforms as a point; an object of type Vector transforms as a vector.

One common use of affine transformations is to define coordinate conversions in two- and three-dimensional space. In this application, x is a two- or three-dimensional vector containing the "source" coordinates of a point, y is a vector containing the "target" coordinates, the matrix A defines the scaling and rotation of the coordinate systems from the source to the target, and b defines the translation of the origin from the source to the target. More generally, A can also define anisotropic scaling and shearing transformations. Any good textbook on computer graphics will discuss coordinate transformations in more detail. Several of the methods in this class are designed for this purpose and use the language appropriate to coordinate conversions.

Any two affine transformations may be composed and the result is another affine transformation. However, the order is important. Given two affine transformations T1 and T2, we will say that "precomposing T1 with T2" yields the transformation which applies T1 to the source, and then applies T2 to that result to obtain the target. Conversely, we will say that "postcomposing T1 with T2" yields the transformation which applies T2 to the source, and then applies T1 to that result to obtain the target. (Whether T1 or T2 comes first lexicographically depends on whether you choose to write mappings from right-to-left or vice versa; we avoid the whole problem by referring to the order of application rather than the textual order.)

There are two template parameters for this class:

TParametersValueType The type to be used for scalar numeric values. Either float or double.

NDimensions The number of dimensions of the vector space.

This class provides several methods for setting the matrix and vector defining the transform. To support the registration framework, the transform parameters can also be set as an Array<double> of size (NDimension + 1) * NDimension using method SetParameters(). The first (NDimension x NDimension) parameters defines the matrix in row-major order (where the column index varies the fastest). The last NDimension parameters defines the translation in each dimensions.

This class also supports the specification of a center of rotation (center) and a translation that is applied with respect to that centered rotation. By default the center of rotation is set to the origin.

Subclassed by itk::AzimuthElevationToCartesianTransform< TParametersValueType, NDimensions >, itk::CenteredAffineTransform< TParametersValueType, NDimensions >, itk::ScalableAffineTransform< TParametersValueType, NDimensions >

See itk::AffineTransform for additional documentation.

template<typename **TInputImage**, unsigned int **VRadius**, typename **TWindowFunction** = Function::HammingWindowFunction
**class WindowedSincInterpolateImageFunction** : **public** itk::InterpolateImageFunction<*TInputImage*, *TCoordRep*>

Use the windowed sinc function to interpolate.

This function is intended to provide an interpolation function that has minimum aliasing artifacts, in contrast to linear interpolation. According to sampling theory, the infinite-support sinc filter, whose Fourier transform is the box filter, is optimal for resampling a function. In practice, the infinite support sinc filter is approximated using a limited support 'windowed' sinc filter.

**Author** Paul A. Yushkevich

**THEORY**

Use this filter the way you would use any ImageInterpolationFunction, so for instance, you can plug it into the ResampleImageFilter class. In order to initialize the filter you must choose several template parameters.

This function is based on the following publication:

Erik H. W. Meijering, Wiro J. Niessen, Josien P. W. Pluim, Max A. Viergever: Quantitative Comparison of Sinc-Approximating Kernels for Medical Image Interpolation. MICCAI 1999, pp. 210-217

In this work, several 'windows' are estimated. In two dimensions, the interpolation at a position (x,y) is given by the following expression:

$$I(x,y) = \sum_{i=\lfloor x \rfloor + 1 - m}^{\lfloor x \rfloor + m} \sum_{j=\lfloor y \rfloor + 1 - m}^{\lfloor y \rfloor + m} I_{i,j} K(x-i) K(y-j),$$

where m is the 'radius' of the window, (3,4 are reasonable numbers), and K(t) is the kernel function, composed of the sinc function and one of several possible window functions:

$$K(t) = w(t)\text{sinc}(t) = w(t)\frac{\sin(\pi t)}{\pi t}$$

Several window functions are provided here in the itk::Function namespace. The conclusions of the referenced paper suggest to use the Welch, Cosine, Kaiser, and Lanczos windows for m = 4,5. These are based on error in rotating medical images w.r.t. the linear interpolation method. In some cases the results achieve a 20-fold improvement in accuracy.

**USING THIS FILTER**

There are a few improvements that an enthusiastic ITK developer could make to this filter. One issue is with the way that the kernel is applied. The computational expense comes from two sources: computing the kernel weights K(t) and multiplying the pixels in the window by the kernel weights. The first is done more or less efficiently in $2md$ operations (where d is the dimensionality of the image). The second can be done better. Presently, each pixel $I(i,j,k)$ is multiplied by the weights $K(x-i), K(y-j), K(z-k)$ and added to the running total. This results in $d(2m)^d$ multiplication operations. However, by keeping intermediate sums, it would be possible to do the operation in $O((2m)^d)$ operations. This would require some creative coding. In addition, in the case when one of the coordinates is integer, the computation could be reduced by an order of magnitude.

The first (TInputImage) is the image type, that's standard.

The second (VRadius) is the radius of the kernel, i.e., the $m$ from the formula above.

The third (TWindowFunction) is the window function object, which you can choose from about five different functions defined in this header. The default is the Hamming window, which is commonly used but not optimal according to the cited paper.

The fourth (TBoundaryCondition) is the boundary condition class used to determine the values of pixels that fall off the image boundary. This class has the same meaning here as in the NeighborhoodItetator classes.

The fifth (TCoordRep) is again standard for interpolating functions, and should be float or double.

**CAVEATS**

**See** LinearInterpolateImageFunction ResampleImageFilter

**See** Function::HammingWindowFunction

**See** Function::CosineWindowFunction

**See** Function::WelchWindowFunction

**See** Function::LanczosWindowFunction

**See** Function::BlackmanWindowFunction

See itk::WindowedSincInterpolateImageFunction for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**, typename **TInterpolatorPrecisionType** = double, typenan
**class ResampleImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Resample an image via a coordinate transform.

ResampleImageFilter resamples an existing image through some coordinate transform, interpolating via some image function. The class is templated over the types of the input and output images.

Note that the choice of interpolator function can be important. This function is set via SetInterpolator(). The default is LinearInterpolateImageFunction<InputImageType, TInterpolatorPrecisionType>, which is reasonable for ordinary medical images. However, some synthetic images have pixels drawn from a finite prescribed set. An example would be a mask indicating the segmentation of a brain into a small number of tissue types. For such an image, one does not want to interpolate between different pixel values, and so NearestNeighborInterpolateImageFunction< InputImageType, TCoordRep > would be a better choice.

If an sample is taken from outside the image domain, the default behavior is to use a default pixel value. If different behavior is desired, an extrapolator function can be set with SetExtrapolator().

Output information (spacing, size and direction) for the output image should be set. This information has the normal defaults of unit spacing, zero origin and identity direction. Optionally, the output information can be obtained from a reference image. If the reference image is provided and UseReferenceImage is On, then the spacing, origin and direction of the reference image will be used.

Since this filter produces an image which is a different size than its input, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Warning** For multithreading, the TransformPoint method of the user-designated coordinate transform must be threadsafe.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Image
- Upsampling An Image
- Resample An Image

See itk::ResampleImageFilter for additional documentation.

### Cartesian to Azimuth Elevation

### Synopsis

Cartesian to AzimuthElevation and vice-versa.

### Results

Output:

```
spherical: [0, 45, 1]
Cartesian: [0.00611663, 0.713237, 0.700896]
```

### Code

### C++

```
#include "itkPoint.h"
#include "itkAzimuthElevationToCartesianTransform.h"

int
main(int, char *[])
{
  using PointType = itk::Point<double, 3>;
  PointType spherical;
  spherical[0] = 0.0;
  spherical[1] = 45; // set elevation to 45 degrees
  spherical[2] = 1;
  std::cout << "spherical: " << spherical << std::endl;

  using AzimuthElevationToCartesian = itk::AzimuthElevationToCartesianTransform
↪<double, 3>;
  AzimuthElevationToCartesian::Pointer azimuthElevation = AzimuthElevationToCartesian:
↪:New();

  std::cout << "Cartesian: " << azimuthElevation->TransformAzElToCartesian(spherical)
↪<< std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class AzimuthElevationToCartesianTransform** : **public** itk::*AffineTransform*<*TParametersValueType*, *NDimensions*>

Transforms from an azimuth, elevation, radius coordinate system to a Cartesian coordinate system, or vice versa.

The three coordinate axis are azimuth, elevation, and range.

The azimuth elevation coordinate system is defined similarly to spherical coordinates but is slightly different in that the azimuth and elevation are measured in degrees between the r-axis (i.e z axis) and the projection on the x-z and y-z planes, respectively. Range, or r, is the distance from the origin.

The equations form performing the conversion from azimuth-elevation coordinates to cartesian coordinates are as follows:

```
z = std::sqrt((r^2*(cos(azimuth))^2)/(1 + (cos(azimuth))^2 * (tan(elevation))^2);
x = z * std::tan(azimuth)
y = z * std::tan(elevation)
```

The reversed transforms are:

```
azimuth = arctan(x/y)
elevation = arctan(y/z)
r = std::sqrt(x^2 + y^2 + z^2)
```

In this class, we can also set what a "forward" transform means. If we call SetForwardAzimuthElevationToCartesian(), a forward transform will return cartesian coordinates when passed azimuth,elevation,r coordinates. Calling SetForwardCartesianToAzimuthElevation() will cause the forward transform to return azimuth,elevation,r coordinates from cartesian coordinates.

Setting the FirstSampleDistance to a non-zero value means that a r value of 12 is actually (12 + FirstSampleDistance) distance from the origin.

There are two template parameters for this class:

TParametersValueType The type to be used for scalar numeric values. Either float or double.

NDimensions The number of dimensions of the vector space (must be >=3).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cartesian To Azimuth Elevation

See itk::AzimuthElevationToCartesianTransform for additional documentation.

### Copy a CompositeTransform

### Synopsis

Copy a CompositeTransform.

### Results

```
Original transform: CompositeTransform (0x173cdb0)
  RTTI typeinfo:   itk::CompositeTransform<float, 3u>
  Reference Count: 2
  Modified Time: 12
  Debug: Off
  Object Name:
  Observers:
    none
  Transforms in queue, from begin to end:
  >>>>>>>>>
  Euler3DTransform (0x173c880)
    RTTI typeinfo:   itk::Euler3DTransform<float>
    Reference Count: 2
    Modified Time: 5
    Debug: Off
    Object Name:
    Observers:
      none
    Matrix:
```

(continues on next page)

(continued from previous page)

```
      0.930432 -0.294044 0.218711
      0.308577 0.950564 -0.0347626
      -0.197677 0.0998334 0.97517
    Offset: [3.63622, 5.66636, 5.62082]
    Center: [-3.5, -4.5, -5.5]
    Translation: [4, 5, 6]
    Inverse:
      0.930432 0.308577 -0.197677
      -0.294044 0.950564 0.0998333
      0.218711 -0.0347626 0.97517
    Singular: 0
    Euler's angles: AngleX=0.1 AngleY=0.2 AngleZ=0.3
    m_ComputeZYX = 0
  >>>>>>>>>
  ScaleTransform (0x173cb10)
    RTTI typeinfo:   itk::ScaleTransform<float, 3u>
    Reference Count: 2
    Modified Time: 9
    Debug: Off
    Object Name:
    Observers:
      none
    Matrix:
      0.6 0 0
      0 0.7 0
      0 0 0.8
    Offset: [0, 0, 0]
    Center: [0, 0, 0]
    Translation: [0, 0, 0]
    Inverse:
      1.66667 0 0
      0 1.42857 0
      0 0 1.25
    Singular: 0
    Scale: [0.6, 0.7, 0.8]
    Center: [0, 0, 0]
  End of MultiTransform.
<<<<<<<<<<
  TransformsToOptimizeFlags, begin() to end():
    1 1
  TransformsToOptimize in queue, from begin to end:
  End of TransformsToOptimizeQueue.
<<<<<<<<<<
  End of CompositeTransform.
<<<<<<<<<<

Transform copy: CompositeTransform (0x173da00)
  RTTI typeinfo:   itk::CompositeTransform<float, 3u>
  Reference Count: 3
  Modified Time: 26
  Debug: Off
  Object Name:
  Observers:
    none
  Transforms in queue, from begin to end:
  >>>>>>>>>
  Euler3DTransform (0x173e2e0)
```

(continues on next page)

(continued from previous page)

```
    RTTI typeinfo:   itk::Euler3DTransform<float>
    Reference Count: 1
    Modified Time: 18
    Debug: Off
    Object Name:
    Observers:
      none
    Matrix:
      0.930432 -0.294044 0.218711
      0.308577 0.950564 -0.0347626
      -0.197677 0.0998334 0.97517
    Offset: [3.63622, 5.66636, 5.62082]
    Center: [-3.5, -4.5, -5.5]
    Translation: [4, 5, 6]
    Inverse:
      0.930432 0.308577 -0.197677
      -0.294044 0.950564 0.0998333
      0.218711 -0.0347626 0.97517
    Singular: 0
    Euler's angles: AngleX=0.1 AngleY=0.2 AngleZ=0.3
    m_ComputeZYX = 0
  >>>>>>>>>
  ScaleTransform (0x173e470)
    RTTI typeinfo:   itk::ScaleTransform<float, 3u>
    Reference Count: 1
    Modified Time: 24
    Debug: Off
    Object Name:
    Observers:
      none
    Matrix:
      0.6 0 0
      0 0.7 0
      0 0 0.8
    Offset: [0, 0, 0]
    Center: [0, 0, 0]
    Translation: [0, 0, 0]
    Inverse:
      1.66667 0 0
      0 1.42857 0
      0 0 1.25
    Singular: 0
    Scale: [0.6, 0.7, 0.8]
    Center: [0, 0, 0]
  End of MultiTransform.
<<<<<<<<<<
  TransformsToOptimizeFlags, begin() to end():
    1 1
  TransformsToOptimize in queue, from begin to end:
  End of TransformsToOptimizeQueue.
<<<<<<<<<<
  End of CompositeTransform.
<<<<<<<<<<
```

### Code

### C++

```
#include "itkEuler3DTransform.h"
#include "itkScaleTransform.h"
#include "itkCompositeTransform.h"
#include "itkCompositeTransformIOHelper.h"

int
main(int, char *[])
{
  using ScalarType = float;
  constexpr unsigned int Dimension = 3;

  using EulerTransformType = itk::Euler3DTransform<ScalarType>;
  EulerTransformType::Pointer        eulerTransform = EulerTransformType::New();
  EulerTransformType::ParametersType eulerParameters(6);
  eulerParameters[0] = 0.1;
  eulerParameters[1] = 0.2;
  eulerParameters[2] = 0.3;
  eulerParameters[3] = 4.0;
  eulerParameters[4] = 5.0;
  eulerParameters[5] = 6.0;
  eulerTransform->SetParameters(eulerParameters);

  EulerTransformType::FixedParametersType eulerFixedParameters(Dimension);
  eulerFixedParameters[0] = -3.5;
  eulerFixedParameters[1] = -4.5;
  eulerFixedParameters[2] = -5.5;
  eulerTransform->SetFixedParameters(eulerFixedParameters);

  using ScaleTransformType = itk::ScaleTransform<ScalarType, Dimension>;
  ScaleTransformType::Pointer        scaleTransform = ScaleTransformType::New();
  ScaleTransformType::ParametersType scaleParameters(Dimension);
  scaleParameters[0] = 0.6;
  scaleParameters[1] = 0.7;
  scaleParameters[2] = 0.8;
  scaleTransform->SetParameters(scaleParameters);

  using CompositeTransformType = itk::CompositeTransform<ScalarType, Dimension>;
  CompositeTransformType::Pointer compositeTransform = CompositeTransformType::New();
  compositeTransform->AddTransform(eulerTransform);
  compositeTransform->AddTransform(scaleTransform);
  std::cout << "Original transform: " << compositeTransform << std::endl;


  CompositeTransformType::Pointer transformCopy = compositeTransform->Clone();
  std::cout << "Transform copy: " << transformCopy << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType**, unsigned int **NInputDimensions** = 3, unsigned int **NOutputDimensions** = 3
**class Transform** : **public** itk::TransformBaseTemplate<*TParametersValueType*>

Transform points and vectors from an input space to an output space.

This abstract class defines the generic interface for a geometric transformation from one space to another. The class provides methods for mapping points, vectors and covariant vectors from the input space to the output space.

Given that transformations are not necessarily invertible, this basic class does not provide the methods for back transformation. Back transform methods are implemented in derived classes where appropriate.

Another requirement of the registration framework is the computation of the transform Jacobian. In general, an ImageToImageMetric requires the knowledge of the Jacobian in order to compute the metric derivatives. The Jacobian is a matrix whose element are the partial derivatives of the output point with respect to the array of parameters that defines the transform.

**Registration Framework Support** Typically a Transform class has several methods for setting its parameters. For use in the registration framework, the parameters must also be represented by an array of doubles to allow communication with generic optimizers. The Array of transformation parameters is set using the SetParameters() method.

Subclasses must provide implementations for: virtual OutputPointType TransformPoint(const InputPointType &) const virtual OutputVectorType TransformVector(const InputVectorType &) const virtual OutputVnlVectorType TransformVector(const InputVnlVectorType &) const virtual OutputCovariantVectorType TransformCovariantVector(const InputCovariantVectorType &) const virtual void SetParameters(const ParametersType &) virtual void SetFixedParameters(const FixedParametersType &) virtual void ComputeJacobianWithRespectToParameters( const InputPointType &, JacobianType &) const

virtual void ComputeJacobianWithRespectToPosition( const InputPointType & x, JacobianPositionType &jacobian ) const;

Since TranformVector and TransformCovariantVector have multiple overloaded methods from the base class, subclasses must specify: using Superclass::TransformVector;

using Superclass::TransformCovariantVector;

Subclassed by itk::MatrixOffsetTransformBase< TParametersValueType, NInputDimensions, NOutputDimensions >, itk::MatrixOffsetTransformBase< TParametersValueType, 2, 2 >, itk::MatrixOffsetTransformBase< TParametersValueType, 3, 3 >, itk::MatrixOffsetTransformBase< TParametersValueType, NDimensions, NDimensions >

See itk::Transform for additional documentation.

### Copy a Non-CompositeTransform

### Synopsis

Copy a non-CompositeTransform.

### Results

```
Original transform: Euler3DTransform (0x132e880)
  RTTI typeinfo:   itk::Euler3DTransform<float>
  Reference Count: 2
  Modified Time: 5
  Debug: Off
  Object Name:
  Observers:
    none
  Matrix:
    0.930432 -0.294044 0.218711
    0.308577 0.950564 -0.0347626
    -0.197677 0.0998334 0.97517
  Offset: [3.63622, 5.66636, 5.62082]
  Center: [-3.5, -4.5, -5.5]
  Translation: [4, 5, 6]
  Inverse:
    0.930432 0.308577 -0.197677
    -0.294044 0.950564 0.0998333
    0.218711 -0.0347626 0.97517
  Singular: 0
  Euler's angles: AngleX=0.1 AngleY=0.2 AngleZ=0.3
  m_ComputeZYX = 0

Transform copy: Euler3DTransform (0x132eda0)
  RTTI typeinfo:   itk::Euler3DTransform<float>
  Reference Count: 3
  Modified Time: 10
  Debug: Off
  Object Name:
  Observers:
    none
  Matrix:
    0.930432 -0.294044 0.218711
    0.308577 0.950564 -0.0347626
    -0.197677 0.0998334 0.97517
  Offset: [3.63622, 5.66636, 5.62082]
  Center: [-3.5, -4.5, -5.5]
  Translation: [4, 5, 6]
  Inverse:
    0.930432 0.308577 -0.197677
    -0.294044 0.950564 0.0998333
    0.218711 -0.0347626 0.97517
  Singular: 0
  Euler's angles: AngleX=0.1 AngleY=0.2 AngleZ=0.3
  m_ComputeZYX = 0
```

### Code

### C++

```
#include "itkEuler3DTransform.h"

int
main(int, char *[])
{
  using TransformType = itk::Euler3DTransform<float>;

  TransformType::Pointer        transform = TransformType::New();
  TransformType::ParametersType parameters(6);
  parameters[0] = 0.1;
  parameters[1] = 0.2;
  parameters[2] = 0.3;
  parameters[3] = 4.0;
  parameters[4] = 5.0;
  parameters[5] = 6.0;
  transform->SetParameters(parameters);

  TransformType::FixedParametersType fixedParameters(3);
  fixedParameters[0] = -3.5;
  fixedParameters[1] = -4.5;
  fixedParameters[2] = -5.5;
  transform->SetFixedParameters(fixedParameters);
  std::cout << "Original transform: " << transform << std::endl;

  TransformType::Pointer transformCopy = transform->Clone();
  std::cout << "Transform copy: " << transformCopy << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType**, unsigned int **NInputDimensions** = 3, unsigned int **NOutputDimensions** = 3
**class Transform** : **public** itk::TransformBaseTemplate<*TParametersValueType*>

Transform points and vectors from an input space to an output space.

This abstract class defines the generic interface for a geometric transformation from one space to another. The class provides methods for mapping points, vectors and covariant vectors from the input space to the output space.

Given that transformations are not necessarily invertible, this basic class does not provide the methods for back transformation. Back transform methods are implemented in derived classes where appropriate.

Another requirement of the registration framework is the computation of the transform Jacobian. In general, an ImageToImageMetric requires the knowledge of the Jacobian in order to compute the metric derivatives. The Jacobian is a matrix whose element are the partial derivatives of the output point with respect to the array of parameters that defines the transform.

**Registration Framework Support** Typically a Transform class has several methods for setting its parameters. For use in the registration framework, the parameters must also be represented by an array of doubles to allow communication with generic optimizers. The Array of transformation parameters is set using the SetParameters() method.

Subclasses must provide implementations for: virtual OutputPointType TransformPoint(const InputPointType &) const virtual OutputVectorType TransformVector(const InputVectorType &) const virtual OutputVnlVectorType TransformVector(const InputVnlVectorType &) const virtual OutputCovariantVectorType TransformCovariantVector(const InputCovariantVectorType &) const virtual void SetParameters(const ParametersType &) virtual void SetFixedParameters(const FixedParametersType &) virtual void ComputeJacobianWithRespectToParameters( const InputPointType &, JacobianType &) const

virtual void ComputeJacobianWithRespectToPosition( const InputPointType & x, JacobianPositionType &jacobian ) const;

Since TranformVector and TransformCovariantVector have multiple overloaded methods from the base class, subclasses must specify: using Superclass::TransformVector;

using Superclass::TransformCovariantVector;

Subclassed by itk::MatrixOffsetTransformBase< TParametersValueType, NInputDimensions, NOutputDimensions >, itk::MatrixOffsetTransformBase< TParametersValueType, 2, 2 >, itk::MatrixOffsetTransformBase< TParametersValueType, 3, 3 >, itk::MatrixOffsetTransformBase< TParametersValueType, NDimensions, NDimensions >

See itk::Transform for additional documentation.

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class CompositeTransform** : **public** itk::MultiTransform<*TParametersValueType*, *NDimensions*, *NDimensions*>

This class contains a list of transforms and concatenates them by composition.

This class concatenates transforms in **reverse queue** order by means of composition: $T_0 o T_1 = T_0(T_1(x))$ Transforms are stored in a container (queue), in the following order: $T_0, T_1, ..., T_N - 1$ Transforms are added via a single method, AddTransform(). This adds the transforms to the back of the queue. A single method for adding transforms is meant to simplify the interface and prevent errors. One use of the class is to optimize only a subset of included transforms.

The sub transforms are the same dimensionality as this class.

Example: A user wants to optimize two Affine transforms together, then add a Deformation Field (DF) transform, and optimize it separately. He first adds the two Affines, then runs the optimization and both Affines transforms are optimized. Next, he adds the DF transform and calls SetOnlyMostRecentTransformToOptimizeOn, which clears the optimization flags for both of the affine transforms, and leaves the flag set only for the DF transform, since it was the last transform added. Now he runs the optimization and only the DF transform is optimized, but the affines are included in the transformation during the optimization.

Optimization Flags: The m_TransformsToOptimize flags hold one flag for each transform in the queue, designating if each transform is to be used for optimization. Note that all transforms in the queue are applied in TransformPoint, regardless of these flags states'. The methods GetParameters, SetParameters, ComputeJacobianWithRespectToParameters, GetTransformCategory, GetFixedParameters, and SetFixedParameters all query these flags and include only those transforms whose corresponding flag is set. Their input or output is a concatenated array of all transforms set for use in optimization. The goal is to be able to optimize multiple transforms at once, while leaving other transforms fixed. See the above example.

Setting Optimization Flags: A transform's optimization flag is set when it is added to the queue, and remains set as other transforms are added. The methods SetNthTransformToOptimize* and SetAllTransformToOptimize* are used to set and clear flags arbitrarily. SetOnlyMostRecentTransformToOptimizeOn is a convenience method for setting only the most recently added transform for optimization, with the idea that this will be a common practice.

Indexing: The index values used in GetNthTransform and SetNthTransformToOptimize* and SetAllTransformToOptimize* follow the order in which transforms were added. Thus, the first transform added is at index 0, the next at index 1, etc.

Inverse: The inverse transform is created by retrieving the inverse from each sub transform and adding them to a composite transform in reverse order. The m_TransformsToOptimizeFlags is copied in reverse for the inverse.

See itk::CompositeTransform for additional documentation.

### Global Registration Two Images (Affine)

### Synopsis

A basic global registration of two images.

### Results

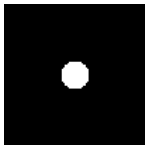


Fig. 70: fixed.png

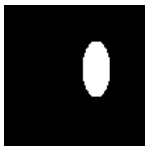


Fig. 71: moving.png

Fig. 72: output.png

Output:

```
Final parameters: [1.1921533496320087, 0.10372902569023787, -0.18132002016416124, 1.
↪1464158834351523, 0.0021451859042650244, -0.003039195975788232]
Result =
Metric value  = 1836.41
```

### Code

### C++

```
#include "itkCastImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkImage.h"
#include "itkImageRegistrationMethod.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMeanSquaresImageToImageMetric.h"
#include "itkRegularStepGradientDescentOptimizer.h"
#include "itkResampleImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkAffineTransform.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned char;

using ImageType = itk::Image<PixelType, Dimension>;

static void
CreateEllipseImage(ImageType::Pointer image);
static void
CreateSphereImage(ImageType::Pointer image);

int
main(int, char *[])
{
  //  The transform that will map the fixed image into the moving image.
  using TransformType = itk::AffineTransform<double, Dimension>;

  //  An optimizer is required to explore the parameter space of the transform
  //  in search of optimal values of the metric.
  using OptimizerType = itk::RegularStepGradientDescentOptimizer;

  //  The metric will compare how well the two images match each other. Metric
  //  types are usually parameterized by the image types as it can be seen in
  //  the following type declaration.
  using MetricType = itk::MeanSquaresImageToImageMetric<ImageType, ImageType>;

  //  Finally, the type of the interpolator is declared. The interpolator will
  //  evaluate the intensities of the moving image at non-grid positions.
  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;

  //  The registration method type is instantiated using the types of the
  //  fixed and moving images. This class is responsible for interconnecting
  //  all the components that we have described so far.
  using RegistrationType = itk::ImageRegistrationMethod<ImageType, ImageType>;

  // Create components
  MetricType::Pointer       metric = MetricType::New();
  TransformType::Pointer    transform = TransformType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
```

(continues on next page)

(continued from previous page)

```
  RegistrationType::Pointer registration = RegistrationType::New();

  // Each component is now connected to the instance of the registration method.
  registration->SetMetric(metric);
  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);

  // Get the two images
  ImageType::Pointer fixedImage = ImageType::New();
  ImageType::Pointer movingImage = ImageType::New();

  CreateSphereImage(fixedImage);
  CreateEllipseImage(movingImage);

  // Write the two synthetic inputs
  using WriterType = itk::ImageFileWriter<ImageType>;

  WriterType::Pointer fixedWriter = WriterType::New();
  fixedWriter->SetFileName("fixed.png");
  fixedWriter->SetInput(fixedImage);
  fixedWriter->Update();

  WriterType::Pointer movingWriter = WriterType::New();
  movingWriter->SetFileName("moving.png");
  movingWriter->SetInput(movingImage);
  movingWriter->Update();

  // Set the registration inputs
  registration->SetFixedImage(fixedImage);
  registration->SetMovingImage(movingImage);

  registration->SetFixedImageRegion(fixedImage->GetLargestPossibleRegion());

  //  Initialize the transform
  using ParametersType = RegistrationType::ParametersType;
  ParametersType initialParameters(transform->GetNumberOfParameters());

  // rotation matrix
  initialParameters[0] = 1.0; // R(0,0)
  initialParameters[1] = 0.0; // R(0,1)
  initialParameters[2] = 0.0; // R(1,0)
  initialParameters[3] = 1.0; // R(1,1)

  // translation vector
  initialParameters[4] = 0.0;
  initialParameters[5] = 0.0;

  registration->SetInitialTransformParameters(initialParameters);

  optimizer->SetMaximumStepLength(.1); // If this is set too high, you will get a
  //"itk::ERROR: MeanSquaresImageToImageMetric(0xa27ce70): Too many samples map␣
↪outside moving image buffer: 1818 /
  // 10000" error

  optimizer->SetMinimumStepLength(0.01);
```

(continues on next page)

(continued from previous page)

```
// Set a stopping criterion
optimizer->SetNumberOfIterations(200);

// Connect an observer
// CommandIterationUpdate::Pointer observer = CommandIterationUpdate::New();
// optimizer->AddObserver( itk::IterationEvent(), observer );

try
{
  registration->Update();
}
catch (itk::ExceptionObject & err)
{
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
}

//  The result of the registration process is an array of parameters that
//  defines the spatial transformation in an unique way. This final result is
//  obtained using the \code{GetLastTransformParameters()} method.

ParametersType finalParameters = registration->GetLastTransformParameters();
std::cout << "Final parameters: " << finalParameters << std::endl;

//  The value of the image metric corresponding to the last set of parameters
//  can be obtained with the \code{GetValue()} method of the optimizer.

const double bestValue = optimizer->GetValue();

// Print out results
//
std::cout << "Result = " << std::endl;
std::cout << " Metric value  = " << bestValue << std::endl;

//  It is common, as the last step of a registration task, to use the
//  resulting transform to map the moving image into the fixed image space.
//  This is easily done with the \doxygen{ResampleImageFilter}.

using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;

ResampleFilterType::Pointer resampler = ResampleFilterType::New();
resampler->SetInput(movingImage);

//  The Transform that is produced as output of the Registration method is
//  also passed as input to the resampling filter. Note the use of the
//  methods \code{GetOutput()} and \code{Get()}. This combination is needed
//  here because the registration method acts as a filter whose output is a
//  transform decorated in the form of a \doxygen{DataObject}. For details in
//  this construction you may want to read the documentation of the
//  \doxygen{DataObjectDecorator}.

resampler->SetTransform(registration->GetOutput()->Get());

//  As described in Section \ref{sec:ResampleImageFilter}, the
//  ResampleImageFilter requires additional parameters to be specified, in
//  particular, the spacing, origin and size of the output image. The default
```

(continues on next page)

(continued from previous page)

```
  //  pixel value is also set to a distinct gray level in order to highlight
  //  the regions that are mapped outside of the moving image.

  resampler->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resampler->SetOutputOrigin(fixedImage->GetOrigin());
  resampler->SetOutputSpacing(fixedImage->GetSpacing());
  resampler->SetOutputDirection(fixedImage->GetDirection());
  resampler->SetDefaultPixelValue(100);

  //  The output of the filter is passed to a writer that will store the
  //  image in a file. An \doxygen{CastImageFilter} is used to convert the
  //  pixel type of the resampled image to the final type used by the
  //  writer. The cast and writer filters are instantiated below.

  using CastFilterType = itk::CastImageFilter<ImageType, ImageType>;

  WriterType::Pointer     writer = WriterType::New();
  CastFilterType::Pointer caster = CastFilterType::New();
  writer->SetFileName("output.png");

  caster->SetInput(resampler->GetOutput());
  writer->SetInput(caster->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateEllipseImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing.Fill(1);
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 20;
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();
```

(continues on next page)

(continued from previous page)

```
  TransformType::OutputVectorType translation;
  translation[0] = 65;
  translation[1] = 45;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}

void
CreateSphereImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing.Fill(1);
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 10;
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = 50;
  translation[1] = 50;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);
```

(continues on next page)

(continued from previous page)

```
  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}
```

### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class AffineTransform** : **public** itk::MatrixOffsetTransformBase<*TParametersValueType*, *NDimensions*, *NDimensions*>

Affine transformation of a vector space (e.g. space coordinates)

This class allows the definition and manipulation of affine transformations of an n-dimensional affine space (and its associated vector space) onto itself. One common use is to define and manipulate Euclidean coordinate transformations in two and three dimensions, but other uses are possible as well.

An affine transformation is defined mathematically as a linear transformation plus a constant offset. If A is a constant n x n matrix and b is a constant n-vector, then y = Ax+b defines an affine transformation from the n-vector x to the n-vector y.

The difference between two points is a vector and transforms linearly, using the matrix only. That is, (y1-y2) = A*(x1-x2).

The AffineTransform class determines whether to transform an object as a point or a vector by examining its type. An object of type Point transforms as a point; an object of type Vector transforms as a vector.

One common use of affine transformations is to define coordinate conversions in two- and three-dimensional space. In this application, x is a two- or three-dimensional vector containing the "source" coordinates of a point, y is a vector containing the "target" coordinates, the matrix A defines the scaling and rotation of the coordinate systems from the source to the target, and b defines the translation of the origin from the source to the target. More generally, A can also define anisotropic scaling and shearing transformations. Any good textbook on computer graphics will discuss coordinate transformations in more detail. Several of the methods in this class are designed for this purpose and use the language appropriate to coordinate conversions.

Any two affine transformations may be composed and the result is another affine transformation. However, the order is important. Given two affine transformations T1 and T2, we will say that "precomposing T1 with T2" yields the transformation which applies T1 to the source, and then applies T2 to that result to obtain the target. Conversely, we will say that "postcomposing T1 with T2" yields the transformation which applies T2 to the source, and then applies T1 to that result to obtain the target. (Whether T1 or T2 comes first lexicographically depends on whether you choose to write mappings from right-to-left or vice versa; we avoid the whole problem by referring to the order of application rather than the textual order.)

There are two template parameters for this class:

TParametersValueType The type to be used for scalar numeric values. Either float or double.

NDimensions The number of dimensions of the vector space.

This class provides several methods for setting the matrix and vector defining the transform. To support the registration framework, the transform parameters can also be set as an Array<double> of size (NDimension +

1) * NDimension using method SetParameters(). The first (NDimension x NDimension) parameters defines the matrix in row-major order (where the column index varies the fastest). The last NDimension parameters defines the translation in each dimensions.

This class also supports the specification of a center of rotation (center) and a translation that is applied with respect to that centered rotation. By default the center of rotation is set to the origin.

Subclassed by itk::AzimuthElevationToCartesianTransform< TParametersValueType, NDimensions >, itk::CenteredAffineTransform< TParametersValueType, NDimensions >, itk::ScalableAffineTransform< TParametersValueType, NDimensions >

See itk::AffineTransform for additional documentation.

template<typename **TFixedImage**, typename **TMovingImage**>
**class ImageRegistrationMethod** : **public** itk::*ProcessObject*

Base class for Image Registration Methods.

This Class define the generic interface for a registration method.

This class is templated over the type of the two image to be registered. A generic Transform is used by this class. That allows to select at run time the particular type of transformation that is to be applied for registering the images.

This method use a generic Metric in order to compare the two images. the final goal of the registration method is to find the set of parameters of the Transformation that optimizes the metric.

The registration method also support a generic optimizer that can be selected at run-time. The only restriction for the optimizer is that it should be able to operate in single-valued cost functions given that the metrics used to compare images provide a single value as output.

The terms : Fixed image and Moving image are used in this class to indicate what image is being mapped by the transform.

This class uses the coordinate system of the Fixed image as a reference and searches for a Transform that will map points from the space of the Fixed image to the space of the Moving image.

For doing so, a Metric will be continuously applied to compare the Fixed image with the Transformed Moving image. This process also requires to interpolate values from the Moving image.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Global Registration Of Two Images
- Global Registration Two Images (Affine)
- Global Registration Of Two Images (BSpline)

See itk::ImageRegistrationMethod for additional documentation.

### Global Registration of Two Images (BSpline)

### Synopsis

A global registration of two images.

### Results

Output:

```
Intial Parameters =
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,␣
↪0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Starting Registration
************************************************
N=50   NUMBER OF CORRECTIONS=5       INITIAL VALUES F= 5410.08   GNORM= 98.7811
************************************************
I   NFN     FUNC         GNORM       STEPLENGTH
1     2      5173.917      47.318       0.010
2     8      1312.526      35.022      67.448
3    10      1241.036      21.952       0.095
4    11      1214.603      20.777       0.500
5    12      1169.045      47.965       0.500
6    13      1066.160      73.507       0.500
7    14       750.612      84.979       0.500
8    16       427.232      44.597       0.235
9    17       371.768      44.133       0.500
10   19        320.897      30.003       0.054
11   20        319.018      23.934       0.500
12   21        276.782      16.253       0.500
13   22        190.496      16.864       0.500
14   23        121.434      17.719       0.500
15   24         80.920      23.984       0.500
16   25         50.736      11.243       0.500
17   26         40.862       5.633       0.500
THE MINIMIZATION TERMINATED WITHOUT DETECTING ERRORS.
Optimizer stop condition = LBFGSOptimizer: Failure
Last Transform Parameters
[0.000004242652759130258, -0.09801257315348906, -0.3773136074775823, 0.
↪04057356343246512, 0.011460974084144235, 0.046750769129838214, 5.442980649652609, -
↪1.3233672220673571, 10.138325385009857, 1.2593858799352904, 0.1744432989808139, 27.
↪255697794222773, 18.42787109104701, 27.82714519423285, 3.443535460389487, 0.
↪04083243642873029, 5.114318164332667, 0.5697948754499333, 5.4282475197978695, 0.
↪7721289215254623, -0.00001989578361620887, -0.034703497612920686, -0.
↪08550784338556144, 0.006239176991931418, 0.001367593141535802, -4.808966670313369e-
↪7, -0.2187505995413243, -1.8087191436710883, -0.9922884310554664, -0.
↪014742349905146031, 0.003053794283557083, -11.874136737768657, -75.22296974556679, -
↪33.96218955913907, -0.42061613449053764, 0.011561263304423593, -0.7922226379398286,␣
↪-7.185598670736005, 0.7735213860631797, 0.1316131069607225, 0.0033284552078211094,␣
↪10.085406006964824, 56.37568206497135, 25.246189532623205, 0.3387536225030382, 0.
↪00001586564956650034 8, 0.11028526291334259, 0.6117750781918082, 0.26063979972963747,
↪ 0.0029171257653189493]
```

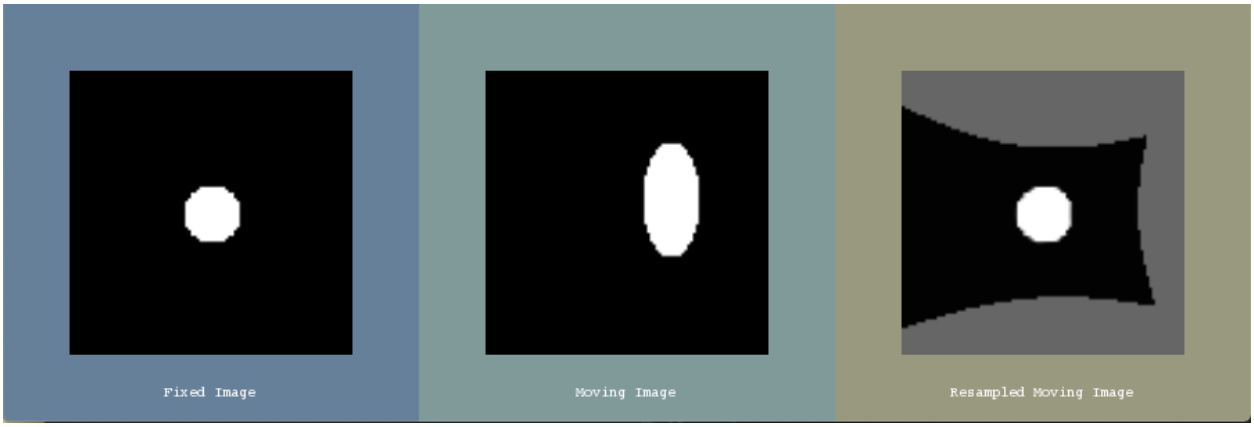



Fig. 73: Output Image

## Code

### C++

```
#include "itkImageRegistrationMethod.h"
#include "itkMeanSquaresImageToImageMetric.h"
#include "itkTimeProbesCollectorBase.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkEllipseSpatialObject.h"

#if ITK_VERSION_MAJOR < 4
#  include "itkBSplineDeformableTransform.h"
#else
#  include "itkBSplineTransform.h"
#endif
#include "itkLBFGSOptimizer.h"
#include "itkImageFileWriter.h"
#include "itkResampleImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkSquaredDifferenceImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

constexpr unsigned int ImageDimension = 2;
using PixelType = float;

using ImageType = itk::Image<PixelType, ImageDimension>;

static void
CreateEllipseImage(ImageType::Pointer image);
static void
CreateCircleImage(ImageType::Pointer image);

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
```

(continues on next page)

(continued from previous page)

```
  const unsigned int     SpaceDimension = ImageDimension;
  constexpr unsigned int SplineOrder = 3;
  using CoordinateRepType = double;

#if ITK_VERSION_MAJOR < 4
  using TransformType = itk::BSplineDeformableTransform<CoordinateRepType,␣
↪SpaceDimension, SplineOrder>;
#else
  using TransformType = itk::BSplineTransform<CoordinateRepType, SpaceDimension,␣
↪SplineOrder>;
#endif
  using OptimizerType = itk::LBFGSOptimizer;


  using MetricType = itk::MeanSquaresImageToImageMetric<ImageType, ImageType>;

  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;

  using RegistrationType = itk::ImageRegistrationMethod<ImageType, ImageType>;

  MetricType::Pointer       metric = MetricType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();


  // The old registration framework has problems with multi-threading
  // For now, we set the number of work units to 1
  registration->SetNumberOfWorkUnits(1);

  registration->SetMetric(metric);
  registration->SetOptimizer(optimizer);
  registration->SetInterpolator(interpolator);

  TransformType::Pointer transform = TransformType::New();
  registration->SetTransform(transform);

  // Create the synthetic images
  ImageType::Pointer fixedImage = ImageType::New();
  CreateCircleImage(fixedImage);

  ImageType::Pointer movingImage = ImageType::New();
  CreateEllipseImage(movingImage);

  // Setup the registration
  registration->SetFixedImage(fixedImage);
  registration->SetMovingImage(movingImage);

  ImageType::RegionType fixedRegion = fixedImage->GetBufferedRegion();
  registration->SetFixedImageRegion(fixedRegion);

  //  Here we define the parameters of the BSplineDeformableTransform grid.  We
  //  arbitrarily decide to use a grid with $5 \times 5$ nodes within the image.
  //  The reader should note that the BSpline computation requires a
  //  finite support region ( 1 grid node at the lower borders and 2
  //  grid nodes at upper borders). Therefore in this example, we set
```

(continues on next page)

(continued from previous page)

```
  //  the grid size to be $8 \times 8$ and place the grid origin such that
  //  grid node (1,1) coincides with the first pixel in the fixed image.

#if ITK_VERSION_MAJOR < 4
  using RegionType = TransformType::RegionType;
  RegionType           bsplineRegion;
  RegionType::SizeType gridSizeOnImage;
  RegionType::SizeType gridBorderSize;
  RegionType::SizeType totalGridSize;

  gridSizeOnImage.Fill(5);
  gridBorderSize.Fill(3); // Border for spline order = 3 ( 1 lower, 2 upper )
  totalGridSize = gridSizeOnImage + gridBorderSize;

  bsplineRegion.SetSize(totalGridSize);

  using SpacingType = TransformType::SpacingType;
  SpacingType spacing = fixedImage->GetSpacing();

  using OriginType = TransformType::OriginType;
  OriginType origin = fixedImage->GetOrigin();

  ImageType::SizeType fixedImageSize = fixedRegion.GetSize();

  for (unsigned int r = 0; r < ImageDimension; r++)
  {
    spacing[r] *= static_cast<double>(fixedImageSize[r] - 1) / static_cast<double>
↪(gridSizeOnImage[r] - 1);
  }

  ImageType::DirectionType gridDirection = fixedImage->GetDirection();
  SpacingType              gridOriginOffset = gridDirection * spacing;

  OriginType gridOrigin = origin - gridOriginOffset;

  transform->SetGridSpacing(spacing);
  transform->SetGridOrigin(gridOrigin);
  transform->SetGridRegion(bsplineRegion);
  transform->SetGridDirection(gridDirection);
#else
  TransformType::PhysicalDimensionsType fixedPhysicalDimensions;
  TransformType::MeshSizeType           meshSize;
  for (unsigned int i = 0; i < ImageDimension; i++)
  {
    fixedPhysicalDimensions[i] =
      fixedImage->GetSpacing()[i] * static_cast<double>(fixedImage->
↪GetLargestPossibleRegion().GetSize()[i] - 1);
  }
  unsigned int numberOfGridNodesInOneDimension = 5;
  meshSize.Fill(numberOfGridNodesInOneDimension - SplineOrder);
  transform->SetTransformDomainOrigin(fixedImage->GetOrigin());
  transform->SetTransformDomainPhysicalDimensions(fixedPhysicalDimensions);
  transform->SetTransformDomainMeshSize(meshSize);
  transform->SetTransformDomainDirection(fixedImage->GetDirection());
#endif

  using ParametersType = TransformType::ParametersType;
```

(continues on next page)

(continued from previous page)

```
  const unsigned int numberOfParameters = transform->GetNumberOfParameters();

  ParametersType parameters(numberOfParameters);

  parameters.Fill(0.0);

  transform->SetParameters(parameters);

  //  We now pass the parameters of the current transform as the initial
  //  parameters to be used when the registration process starts.

  registration->SetInitialTransformParameters(transform->GetParameters());

  std::cout << "Intial Parameters = " << std::endl;
  std::cout << transform->GetParameters() << std::endl;

  //  Next we set the parameters of the LBFGS Optimizer.

  optimizer->SetGradientConvergenceTolerance(0.05);
  optimizer->SetLineSearchAccuracy(0.9);
  optimizer->SetDefaultStepLength(.5);
  optimizer->TraceOn();
  optimizer->SetMaximumNumberOfFunctionEvaluations(1000);

  std::cout << std::endl << "Starting Registration" << std::endl;

  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition = " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }

  OptimizerType::ParametersType finalParameters = registration->
↪GetLastTransformParameters();

  std::cout << "Last Transform Parameters" << std::endl;
  std::cout << finalParameters << std::endl;

  transform->SetParameters(finalParameters);

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;

  ResampleFilterType::Pointer resample = ResampleFilterType::New();

  resample->SetTransform(transform);
  resample->SetInput(movingImage);

  resample->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
```

(continues on next page)

(continued from previous page)

```
  resample->SetOutputOrigin(fixedImage->GetOrigin());
  resample->SetOutputSpacing(fixedImage->GetSpacing());
  resample->SetOutputDirection(fixedImage->GetDirection());
  resample->SetDefaultPixelValue(100);

  using OutputPixelType = unsigned char;

  using OutputImageType = itk::Image<OutputPixelType, ImageDimension>;

  using CastFilterType = itk::CastImageFilter<ImageType, OutputImageType>;

  using OutputWriterType = itk::ImageFileWriter<OutputImageType>;

  OutputWriterType::Pointer writer = OutputWriterType::New();
  CastFilterType::Pointer   caster = CastFilterType::New();


  writer->SetFileName("output.png");

  caster->SetInput(resample->GetOutput());
  writer->SetInput(caster->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(fixedImage.GetPointer(), true, "Fixed Image");
  viewer.AddImage(movingImage.GetPointer(), true, "Moving Image");
  viewer.AddImage(resample->GetOutput(), true, "Resampled Moving Image");

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

void
CreateEllipseImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<ImageDimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
```

(continues on next page)

(continued from previous page)

```
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing.Fill(1);
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 20;
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = 65;
  translation[1] = 45;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}

void
CreateCircleImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<ImageDimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing.Fill(1);
```

(continues on next page)

(continued from previous page)

```
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 10;
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = 50;
  translation[1] = 50;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}
```

## Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3, unsigned int **VSplineOrder** = 3>
**class BSplineDeformableTransform** : **public** itk::BSplineBaseTransform<*TParametersValueType*, *NDimensions*, *VSpline*

Deformable transform using a BSpline representation.

This class encapsulates a deformable transform of points from one N-dimensional space to another N-dimensional space. The deformation field is modelled using B-splines. A deformation is defined on a sparse regular grid of control points $\vec{\lambda}_j$ and is varied by defining a deformation $\vec{g}(\vec{\lambda}_j)$ of each control point. The deformation $D(\vec{x})$ at any point $\vec{x}$ is obtained by using a B-spline interpolation kernel.

**Note** BSplineTransform is a newer version of this class, and it is preferred.

The deformation field grid is defined by a user specified GridRegion, GridSpacing and GridOrigin. Each grid/control point has associated with it N deformation coefficients $\vec{\delta}_j$, representing the N directional components of the deformation. Deformation outside the grid plus support region for the BSpline interpolation is assumed to be zero.

Additionally, the user can specified an addition bulk transform $B$ such that the transformed point is given by:

$$\vec{y} = B(\vec{x}) + D(\vec{x})$$

The parameters for this transform is an N x N-D grid of spline coefficients. The user specifies the parameters as one flat array: each N-D grid is represented by an array in the same way an N-D image is represented in the buffer; the N arrays are then concatenated together on form a single array.

For efficiency, this transform does not make a copy of the parameters. It only keeps a pointer to the input parameters and assumes that the memory is managed by the caller.

The following illustrates the typical usage of this class:

```
using TransformType = BSplineDeformableTransform<double,2,3>;
TransformType::Pointer transform = TransformType::New();

transform->SetGridRegion( region );
transform->SetGridSpacing( spacing );
transform->SetGridOrigin( origin );

// NB: the region must be set first before setting the parameters

TransformType::ParametersType parameters( transform->GetNumberOfParameters() );

// Fill the parameters with values

transform->SetParameters( parameters )

outputPoint = transform->TransformPoint( inputPoint );
```

An alternative way to set the B-spline coefficients is via array of images. The grid region, spacing and origin information is taken directly from the first image. It is assumed that the subsequent images are the same buffered region. The following illustrates the API:

```
TransformType::ImageConstPointer images[2];

// Fill the images up with values

transform->SetCoefficientImages( images );
outputPoint = transform->TransformPoint( inputPoint );
```

Warning: use either the SetParameters() or SetCoefficientImages() API. Mixing the two modes may results in unexpected results.

The class is templated coordinate representation type (float or double), the space dimension and the spline order.

**See** BSplineTransform

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Global Registration Of Two Images (BSpline)

See itk::BSplineDeformableTransform for additional documentation.

template<typename **TFixedImage**, typename **TMovingImage**>
**class ImageRegistrationMethod** : **public** itk::*ProcessObject*

Base class for Image Registration Methods.

This Class define the generic interface for a registration method.

This class is templated over the type of the two image to be registered. A generic Transform is used by this class. That allows to select at run time the particular type of transformation that is to be applied for registering the images.

This method use a generic Metric in order to compare the two images. the final goal of the registration method is to find the set of parameters of the Transformation that optimizes the metric.

The registration method also support a generic optimizer that can be selected at run-time. The only restriction for the optimizer is that it should be able to operate in single-valued cost functions given that the metrics used to compare images provide a single value as output.

The terms : Fixed image and Moving image are used in this class to indicate what image is being mapped by the transform.

This class uses the coordinate system of the Fixed image as a reference and searches for a Transform that will map points from the space of the Fixed image to the space of the Moving image.

For doing so, a Metric will be continuously applied to compare the Fixed image with the Transformed Moving image. This process also requires to interpolate values from the Moving image.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Global Registration Of Two Images
- Global Registration Two Images (Affine)
- Global Registration Of Two Images (BSpline)

See itk::ImageRegistrationMethod for additional documentation.

### Mutual Information Affine

### Mutual Information Metric

The `MutualInformationImageToImageMetric` class computes the mutual information between two images, i.e. the degree to which information content in one image is dependent on the other image. This example shows how `MutualInformationImageToImageMetric` can be used to map affine transformation parameters and register two images using a gradient ascent algorithm.

[1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from urllib.request import urlretrieve

import itk
from itkwidgets import compare, checkerboard
```

### Retrieve fixed and moving images for registration

We aim to register two slice images, one of which has an arbitrary offset and rotation. We seek to use an affine transform to appropriately rotate and translate the moving image to register with the fixed image.

[2]:
```
fixed_image_path = 'fixed.png'
moving_image_path = 'moving.png'
```

[3]:
```
if not os.path.exists(fixed_image_path):
    url = 'https://data.kitware.com/api/v1/file/602c10a22fa25629b97d2896/download'
    urlretrieve(url, fixed_image_path)
if not os.path.exists(moving_image_path):
    url = 'https://data.kitware.com/api/v1/file/602c10a32fa25629b97d28a0/download'
    urlretrieve(url, moving_image_path)
```

[4]:
```
fixed_image = itk.imread(fixed_image_path, itk.F)
moving_image = itk.imread(moving_image_path, itk.F)
```

[5]:
```
checkerboard(fixed_image, moving_image)
```

```
VBox(children=(Viewer(annotations=False, interpolation=False, rendered_image=<itk.
↪itkImagePython.itkImageF2; p...
```

### Prepare images for registration

[6]:
```
ImageType = type(fixed_image)
```

[7]:
```
fixed_normalized_image = itk.normalize_image_filter(fixed_image)
fixed_smoothed_image = itk.discrete_gaussian_image_filter(fixed_normalized_image,␣
↪variance=2.0)

moving_normalized_image = itk.normalize_image_filter(moving_image)
moving_smoothed_image = itk.discrete_gaussian_image_filter(moving_normalized_image,␣
↪variance=2.0)
```

[8]:
```
compare(fixed_smoothed_image, moving_smoothed_image)
```

```
AppLayout(children=(HBox(children=(Label(value='Link:'), Checkbox(value=False,␣
↪description='cmap'), Checkbox(v...
```

### Plot the `MutualInformationImageToImageMetric` surface

For this relatively simple example we seek to adjust only the x- and y-offset of the moving image with a `TranslationTransform`. We can acquire `MutualInformationImageToImageMetric` values comparing the two images at many different possible offset pairs with `ExhaustiveOptimizer` and visualize this data set as a surface with `matplotlib`.

The affine transform contains six parameters representing each element in an affine matrix `A` which will dictate how the moving image is sampled. We know that the moving image has been translated so we will visualize the two translation parameters, but we could set `X_INDEX` and `Y_INDEX` to visualize any pair of parameters. See https://en.wikipedia.org/wiki/Affine_transformation#Image_transformation for more information on affine transformations.

[9]:
```
X_INDEX = 4    # Translation in the X direction
Y_INDEX = 5    # Translation in the Y direction
```

[10]:
```
# Move at most 20 pixels away from the initial position
window_size = [0] * 6
window_size[X_INDEX] = 20     # Set lower if visualizing elements 0-3
window_size[Y_INDEX] = 20     # Set lower if visualizing elements 0-3
```

(continues on next page)

(continued from previous page)

```
# Collect 50 steps of data along each axis
n_steps = [0] * 6
n_steps[X_INDEX] = 50
n_steps[Y_INDEX] = 50
```

[11]:
```
dim = fixed_image.GetImageDimension()

TransformType = itk.AffineTransform[itk.D,dim]
transform = TransformType.New()
```

[12]:
```
InterpolatorType = itk.LinearInterpolateImageFunction[ImageType, itk.D]
interpolator = InterpolatorType.New()
```

[13]:
```
MetricType = itk.MutualInformationImageToImageMetric[ImageType, ImageType]
metric = MetricType.New()

metric.SetNumberOfSpatialSamples(100)
metric.SetFixedImageStandardDeviation(5.0)
metric.SetMovingImageStandardDeviation(5.0)

metric.ReinitializeSeed(121212)
```

[14]:
```
ExhaustiveOptimizerType = itk.ExhaustiveOptimizer
optimizer = ExhaustiveOptimizerType.New()

# Map out [n_steps] in each direction
optimizer.SetNumberOfSteps(n_steps)

# Move [window_size / n_steps] units with every step
scales = optimizer.GetScales()
scales.SetSize(6)

for i in range(0,6):
    scales.SetElement(i, (window_size[i] / n_steps[i]) if n_steps[i] != 0 else 1)

optimizer.SetScales(scales)
```

[15]:
```
# Collect data describing the parametric surface with an observer
surface = dict()

def print_iteration():
    surface[tuple(optimizer.GetCurrentPosition())] = optimizer.GetCurrentValue()

optimizer.AddObserver(itk.IterationEvent(), print_iteration)
```

[15]:
```
0
```

[16]:
```
RegistrationType = itk.ImageRegistrationMethod[ImageType, ImageType]
registrar = RegistrationType.New()

registrar.SetFixedImage(fixed_smoothed_image)
registrar.SetMovingImage(moving_smoothed_image)
registrar.SetOptimizer(optimizer)
registrar.SetTransform(transform)
registrar.SetInterpolator(interpolator)
```

(continues on next page)

(continued from previous page)

```
registrar.SetMetric(metric)

registrar.SetFixedImageRegion(fixed_image.GetBufferedRegion())
registrar.SetInitialTransformParameters(transform.GetParameters())
```

[17]:
```
registrar.Update()
```

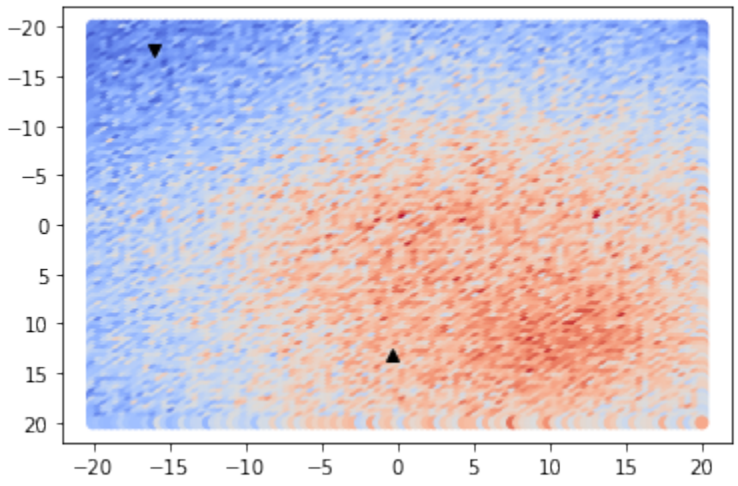
[18]:
```
# Check the extreme positions within the observed window
max_position = list(optimizer.GetMaximumMetricValuePosition())
min_position = list(optimizer.GetMinimumMetricValuePosition())

max_val = optimizer.GetMaximumMetricValue()
min_val = optimizer.GetMinimumMetricValue()

print(max_position)
print(min_position)
```

```
[1.0, 0.0, 0.0, 1.0, -0.4, 13.200000000000001]
[1.0, 0.0, 0.0, 1.0, -16.0, -17.6]
```

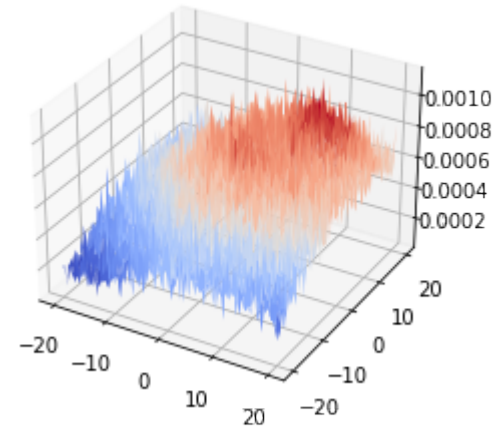
[19]:
```
# Set up values for the plot
x_vals = [list(set([x[i]
                for x in surface.keys()])) for i in range(0,transform.
↪GetNumberOfParameters())]

for i in range(0, transform.GetNumberOfParameters()):
    x_vals[i].sort()

X, Y = np.meshgrid(x_vals[X_INDEX], x_vals[Y_INDEX])
Z = np.array([[surface[(1,0,0,1,x0,x1)] for x1 in x_vals[X_INDEX]]for x0 in x_vals[Y_
↪INDEX]])
```

[20]:
```
# Plot the surface as a 2D heat map
fig = plt.figure()

# Invert the y-axis to represent the image coordinate system
plt.gca().invert_yaxis()
ax = plt.gca()

surf = ax.scatter(X, Y, c=Z, cmap=cm.coolwarm)

# Mark extremes on the plot
ax.plot(max_position[X_INDEX],max_position[Y_INDEX],'k^')
ax.plot(min_position[X_INDEX],min_position[Y_INDEX],'kv')
```

[20]:
```
[<matplotlib.lines.Line2D at 0x2a56ed10d90>]
```




[21]:
```
# Plot the surface as a 3D scatter plot
fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X,Y,Z,cmap=cm.coolwarm)
```




### Follow gradient ascent

Once we understand the shape of the parametric surface it is easier to visualize the gradient ascent algorithm. We see that there is some roughness to the surface, but it has a clear slope upwards. We want to maximize the mutual information between the two images in order to optimize registration. The results of gradient ascent optimization can be superimposed onto the `matplotlib` plot.

[22]:
```
transform = TransformType.New()
interpolator = InterpolatorType.New()
```

[23]:
```
metric = MetricType.New()

metric.SetNumberOfSpatialSamples(100)
```

(continues on next page)

(continued from previous page)

```
metric.SetFixedImageStandardDeviation(5.0)
metric.SetMovingImageStandardDeviation(5.0)

metric.ReinitializeSeed(121212)
```

[24]:
```
n_iterations = 200

optimizer = itk.GradientDescentOptimizer.New()

optimizer.SetLearningRate(1.0)
optimizer.SetNumberOfIterations(n_iterations)
optimizer.MaximizeOn()

# Set scales so that the optimizer can take
# large steps along translation parameters,
# moderate steps along rotational parameters, and
# small steps along scale parameters
optimizer.SetScales([100,0.5,0.5,100,0.0001,0.0001])
```

[25]:
```
descent_data = dict()
descent_data[0] = (1,0,0,1,0,0)

def log_iteration():
    descent_data[optimizer.GetCurrentIteration() + 1] = tuple(optimizer.
↪GetCurrentPosition())

optimizer.AddObserver(itk.IterationEvent(), log_iteration)
```

[25]:
```
0
```

[26]:
```
registrar = RegistrationType.New()
registrar.SetFixedImage(fixed_smoothed_image)
registrar.SetMovingImage(moving_smoothed_image)
registrar.SetTransform(transform)
registrar.SetInterpolator(interpolator)
registrar.SetMetric(metric)
registrar.SetOptimizer(optimizer)

registrar.SetFixedImageRegion(fixed_image.GetBufferedRegion())
registrar.SetInitialTransformParameters(transform.GetParameters())
```

[27]:
```
registrar.Update()
```

[28]:
```
print(f'Its: {optimizer.GetCurrentIteration()}')
print(f'Final Value: {optimizer.GetValue()}')
print(f'Final Position: {list(registrar.GetLastTransformParameters())}')
```

```
Its: 200
Final Value: 0.0007345867979511311
Final Position: [1.0029839517261592, -0.010245644582257242, -0.01641632700222059, 1.
↪0032415646265085, 13.10655567089651, 12.761316915069711]
```

[29]:
```
descent_data
```

[29]:
```
{0: (1, 0, 0, 1, 0, 0),
 1: (0.999999965527849,
```

(continues on next page)

(continued from previous page)

```
  -0.0016629375454765493,
  -0.0019468854493280953,
  1.0000030221155434,
  0.0070724728333067316,
  -0.03606097600487626),
 2: (1.0000158649938202,
  0.0017875012247273974,
  0.0014649178874578785,
  1.000018244613104,
  0.16180008682720395,
  0.11360707328765127),
 3: (1.0000425997042064,
  0.0032077425755804466,
  -0.0021841309405414124,
  1.0000041473195547,
  0.22605434788625947,
  0.0009694550761079751),
 4: (1.0000753342618283,
  0.008455524952988737,
  -0.00010225287048921733,
  1.0000301796832407,
  0.4832979753254426,
  0.13790535364603634),
 5: (1.0000557531623095,
  0.0016169386175090023,
  0.0001399946290998774,
  1.0000317304876585,
  0.3263146246837202,
  0.2067871716700091),
 6: (1.0000514289215043,
  0.0013991846942296771,
  0.0034409916360437867,
  1.0000603451418513,
  0.2547503259637926,
  0.32331389761150553),
 7: (1.0000390671368042,
  -0.00464084624417071,
  0.0020064603033397776,
  1.0000567323503309,
  0.0867925370379804,
  0.26824463962805783),
 8: (1.0000323035713792,
  -0.006548715918176186,
  0.008161116356050421,
  1.000107659902533,
  0.062976923187307,
  0.5532678365738305),
 9: (1.0000427625470925,
  0.0009210905481415264,
  0.012503459911795356,
  1.0001106920885776,
  0.24221370589392452,
  0.6564395759394801),
 10: (1.0000644916142147,
  0.01134327391351616,
  0.012233302020000162,
  1.000103016019471,
```

(continues on next page)

(continued from previous page)

```
 0.5229809457512343,
 0.5915683378194321),
11: (1.0000518070970936,
 0.009506212296739217,
 0.009755186043286826,
 1.0000965159502793,
 0.4786979296746874,
 0.5341487982419897),
12: (1.0000632855970673,
 0.012466891904789178,
 0.012024568878491737,
 1.0000992466585101,
 0.5905852114954294,
 0.6268634183585254),
13: (1.0000373290348534,
 0.01101344325538434,
 0.014841027517697895,
 1.000112473200331,
 0.4987265352644986,
 0.7418373068560897),
14: (1.0000546866010733,
 0.015493937057581212,
 0.01666651384068458,
 1.000132765035817,
 0.6463833982580928,
 0.8080911676910822),
15: (1.0000631867012741,
 0.01342235419293244,
 0.01928722841564486,
 1.0001677388801908,
 0.6313075016185806,
 0.9504706426545657),
16: (1.0000658347903864,
 0.013760347690960288,
 0.020309308864496344,
 1.0001802914015183,
 0.6381874605855995,
 1.0074894422852676),
17: (1.0001104781926822,
 0.02089549688220317,
 0.01754713377932798,
 1.0001855726881073,
 0.9287857148328786,
 0.9338002039563951),
18: (1.0001122807183165,
 0.017763036125256645,
 0.020640107973677255,
 1.0002135832291803,
 0.8593389248830472,
 1.105353339897966),
19: (1.000122821487329,
 0.018582231918642292,
 0.020335068820269128,
 1.0002246377256918,
 0.8666517882674349,
 1.1110164493119412),
20: (1.0001273603009995,
```

(continues on next page)

(continued from previous page)

```
  0.021960842879295686,
  0.021033407917090154,
  1.000232420518082,
  1.002845683269043,
  1.1684552062025277),
 21: (1.0001397514011425,
  0.024740142447823087,
  0.022433020183041844,
  1.000244763827763,
  1.129079120734455,
  1.2430975064681058),
 22: (1.0001223435989122,
  0.02167390074035541,
  0.02413304640059116,
  1.000261266375767,
  1.0284992108314137,
  1.3138294318612544),
 23: (1.0001231381137234,
  0.01882320833012633,
  0.025686464118271607,
  1.0002857117503006,
  0.9582982738986365,
  1.4011972381355713),
 24: (1.0001077910687561,
  0.016391824328981664,
  0.02561497988662803,
  1.0003018794849772,
  0.8709406445656998,
  1.4167659128852281),
 25: (1.0001088003263428,
  0.014991274941814237,
  0.02713568191592862,
  1.0003164948852543,
  0.833210537074188,
  1.505187689416831),
 26: (1.000112280135333,
  0.013455063290054583,
  0.027778099818430543,
  1.0003439451387963,
  0.765573189118558,
  1.5669164620080727),
 27: (1.0001450154081564,
  0.0177292068447402,
  0.03261127598071813,
  1.0003734817767191,
  0.9018893930559542,
  1.7629587392841095),
 28: (1.0001478929494814,
  0.014634770850063202,
  0.031209234963315156,
  1.000390997915687,
  0.8764704378125825,
  1.7600207879040424),
 29: (1.0001700260192306,
  0.023304792919655027,
  0.03334327221311678,
  1.000415861108279,
```

(continues on next page)

(continued from previous page)

```
 1.1050624653061767,
 1.8286067547503744),
30: (1.0002238314105383,
 0.03161445006384011,
 0.030894925266538815,
 1.00042125420657,
 1.48370570704743,
 1.753463644918086),
31: (1.000267445438375,
 0.03519002134376226,
 0.03206699672260963,
 1.0004408361991857,
 1.637502178426234,
 1.8135444785593309),
32: (1.0002925314787032,
 0.038194063131100985,
 0.03175449876971995,
 1.0004437954970606,
 1.8007953543880255,
 1.8203698944795941),
33: (1.0003377638901942,
 0.04502109006274137,
 0.034993001930344464,
 1.0004917167239793,
 2.0943103237531275,
 2.0200348747122416),
34: (1.0003803879833888,
 0.04979456747358726,
 0.037166416447014544,
 1.0005113084616988,
 2.262996533539432,
 2.138944755461071),
35: (1.0003964256083064,
 0.05117236577832922,
 0.036139025055172636,
 1.000517255314559,
 2.3653462187910086,
 2.149530830319958),
36: (1.0004360747618182,
 0.05345144578045517,
 0.041353120222081374,
 1.000574529441919,
 2.506749317525062,
 2.4517033740702763),
37: (1.0004800273917174,
 0.055476026763283504,
 0.03497869525349116,
 1.0005694882764171,
 2.699417361085486,
 2.2820352287186982),
38: (1.0005206995135523,
 0.05935785756284043,
 0.03680599549073893,
 1.00059995447889,
 2.8314656220050343,
 2.379800641551389),
39: (1.0005597309781562,
```

(continues on next page)

(continued from previous page)

```
 0.0629519671554236,
 0.03286977985068564,
 1.0006061112248472,
 3.0482611975454437,
 2.300140204936005),
40: (1.0005771303435242,
 0.06359760525832305,
 0.03084883069424145,
 1.0006276527065008,
 3.094388142005032,
 2.302490335848059),
41: (1.0005531002842671,
 0.05406694461103779,
 0.03197192149481904,
 1.0006410504293661,
 2.8277726043789086,
 2.4080371348532528),
42: (1.0005540699748965,
 0.0499425356669065,
 0.03146040367046596,
 1.000651531523379,
 2.7609765291414368,
 2.4179316449263375),
43: (1.000610737465348,
 0.055348765756564236,
 0.033964422597629856,
 1.0006648447876705,
 3.0028005303581273,
 2.5357804191677906),
44: (1.0006330623321382,
 0.0604441118756231,
 0.03541207386386212,
 1.0006892015344302,
 3.1729606591188624,
 2.6409718876715735),
45: (1.0006442795064596,
 0.06054527728396184,
 0.034876033570904005,
 1.0006762494684076,
 3.2010553258318533,
 2.613460581514949),
46: (1.000662149541532,
 0.0607445636408164,
 0.03813667098198882,
 1.0007125957729466,
 3.2382560865404764,
 2.770594834115738),
47: (1.0006729481736498,
 0.060918231289935475,
 0.03233310861663754,
 1.0006972055901309,
 3.317069230559112,
 2.615928702093721),
48: (1.0006671735013677,
 0.058603049816866676,
 0.031921242837147284,
 1.0007020569642333,
```

(continues on next page)

(continued from previous page)

```
 3.2015939820683537,
 2.5920285219449837),
49: (1.0006967794678154,
 0.062351381457950376,
 0.03219102024577858,
 1.0007204433036898,
 3.389920656675441,
 2.6347548227953634),
50: (1.0007074104619513,
 0.06005071670104204,
 0.03276782470243937,
 1.0007309277353125,
 3.35628336706342,
 2.7091459084586655),
51: (1.0007396923199825,
 0.061291160665742646,
 0.030129086972525133,
 1.0007287540495768,
 3.491521638128275,
 2.6836909479579494),
52: (1.0007572355137908,
 0.05960947091978929,
 0.03198540406396382,
 1.000758633658917,
 3.5365386031695225,
 2.8431750698970517),
53: (1.0007798845923912,
 0.06509119967730434,
 0.035952863624348,
 1.000789187579904,
 3.7494949981560985,
 3.0473717852738775),
54: (1.0007814735197 86,
 0.060341197508327865,
 0.035722251156929986,
 1.0007937977676336,
 3.6918683782560695,
 3.1184458997851303),
55: (1.0007872554502402,
 0.06040160758394653,
 0.03518786014563553,
 1.000800477687533,
 3.7135825408548624,
 3.1342223383341103),
56: (1.0008145677249618,
 0.06452951906037036,
 0.03734410678688241,
 1.0008181568848529,
 3.935213218074109,
 3.3080838931206684),
57: (1.0008140957881748,
 0.06490597049903526,
 0.037054637041254314,
 1.0008340727056613,
 3.9740722322538815,
 3.342867380839161),
58: (1.0008180551607286,
```

(continues on next page)

(continued from previous page)

```
 0.061866698794218127,
 0.03526112066702049,
 1.0008347154048034,
 3.8996560917193013,
 3.2997991235710646),
59: (1.0008184915440024,
 0.05810194800622753,
 0.032078749472733536,
 1.0008174040415063,
 3.8702246971271292,
 3.275480122016447),
60: (1.0008289755868198,
 0.05760769157075074,
 0.035245271559863914,
 1.000846651219139,
 3.902919247486544,
 3.47761867695573),
61: (1.0008633410589363,
 0.05572871476066318,
 0.031553091525065226,
 1.0008579228538625,
 3.990726748079648,
 3.448824699625424),
62: (1.0008784378417017,
 0.05266122941402723,
 0.03264443802534106,
 1.0009120469223736,
 3.978242611217881,
 3.600547321798405),
63: (1.0008928890970574,
 0.0526558541734464,
 0.033503931295208476,
 1.0009317713690227,
 4.034332325489235,
 3.6886036941446476),
64: (1.0009139729436916,
 0.055303340473382784,
 0.037179269502224506,
 1.0009491506310282,
 4.143989163518143,
 3.8492430670691165),
65: (1.0009452007831592,
 0.05673317055934501,
 0.03769485457324463,
 1.0009815413572847,
 4.3005043187866026,
 3.9354929724543823),
66: (1.000964661596496,
 0.056193617300998526,
 0.038228221343659385,
 1.0009985420720955,
 4.391974061573027,
 4.047226171582918),
67: (1.0009489516390504,
 0.05131866362736038,
 0.034655435593925296,
 1.0009733373448018,
```

(continues on next page)

(continued from previous page)

```
 4.292328227469166,
 3.9568703108886076),
68: (1.0009413430480196,
 0.04894249095833897,
 0.03671924896600889,
 1.0010030560844039,
 4.25078789548804,
 4.118073144864641),
69: (1.0009619851850104,
 0.04729950161388283,
 0.03709263153386867,
 1.0010230937192157,
 4.281357081635324,
 4.2287662477296895),
70: (1.0009883423387873,
 0.048950062105423985,
 0.038046513712827364,
 1.0010597645316839,
 4.430992282208885,
 4.34743407298929),
71: (1.0010137638710939,
 0.05336056386113549,
 0.03689130248075374,
 1.001072749055473,
 4.671966314989308,
 4.356481292663306),
72: (1.0010035331560332,
 0.051521442717293024,
 0.03407454202526072,
 1.001069535774639,
 4.675948704401148,
 4.318943572622043),
73: (1.0010133467070166,
 0.05025409278757773,
 0.03552744416230726,
 1.0010991581691187,
 4.706984724922817,
 4.469211006294377),
74: (1.00103872421304,
 0.04799225072152847,
 0.034678317190555384,
 1.00111106699047,
 4.747261791561516,
 4.522030002434047),
75: (1.0010537945165752,
 0.048916122961739084,
 0.03456239062299948,
 1.0011399296717824,
 4.893172777587763,
 4.678249007363964),
76: (1.0010708116503542,
 0.04966257392911915,
 0.03151892711151409,
 1.0011406407015644,
 4.999013958146794,
 4.606774486089828),
77: (1.0011074021693485,
```

(continues on next page)

(continued from previous page)

```
 0.05330094656215735,
 0.02955549065724992,
 1.001148211132933,
 5.168830012564221,
 4.591695110628777),
78: (1.0011180403178352,
 0.04911265606635972,
 0.028874066097180205,
 1.0011697779064674,
 5.096506430676856,
 4.655314137512003),
79: (1.0011304572059416,
 0.048411502402000844,
 0.026947412173529944,
 1.0011677592034094,
 5.152398101773921,
 4.635063604691433),
80: (1.0011534166529035,
 0.045533538723998894,
 0.0247009249692374,
 1.0011741027229923,
 5.167592764506425,
 4.649732789182321),
81: (1.0011874751177516,
 0.04608734927115699,
 0.023052334706844004,
 1.0011910423241892,
 5.316733019992377,
 4.7044779053144365),
82: (1.0012216530417395,
 0.051030729738921046,
 0.024090469721467323,
 1.0012143207041417,
 5.5741717336269865,
 4.84196772524597),
83: (1.0012155910218057,
 0.047979595405993385,
 0.02060019659317837,
 1.001201637958086,
 5.571806837673036,
 4.7889465009998675),
84: (1.0012275595939713,
 0.049104395383112634,
 0.01985796096916095,
 1.0012032163768907,
 5.640324279276126,
 4.803288119178803),
85: (1.0012314523542247,
 0.050020319896353506,
 0.016917352746062826,
 1.001191677914485,
 5.66305474131709,
 4.696474927128242),
86: (1.0012231554137863,
 0.0459316553389,
 0.0172195732646551,
 1.001202043874071,
```

(continues on next page)

(continued from previous page)

```
 5.567941410443212,
 4.762817321161207),
87: (1.0012255448411016,
 0.04368101466938556,
 0.01981615508009791,
 1.0012343996433413,
 5.527174476496044,
 4.925926300210495),
88: (1.0012438685165126,
 0.04536241517724545,
 0.020819924588875043,
 1.001243630091461,
 5.64088706315316,
 5.008415816496351),
89: (1.0012529681222695,
 0.04500445744378577,
 0.01825969037534871,
 1.0012471442579247,
 5.673998014080022,
 4.943213170012537),
90: (1.0012466479207036,
 0.04301406713437193,
 0.02062079465068866,
 1.0012772145535302,
 5.64032722398762,
 5.0967382031312),
91: (1.0012702065173769,
 0.03919037399383151,
 0.018848213311686898,
 1.0013013012804877,
 5.65402825580481,
 5.146696157604204),
92: (1.0013122519844961,
 0.045587033304291084,
 0.01942914361498954,
 1.0013390244055498,
 5.9799874568084155,
 5.280096943144681),
93: (1.0013238518956937,
 0.04726847057600322,
 0.01746131636518727,
 1.0013366753312876,
 6.059948204069908,
 5.243347406334709),
94: (1.0013432211440945,
 0.05017160839262471,
 0.017760160273959325,
 1.0013385900967755,
 6.227867299852726,
 5.336827192070069),
95: (1.00138194743515,
 0.05338162142606548,
 0.019636002898359105,
 1.0013632592399198,
 6.45888100317004,
 5.5164338840783491),
96: (1.001381582879274,
```

(continues on next page)

(continued from previous page)

```
 0.049469137405552234,
 0.019300379997726096,
 1.0013737444800777,
 6.390238529153018,
 5.574935374902547),
97: (1.0013613147845113,
 0.042036458090255586,
 0.017203201204209487,
 1.0013796450507715,
 6.26251297746609,
 5.597725239436271),
98: (1.0013623974055619,
 0.03976736948692073,
 0.01941599991203298,
 1.0014089134131836,
 6.249321082440559,
 5.762854116216476),
99: (1.00139726526146,
 0.04412004516156002,
 0.02573942288520286,
 1.0014562562514249,
 6.487622112286879,
 6.101108327904233),
100: (1.0014010328638006,
 0.0379669742778653,
 0.024572631664173908,
 1.0014628283539142,
 6.383539464934111,
 6.150896352637763),
101: (1.0014149262114136,
 0.041378435781837794,
 0.026972688564634934,
 1.0014742044756928,
 6.4906024381097795,
 6.2622292763551926),
102: (1.0014482705832934,
 0.04645118396678878,
 0.02718414409075883,
 1.0014865399033572,
 6.738645539197731,
 6.332576271079053),
103: (1.001441702134815,
 0.03742094561902583,
 0.023041360279085946,
 1.0014959164165806,
 6.614978071563363,
 6.3458020769269),
104: (1.001470899238623,
 0.03996662305978739,
 0.022611364691556567,
 1.0015023160362937,
 6.838050131268682,
 6.410886597716072),
105: (1.0014867384509618,
 0.04013115597118111,
 0.02211848469257116,
 1.0015260855565642,
```

(continues on next page)

(continued from previous page)

```
 6.923321358271605,
 6.507334867695091),
106: (1.0014829435376351,
 0.034595999005676836,
 0.019201735339162744,
 1.0015291577499232,
 6.884027415499893,
 6.5419377689552745),
107: (1.001489794752776,
 0.03018596527246648,
 0.017791319153445343,
 1.0015427911719266,
 6.851726145955262,
 6.61478836386323),
108: (1.0014715489793686,
 0.022633193203852528,
 0.019015449479443336,
 1.001570165378783,
 6.620603357920789,
 6.740712292523077),
109: (1.001500162701251,
 0.026487924533173713,
 0.019990747350695937,
 1.0015989212740382,
 6.856752942813844,
 6.867842765389764),
110: (1.0015509438817778,
 0.027242278092531422,
 0.020362279700267927,
 1.0016415012953963,
 7.027932705213258,
 7.005081363564973),
111: (1.0015668815656489,
 0.029448817637910765,
 0.0232755679407571,
 1.0016699408100305,
 7.166489885024689,
 7.183240996636877),
112: (1.001574210620731,
 0.02823845267609452,
 0.023834445539348315,
 1.0016775695764397,
 7.21961240167998,
 7.277345275352267),
113: (1.0015797695672788,
 0.027537454115571566,
 0.024680652071500355,
 1.001706752910666,
 7.295887091777702,
 7.431088809402569),
114: (1.0015876297889528,
 0.021349873406022995,
 0.028117663094964662,
 1.0017499399414185,
 7.238044790518791,
 7.706090230472631),
115: (1.001637058151661,
```

(continues on next page)

(continued from previous page)

```
 0.026778711088343902,
 0.018953030939650603,
 1.0017309729368442,
 7.644316425332046,
 7.464771623412593),
116: (1.0016818510558196,
 0.032364129202914406,
 0.016941239211920995,
 1.0017314468890584,
 7.940030314212486,
 7.438644962181216),
117: (1.0017071445778736,
 0.02628104344114241,
 0.013359352125705673,
 1.001750707808061,
 7.891921812119137,
 7.433340341929363),
118: (1.00175125155169,
 0.028049322362313462,
 0.011754147243330158,
 1.0017767127016353,
 8.109698590771947,
 7.5255627143892365),
119: (1.0017691132097544,
 0.02532885173191134,
 0.015227420589668899,
 1.0018401096480538,
 8.13860084006787,
 7.831474719459007),
120: (1.0017546789884808,
 0.018737382269375946,
 0.008593788821656012,
 1.0018283305349545,
 8.038537145215951,
 7.670300782551979),
121: (1.001769085060014,
 0.017198352238636284,
 0.0074900756827615276,
 1.0018508239437725,
 8.07146720613475,
 7.728350718696938),
122: (1.00180308725956,
 0.017078727318170013,
 0.0026358144462596423,
 1.001852102837873,
 8.171412048709891,
 7.631328646149379),
123: (1.0018450596501052,
 0.025174423895821592,
 0.003803416383942778,
 1.0018802855336497,
 8.508271553555526,
 7.7301143138514465),
124: (1.001868716780902,
 0.027869945112617138,
 0.006898295442192711,
 1.0019098292467803,
```

(continues on next page)

(continued from previous page)

```
 8.674463968869048,
 7.961878222634894),
125: (1.0018975943961956,
 0.02555863942469917,
 0.007203020911408353,
 1.0019416902575486,
 8.719008720267837,
 8.086630186820488),
126: (1.0019321075344925,
 0.03025701602610059,
 0.00977908516863384,
 1.0019896055669617,
 8.980513235319998,
 8.302325562149456),
127: (1.0019434802840155,
 0.02641931134636337,
 0.006890119948983443,
 1.0020049643264584,
 9.013813156726519,
 8.331631036133144),
128: (1.0019649443283094,
 0.02481934696813929,
 0.008700023842445207,
 1.0020214993877548,
 9.052787545162385,
 8.468835714061086),
129: (1.001977466045828,
 0.025602368331860595,
 0.007656998926756687,
 1.0020228684827135,
 9.264991814102144,
 8.570812050566111),
130: (1.0019880400527548,
 0.02366754833492131,
 0.0049214420606852 71,
 1.0020199403676828,
 9.2276099142676,
 8.492107473158196),
131: (1.002012808656276,
 0.023067581783013637,
 0.0013147090602290477,
 1.0020134557439981,
 9.281250255733404,
 8.395547033640439),
132: (1.0020142931848017,
 0.016937804173622506,
 0.0027107467356588116,
 1.0020422575863144,
 9.193813887839951,
 8.607520248931714),
133: (1.0020566752649949,
 0.02470093527526523,
 0.0053171400098446986,
 1.0020894229051642,
 9.50803108334762,
 8.808864489497951),
134: (1.0020671798303653,
```

(continues on next page)

(continued from previous page)

```
 0.022780214040542982,
 0.004839560672355789,
 1.002108157932218,
 9.580582232353557,
 8.913717226204195),
135: (1.0020545961823053,
 0.010831889106925218,
 0.0007168892147971983,
 1.0021025981677067,
 9.310006256558985,
 8.883731253770843),
136: (1.0020819029166785,
 0.012982215474751485,
 0.0032605154650179467,
 1.0021524524449537,
 9.490352307116236,
 9.09744727805679),
137: (1.002109180964236,
 0.015690414259106136,
 0.002035795364862914,
 1.0021686192761907,
 9.653761410090535,
 9.116800533432912),
138: (1.0021347563864977,
 0.012775032716776897,
 0.002241634122103183,
 1.0022122853506348,
 9.706648627847356,
 9.301233114588095),
139: (1.0021467645240343,
 0.012392837436219047,
 0.004974770308221652,
 1.002254247595866,
 9.808506194749773,
 9.517054086232006),
140: (1.0021531263959906,
 0.011454319092641213,
 0.005968666978669034,
 1.0022691603170968,
 9.841185810692997,
 9.615961196728737),
141: (1.0021636129379816,
 0.012860474410313537,
 0.00716679477386708,
 1.0022844451757713,
 9.9334776367762,
 9.736766019305797),
142: (1.0021717067834552,
 0.0077128345348900906,
 0.00505315994133678,
 1.0022823628079773,
 9.858195652628702,
 9.724085749556073),
143: (1.0021957169862827,
 0.00815006081402336,
 0.006964109380848566,
 1.0023104271672316,
```

(continues on next page)

(continued from previous page)

```
 9.937004586752606,
 9.886346442920633),
144: (1.0022227436436295,
 0.009124131247707395,
 0.004897834540664515,
 1.0023198603308,
 10.082870304596783,
 9.882098531290065),
145: (1.002244660234744,
 0.010330327697804757,
 0.004619950757925363,
 1.0023376700099085,
 10.211732957604358,
 9.938269227639653),
146: (1.0022509357955849,
 0.0045462724061707165,
 0.003207062851745687,
 1.0023499522238073,
 10.109823367661134,
 10.007119784592303),
147: (1.0022897146001202,
 0.007009077859381334,
 0.001905690288132529,
 1.0023603969312305,
 10.249674556962448,
 10.004899338281737),
148: (1.0023307339375538,
 0.011854440472387397,
 0.001395537500113279,
 1.0023836674565736,
 10.562623227085258,
 10.091395779709025),
149: (1.0023491080827485,
 0.010023367509582841,
 0.0016006347717150862,
 1.002405795274947,
 10.569876775313933,
 10.170316462493894),
150: (1.0023415302697172,
 0.005033589388035417,
 0.0030580317013123187,
 1.002433510493446,
 10.46334590307692,
 10.288912199920812),
151: (1.002355187464412,
 0.007438562624094941,
 0.0014666261671097383,
 1.002449103175267,
 10.57800832871742,
 10.29096938129903),
152: (1.0023779639797088,
 0.006611598905641159,
 -0.0009694616343345389,
 1.0024603191385222,
 10.647795929263976,
 10.268152947214265),
153: (1.0023790503810388,
```

(continues on next page)

(continued from previous page)

```
  -0.0009820093817523431,
  -0.0033278558507537173,
  1.002487805648212,
  10.501452276263736,
  10.317654090534338),
 154: (1.0023897385072735,
  0.0005008292798210276,
  -0.0044510083020785985,
  1.0024859716108288,
  10.634952019200377,
  10.335173135623029),
 155: (1.0024362890140468,
  0.004387502551718853,
  -0.0035224457675955027,
  1.0025140734756244,
  10.895892979562953,
  10.447644622501505),
 156: (1.0024466941909564,
  0.004699089211273155,
  -0.0003068668122849232,
  1.002561952379878,
  10.967113915372746,
  10.683013611641053),
 157: (1.0024554494000404,
  0.003423055595332401,
  -0.0014416670133380646,
  1.0025735090184824,
  11.011909110739651,
  10.732587967512332),
 158: (1.0024954148673397,
  0.011234636500195003,
  -0.0018110726266817893,
  1.002590733166248,
  11.402080297169924,
  10.84176600097245),
 159: (1.0025051015835722,
  0.00874200203332762,
  -0.004019833937020776,
  1.0025977394525125,
  11.406249926717942,
  10.823691690204441),
 160: (1.0025570245947704,
  0.014855113112736826,
  -0.006992764462789507,
  1.0026105810429808,
  11.72866922576604,
  10.794607122723155),
 161: (1.0025359563440432,
  0.009325001855811987,
  -0.006238214456332685,
  1.002630774106633,
  11.553168385270247,
  10.872059976477777),
 162: (1.0025637628994528,
  0.009866145652299908,
  -0.005634243638170224,
  1.0026501411813078,
```

(continues on next page)

(continued from previous page)

```
 11.681333233623228,
 10.984178282247514),
163: (1.0025683451519263,
 0.008205152159930028,
 -0.006566921600087174,
 1.0026515364772985,
 11.645487917927875,
 10.976362115703154),
164: (1.0025708974079064,
 0.0036928939194679506,
 -0.005910005869683161,
 1.0026852235294212,
 11.549738995436346,
 11.130181727987276),
165: (1.0025661456923456,
 -0.0019473377976114876,
 -0.0038312347685751404,
 1.002720535989897,
 11.45810100216064,
 11.331800447991817),
166: (1.0026045020969392,
 0.0014724612997977986,
 -0.0029816498869544386,
 1.0027596099262128,
 11.671567624226059,
 11.44724569982194),
167: (1.0026179188511732,
 -0.0009988831204435283,
 -0.007349765288631139,
 1.0027608479314447,
 11.767708714535903,
 11.397670115012215),
168: (1.0026328177169597,
 -0.0012009088006586452,
 -0.005533752440411114,
 1.0028011026565333,
 11.837356199495847,
 11.54892238210282),
169: (1.0026558099372855,
 0.0016127174005692564,
 -0.004173752289507825,
 1.0028209155926329,
 12.034539276742827,
 11.665876722964216),
170: (1.0026812322471446,
 0.004288232446802556,
 -0.007083763040147426,
 1.0028234422345583,
 12.189257114560851,
 11.625397078585234),
171: (1.0026727108133495,
 4.1514506145755824e-05,
 -0.006304579706448422,
 1.0028379936273555,
 12.100920153893794,
 11.711202289518422),
172: (1.0026918374815659,
```

(continues on next page)

(continued from previous page)

```
 -0.0002504564916615083,
 -0.010624145002741621,
 1.0028441568903979,
 12.206653464299775,
 11.680096121366361),
173: (1.0027087934437167,
 -0.0006164357258983809,
 -0.011674723356247196,
 1.002850980752777,
 12.274434388035639,
 11.687624939643868),
174: (1.0027027995977735,
 -0.0040625673991487084,
 -0.013456790525997592,
 1.0028523991610334,
 12.18493805988577,
 11.668671820259437),
175: (1.002717948252488,
 -0.0019180976676498925,
 -0.012127969858136542,
 1.0028759159699396,
 12.271589527895541,
 11.75500739710454),
176: (1.002768517000085,
 0.005535949234591564,
 -0.013487806812915414,
 1.0028906393693426,
 12.608494713594943,
 11.771240908075697),
177: (1.0027568328451135,
 -0.003916800322814321,
 -0.011560820482554524,
 1.0029285915774904,
 12.357598239126128,
 11.953544412097122),
178: (1.002813683361649,
 0.001961950109245625,
 -0.013304825374824164,
 1.0029393180800068,
 12.700242065509961,
 11.956347569140418),
179: (1.00279789771569,
 -0.003193374892262785,
 -0.012880220347632582,
 1.0029579821002805,
 12.559326146592904,
 12.025573281023275),
180: (1.002802586201974,
 -0.00440998412181167,
 -0.012365375707334546,
 1.0030022965984196,
 12.576081151667733,
 12.153860219616268),
181: (1.0028171010961264,
 -0.004168638325610454,
 -0.01355736993851798,
 1.003008819069466,
```

(continues on next page)

(continued from previous page)

```
 12.67004411668296,
 12.17220077955086),
182: (1.0028336261164101,
 -0.004659945960625215,
 -0.013078205409303219,
 1.0030229419991026,
 12.731497110624838,
 12.261402751806099),
183: (1.0028604713353673,
 -0.0015658607855911643,
 -0.012098971139120134,
 1.0030481953989219,
 12.886170487293692,
 12.348160722182769),
184: (1.0028391199434779,
 -0.0060682743974767445,
 -0.012710025423094413,
 1.003051980482818,
 12.713402110572995,
 12.354469304845153),
185: (1.0028716066519192,
 -0.0022402529215429755,
 -0.010531800997446685,
 1.003065705525438,
 12.908507468934957,
 12.449980443294777),
186: (1.002835145455527,
 -0.012918649903239376,
 -0.011778209644880954,
 1.0030741856208256,
 12.589461289364138,
 12.458567659637575),
187: (1.0028520671829444,
 -0.007926316106780379,
 -0.013247294121776654,
 1.0030846276708403,
 12.742940663835762,
 12.39912214878219),
188: (1.0028780520908482,
 -0.003461006808138728,
 -0.014222028826517415,
 1.0030923876467457,
 12.886803620486718,
 12.373312430926969),
189: (1.002902656929669,
 0.000205777477017971,
 -0.014202676036452231,
 1.0031073723328114,
 13.062210917092276,
 12.446303575932543),
190: (1.0028867480131376,
 -0.005844044675561992,
 -0.013870468438909586,
 1.0031357310484912,
 12.882414795728224,
 12.56653743056387),
191: (1.0028958526262244,
```

(continues on next page)

(continued from previous page)

```
 -0.007568305207781002,
 -0.010605578847616657,
 1.0031755396313145,
 12.852062833325618,
 12.783826271634979),
192: (1.0028963479903135,
 -0.009815130463589205,
 -0.009754698885980396,
 1.0031843810024403,
 12.802034975401906,
 12.857856947593104),
193: (1.0029023334878442,
 -0.009283392377352285,
 -0.010583307768849984,
 1.0031926652781853,
 12.81404673064819,
 12.846984324494825),
194: (1.0029502435355997,
 -0.001769953603955812,
 -0.01280460451576811,
 1.0031941136052003,
 13.112260351920053,
 12.77927424946506),
195: (1.0029537393766395,
 -0.0034345249773231144,
 -0.013593367756741392,
 1.0031937693307251,
 13.082299346551222,
 12.72771859751158),
196: (1.0029523340382582,
 -0.006568016544511244,
 -0.011275040609366172,
 1.0032218753367068,
 13.016930425149347,
 12.874030691163242),
197: (1.0029877604510269,
 -0.0027191514271736107,
 -0.010622643176869562,
 1.0032301481911745,
 13.232374787855136,
 12.919664502155081),
198: (1.0029982208042074,
 -0.0048912418557315535,
 -0.007496627803292464,
 1.003262644468391,
 13.18926787990394,
 13.085748816506415),
199: (1.0029896781839711,
 -0.00807390329113471,
 -0.01401479995489551,
 1.0032479288751739,
 13.162486728355939,
 12.874730622712242),
200: (1.0029839517261592,
 -0.010245644582257242,
 -0.01641632700222059,
 1.0032415646265085,
```

(continues on next page)

(continued from previous page)

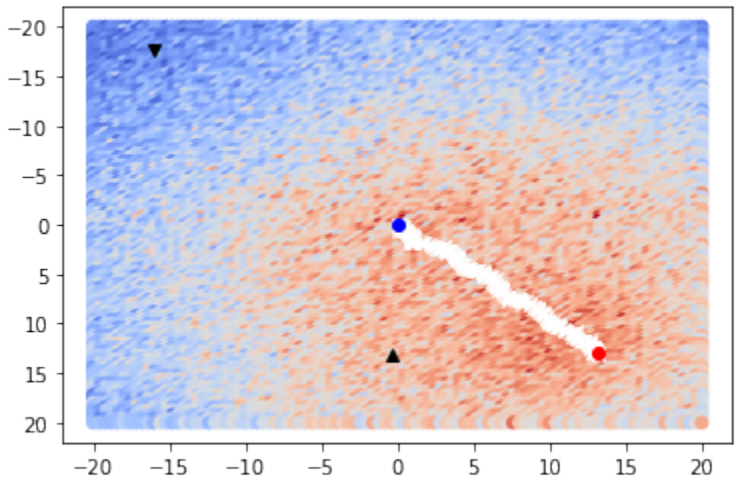
```
  13.10655567089651,
  12.761316915069711)}
```

[30]:
```
x_vals = [descent_data[i][X_INDEX] for i in range(0,n_iterations)]
y_vals = [descent_data[i][Y_INDEX] for i in range(0,n_iterations)]
```

We see in the plot that the metric generally improves as transformation parameters are updated with each iteration, but the final position may not align with the maximum position on the plot. This is one case in which it is difficult to visualize gradient ascent over a hyperdimensional space, where the optimizer is stepping through six parameter dimensions but the 2D plot we collected with `ExhaustiveOptimizer` represents a 'slice' in space with x[0:4] fixed at (1,0,0,1). Here it may be more useful to directly compare the two images after registration to evaluate fitness.

[31]:
```
fig = plt.figure()
# Note: We invert the y-axis to represent the image coordinate system
plt.gca().invert_yaxis()
ax = plt.gca()

surf = ax.scatter(X, Y, c=Z, cmap=cm.coolwarm)

for i in range(0,n_iterations-1):
    plt.plot(x_vals[i:i+2],y_vals[i:i+2],'wx-')
plt.plot(descent_data[0][X_INDEX], descent_data[0][Y_INDEX],'bo')
plt.plot(descent_data[n_iterations-1][X_INDEX],descent_data[n_iterations-1][Y_INDEX],
↪'ro')

plt.plot(max_position[X_INDEX], max_position[Y_INDEX], 'k^')
plt.plot(min_position[X_INDEX], min_position[Y_INDEX], 'kv')
```

[31]:
```
[<matplotlib.lines.Line2D at 0x2a56f1a7c70>]
```



### Resample the moving image

In order to apply the results of gradient ascent we must resample the moving image into the domain of the fixed image. The `TranslationTransform` whose parameters have been selected through gradient ascent is used to dictate how the moving image is sampled from the fixed image domain. We can compare the two images with `itkwidgets` to verify that registration is successful.

[32]:
```
ResampleFilterType = itk.ResampleImageFilter[ImageType,ImageType]
resample = ResampleFilterType.New(
    Transform=transform,
    Input=moving_image,
    Size=fixed_image.GetLargestPossibleRegion().GetSize(),
    OutputOrigin=fixed_image.GetOrigin(),
    OutputSpacing=fixed_image.GetSpacing(),
    OutputDirection=fixed_image.GetDirection(),
    DefaultPixelValue=100)
```

[33]:
```
resample.Update()
```

[34]:
```
checkerboard(fixed_image, resample.GetOutput())
```

```
VBox(children=(Viewer(annotations=False, interpolation=False, rendered_image=<itk.
↪itkImagePython.itkImageF2; p...
```

The image comparison shows that the images were successfully translated to overlap, but were not fully rotated to exactly align. If we were to explore further we could use a different optimizer with the metric, such as the `LBFGSBOptimizer` class, which may be more successful in optimizing over a rough parametric surface. We can also explore different metrics such as the `MattesMutualInformationImageToImageMetricv4` class to take advantage of the ITK v4+ registration framework, in contrast with the `MutualInformationImageToImageMetric` used in this example as part of the v3 framework.

### Clean up

[35]:
```
os.remove(fixed_image_path)
os.remove(moving_image_path)
```

### Synopsis

Global registration by maximizing the mutual information and using an affine transform.

### Results

Output:

```
Optimizer stop condition: GradientDescentOptimizer: Maximum number of iterations␣
↪(200) exceeded.
Final Parameters: [1.0028069041777101, -0.009647107048100798, -0.010717116855425006,␣
↪1.0029040091646872, 13.26279335943067, 12.226979744206531]

Result =
```

(continues on next page)

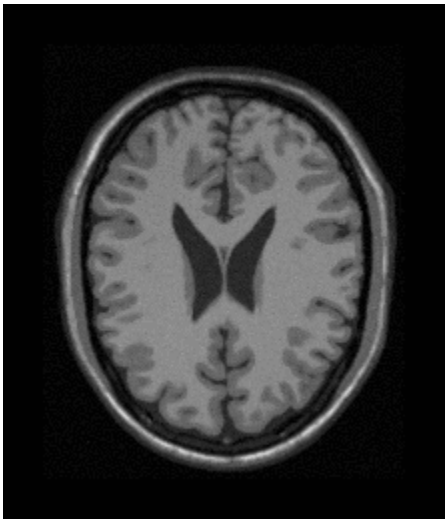


Fig. 74: Fixed

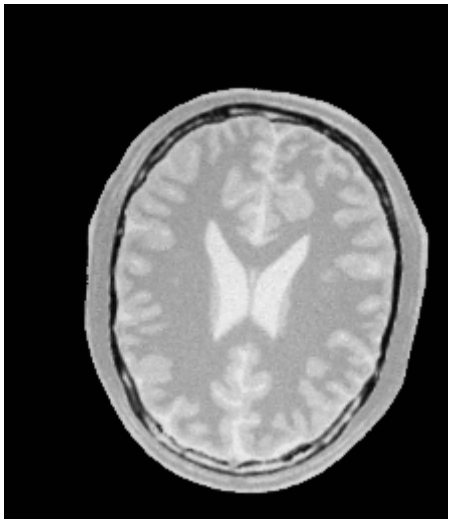


Fig. 75: Moving

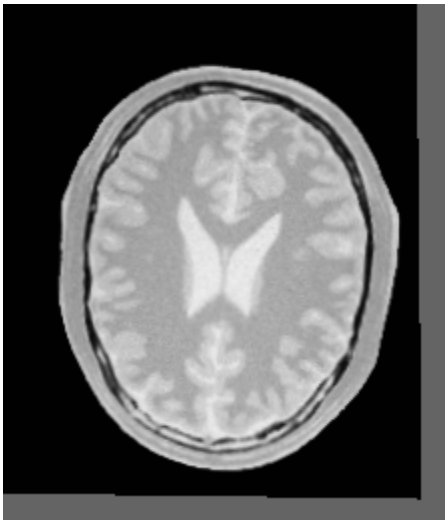


Fig. 76: Output

(continued from previous page)

```
Iterations    = 200
Metric value  = 0.000971698
Numb. Samples = 567
```

### Jupyter Notebook

launch binder

### Code

### C++

```
#include "itkImageRegistrationMethod.h"
#include "itkAffineTransform.h"
#include "itkMutualInformationImageToImageMetric.h"
#include "itkGradientDescentOptimizer.h"
#include "itkNormalizeImageFilter.h"
#include "itkDiscreteGaussianImageFilter.h"
#include "itkResampleImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkCheckerBoardImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkImageFileReader.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned char;

using ImageType = itk::Image<PixelType, Dimension>;
```

(continues on next page)

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  using ReaderType = itk::ImageFileReader<ImageType>;

  if (argc < 4)
  {
    std::cout << "Usage: " << argv[0] << " imageFile1 imageFile2 outputFile" << std::
↪endl;
    return EXIT_FAILURE;
  }
  ReaderType::Pointer fixedReader = ReaderType::New();
  fixedReader->SetFileName(argv[1]);
  fixedReader->Update();
  ImageType::Pointer fixedImage = fixedReader->GetOutput();

  ReaderType::Pointer movingReader = ReaderType::New();
  movingReader->SetFileName(argv[2]);
  movingReader->Update();
  ImageType::Pointer movingImage = movingReader->GetOutput();

  // We use floats internally
  using InternalImageType = itk::Image<float, Dimension>;

  // Normalize the images
  using NormalizeFilterType = itk::NormalizeImageFilter<ImageType, InternalImageType>;

  NormalizeFilterType::Pointer fixedNormalizer = NormalizeFilterType::New();
  NormalizeFilterType::Pointer movingNormalizer = NormalizeFilterType::New();

  fixedNormalizer->SetInput(fixedImage);
  movingNormalizer->SetInput(movingImage);

  // Smooth the normalized images
  using GaussianFilterType = itk::DiscreteGaussianImageFilter<InternalImageType,␣
↪InternalImageType>;

  GaussianFilterType::Pointer fixedSmoother = GaussianFilterType::New();
  GaussianFilterType::Pointer movingSmoother = GaussianFilterType::New();

  fixedSmoother->SetVariance(2.0);
  movingSmoother->SetVariance(2.0);

  fixedSmoother->SetInput(fixedNormalizer->GetOutput());
  movingSmoother->SetInput(movingNormalizer->GetOutput());

  using TransformType = itk::AffineTransform<double, Dimension>;
  using OptimizerType = itk::GradientDescentOptimizer;
  using InterpolatorType = itk::LinearInterpolateImageFunction<InternalImageType,␣
↪double>;
  using RegistrationType = itk::ImageRegistrationMethod<InternalImageType,␣
↪InternalImageType>;
  using MetricType = itk::MutualInformationImageToImageMetric<InternalImageType,␣
↪InternalImageType>;

  TransformType::Pointer    transform = TransformType::New();
```

(continues on next page)

(continued from previous page)

```
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();

  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);

  MetricType::Pointer metric = MetricType::New();
  registration->SetMetric(metric);

  //  The metric requires a number of parameters to be selected, including
  //  the standard deviation of the Gaussian kernel for the fixed image
  //  density estimate, the standard deviation of the kernel for the moving
  //  image density and the number of samples use to compute the densities
  //  and entropy values. Details on the concepts behind the computation of
  //  the metric can be found in Section
  //  \ref{sec:MutualInformationMetric}.  Experience has
  //  shown that a kernel standard deviation of $0.4$ works well for images
  //  which have been normalized to a mean of zero and unit variance.  We
  //  will follow this empirical rule in this example.

  metric->SetFixedImageStandardDeviation(5.0);
  metric->SetMovingImageStandardDeviation(5.0);

  registration->SetFixedImage(fixedSmoother->GetOutput());
  registration->SetMovingImage(movingSmoother->GetOutput());

  fixedNormalizer->Update();
  ImageType::RegionType fixedImageRegion = fixedNormalizer->GetOutput()->
↪GetBufferedRegion();
  registration->SetFixedImageRegion(fixedImageRegion);

  using ParametersType = RegistrationType::ParametersType;
  ParametersType initialParameters(transform->GetNumberOfParameters());

  // rotation matrix (identity)
  initialParameters[0] = 1.0; // R(0,0)
  initialParameters[1] = 0.0; // R(0,1)
  initialParameters[2] = 0.0; // R(1,0)
  initialParameters[3] = 1.0; // R(1,1)

  // translation vector
  initialParameters[4] = 0.0;
  initialParameters[5] = 0.0;

  registration->SetInitialTransformParameters(initialParameters);

  //  Software Guide : BeginLatex
  //
  //  We should now define the number of spatial samples to be considered in
  //  the metric computation. Note that we were forced to postpone this setting
  //  until we had done the preprocessing of the images because the number of
  //  samples is usually defined as a fraction of the total number of pixels in
  //  the fixed image.
  //
  //  The number of spatial samples can usually be as low as $1\%$ of the total
```

(continues on next page)

(continued from previous page)

```
  //  number of pixels in the fixed image. Increasing the number of samples
  //  improves the smoothness of the metric from one iteration to another and
  //  therefore helps when this metric is used in conjunction with optimizers
  //  that rely of the continuity of the metric values. The trade-off, of
  //  course, is that a larger number of samples result in longer computation
  //  times per every evaluation of the metric.
  //
  //  It has been demonstrated empirically that the number of samples is not a
  //  critical parameter for the registration process. When you start fine
  //  tuning your own registration process, you should start using high values
  //  of number of samples, for example in the range of $20\%$ to $50\%$ of the
  //  number of pixels in the fixed image. Once you have succeeded to register
  //  your images you can then reduce the number of samples progressively until
  //  you find a good compromise on the time it takes to compute one evaluation
  //  of the Metric. Note that it is not useful to have very fast evaluations
  //  of the Metric if the noise in their values results in more iterations
  //  being required by the optimizer to converge. You must then study the
  //  behavior of the metric values as the iterations progress.

  const unsigned int numberOfPixels = fixedImageRegion.GetNumberOfPixels();

  const auto numberOfSamples = static_cast<unsigned int>(numberOfPixels * 0.01);

  metric->SetNumberOfSpatialSamples(numberOfSamples);

  // For consistent results when regression testing.
  metric->ReinitializeSeed(121212);

  // Note that large values of the learning rate will make the optimizer
  // unstable. Small values, on the other hand, may result in the optimizer
  // needing too many iterations in order to walk to the extrema of the cost
  // function. The easy way of fine tuning this parameter is to start with
  // small values, probably in the range of $\{5.0,10.0\}$. Once the other
  // registration parameters have been tuned for producing convergence, you
  // may want to revisit the learning rate and start increasing its value until
  // you observe that the optimization becomes unstable.  The ideal value for
  // this parameter is the one that results in a minimum number of iterations
  // while still keeping a stable path on the parametric space of the
  // optimization. Keep in mind that this parameter is a multiplicative factor
  // applied on the gradient of the Metric. Therefore, its effect on the
  // optimizer step length is proportional to the Metric values themselves.
  // Metrics with large values will require you to use smaller values for the
  // learning rate in order to maintain a similar optimizer behavior.
  optimizer->SetLearningRate(1.0);

  // Note that the only stop condition for the v3 GradientDescentOptimizer class
  // is that the maximum number of iterations is reached.
  // For the option to exit early on convergence use GradientDescentOptimizerv4
  // with an accompanying v4 metric class.
  optimizer->SetNumberOfIterations(200);
  optimizer->MaximizeOn(); // We want to maximize mutual information (the default of␣
↪the optimizer is to minimize)

  auto scales = optimizer->GetScales();

  // Let optimizer take
  // large steps along translation parameters,
```

(continues on next page)

(continued from previous page)

```
  // moderate steps along rotational parameters,
  // and small steps along scale parameters
  scales.SetSize(6);
  scales.SetElement(0, 100);
  scales.SetElement(1, 0.5);
  scales.SetElement(2, 0.5);
  scales.SetElement(3, 100);
  scales.SetElement(4, 0.0001);
  scales.SetElement(5, 0.0001);

  optimizer->SetScales(scales);

  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition: " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cout << "ExceptionObject caught !" << std::endl;
    std::cout << err << std::endl;
    return EXIT_FAILURE;
  }

  ParametersType finalParameters = registration->GetLastTransformParameters();

  std::cout << "Final Parameters: " << finalParameters << std::endl;

  unsigned int numberOfIterations = optimizer->GetCurrentIteration();

  double bestValue = optimizer->GetValue();

  // Print out results
  std::cout << std::endl;
  std::cout << "Result = " << std::endl;
  std::cout << " Iterations    = " << numberOfIterations << std::endl;
  std::cout << " Metric value  = " << bestValue << std::endl;
  std::cout << " Numb. Samples = " << numberOfSamples << std::endl;

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;

  TransformType::Pointer finalTransform = TransformType::New();

  finalTransform->SetParameters(finalParameters);
  finalTransform->SetFixedParameters(transform->GetFixedParameters());

  ResampleFilterType::Pointer resample = ResampleFilterType::New();

  resample->SetTransform(finalTransform);
  resample->SetInput(movingImage);

  resample->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resample->SetOutputOrigin(fixedImage->GetOrigin());
  resample->SetOutputSpacing(fixedImage->GetSpacing());
  resample->SetOutputDirection(fixedImage->GetDirection());
```

(continues on next page)

(continued from previous page)

```
  resample->SetDefaultPixelValue(100);

  using WriterType = itk::ImageFileWriter<ImageType>;

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[3]);
  writer->SetInput(resample->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class AffineTransform** : **public** itk::MatrixOffsetTransformBase<*TParametersValueType*, *NDimensions*, *NDimensions*>

Affine transformation of a vector space (e.g. space coordinates)

This class allows the definition and manipulation of affine transformations of an n-dimensional affine space (and its associated vector space) onto itself. One common use is to define and manipulate Euclidean coordinate transformations in two and three dimensions, but other uses are possible as well.

An affine transformation is defined mathematically as a linear transformation plus a constant offset. If A is a constant n x n matrix and b is a constant n-vector, then y = Ax+b defines an affine transformation from the n-vector x to the n-vector y.

The difference between two points is a vector and transforms linearly, using the matrix only. That is, (y1-y2) = A*(x1-x2).

The AffineTransform class determines whether to transform an object as a point or a vector by examining its type. An object of type Point transforms as a point; an object of type Vector transforms as a vector.

One common use of affine transformations is to define coordinate conversions in two- and three-dimensional space. In this application, x is a two- or three-dimensional vector containing the "source" coordinates of a point, y is a vector containing the "target" coordinates, the matrix A defines the scaling and rotation of the coordinate systems from the source to the target, and b defines the translation of the origin from the source to the target. More generally, A can also define anisotropic scaling and shearing transformations. Any good textbook on computer graphics will discuss coordinate transformations in more detail. Several of the methods in this class are designed for this purpose and use the language appropriate to coordinate conversions.

Any two affine transformations may be composed and the result is another affine transformation. However, the order is important. Given two affine transformations T1 and T2, we will say that "precomposing T1 with T2" yields the transformation which applies T1 to the source, and then applies T2 to that result to obtain the target. Conversely, we will say that "postcomposing T1 with T2" yields the transformation which applies T2 to the source, and then applies T1 to that result to obtain the target. (Whether T1 or T2 comes first lexicographically depends on whether you choose to write mappings from right-to-left or vice versa; we avoid the whole problem by referring to the order of application rather than the textual order.)

There are two template parameters for this class:

TParametersValueType The type to be used for scalar numeric values. Either float or double.

NDimensions The number of dimensions of the vector space.

This class provides several methods for setting the matrix and vector defining the transform. To support the registration framework, the transform parameters can also be set as an Array<double> of size (NDimension + 1) * NDimension using method SetParameters(). The first (NDimension x NDimension) parameters defines the

matrix in row-major order (where the column index varies the fastest). The last NDimension parameters defines the translation in each dimensions.

This class also supports the specification of a center of rotation (center) and a translation that is applied with respect to that centered rotation. By default the center of rotation is set to the origin.

Subclassed by itk::AzimuthElevationToCartesianTransform< TParametersValueType, NDimensions >, itk::CenteredAffineTransform< TParametersValueType, NDimensions >, itk::ScalableAffineTransform< TParametersValueType, NDimensions >

See itk::AffineTransform for additional documentation.

template<typename **TFixedImage**, typename **TMovingImage**>
**class MutualInformationImageToImageMetric** : **public** itk::ImageToImageMetric<*TFixedImage*, *TMovingImage*>
Computes the mutual information between two images to be registered.

MutualInformationImageToImageMetric computes the mutual information between a fixed and moving image to be registered.

This class is templated over the FixedImage type and the MovingImage type.

The fixed and moving images are set via methods SetFixedImage() and SetMovingImage(). This metric makes use of user specified Transform and Interpolator. The Transform is used to map points from the fixed image to the moving image domain. The Interpolator is used to evaluate the image intensity at user specified geometric points in the moving image. The Transform and Interpolator are set via methods SetTransform() and SetInterpolator().

The method GetValue() computes of the mutual information while method GetValueAndDerivative() computes both the mutual information and its derivatives with respect to the transform parameters.

**Warning** This metric assumes that the moving image has already been connected to the interpolator outside of this class.

The calculations are based on the method of Viola and Wells where the probability density distributions are estimated using Parzen windows.

By default a Gaussian kernel is used in the density estimation. Other option include Cauchy and spline-based. A user can specify the kernel passing in a pointer a KernelFunctionBase using the SetKernelFunction() method.

Mutual information is estimated using two sample sets: one to calculate the singular and joint pdf's and one to calculate the entropy integral. By default 50 samples points are used in each set. Other values can be set via the SetNumberOfSpatialSamples() method.

Quality of the density estimate depends on the choice of the kernel's standard deviation. Optimal choice will depend on the images. It is can be shown that around the optimal variance, the mutual information estimate is relatively insensitive to small changes of the standard deviation. In our experiments, we have found that a standard deviation of 0.4 works well for images normalized to have a mean of zero and standard deviation of 1.0. The variance can be set via methods SetFixedImageStandardDeviation() and SetMovingImageStandardDeviation().

Implementaton of this class is based on: Viola, P. and Wells III, W. (1997). "Alignment by Maximization of Mutual Information" International Journal of Computer Vision, 24(2):137-154

**See** KernelFunctionBase

**See** GaussianKernelFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mutual Information
- Mutual Information Affine

See itk::MutualInformationImageToImageMetric for additional documentation.

**class GradientDescentOptimizer** : **public** itk::SingleValuedNonLinearOptimizer

Implement a gradient descent optimizer.

GradientDescentOptimizer implements a simple gradient descent optimizer. At each iteration the current position is updated according to

$$p_{n+1} = p_n + \text{learningRate}\,\frac{\partial f(p_n)}{\partial p_n}$$

The learning rate is a fixed scalar defined via SetLearningRate(). The optimizer steps through a user defined number of iterations; no convergence checking is done.

Additionally, user can scale each component, $\partial f/\partial p$, by setting a scaling vector using method SetScale().

**See** RegularStepGradientDescentOptimizer

Subclassed by itk::QuaternionRigidTransformGradientDescentOptimizer

See itk::GradientDescentOptimizer for additional documentation.

### Scale an Image

### Synopsis

Scale an image.

### Results

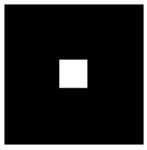


Fig. 77: Input Image

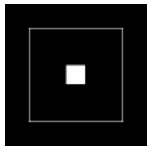


Fig. 78: Output Image

### Code

### C++

```
#include "itkImage.h"
#include "itkScaleTransform.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkResampleImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer inputWriter = WriterType::New();
  inputWriter->SetFileName("input.png");
  inputWriter->SetInput(image);
  inputWriter->Update();

  // using TransformType = itk::ScaleTransform<float, 2>; // If you want to use float␣
↪here, you must use:
  // using ResampleImageFilterType = itk::ResampleImageFilter<ImageType, ImageType,␣
↪float>; later.
  using TransformType = itk::ScaleTransform<double, 2>;
  TransformType::Pointer    scaleTransform = TransformType::New();
  itk::FixedArray<float, 2> scale;
  scale[0] = 1.5; // newWidth/oldWidth
  scale[1] = 1.5;
  scaleTransform->SetScale(scale);

  itk::Point<float, 2> center;
  center[0] = image->GetLargestPossibleRegion().GetSize()[0] / 2;
  center[1] = image->GetLargestPossibleRegion().GetSize()[1] / 2;

  scaleTransform->SetCenter(center);

  using ResampleImageFilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  ResampleImageFilterType::Pointer resampleFilter = ResampleImageFilterType::New();
  resampleFilter->SetTransform(scaleTransform);
  resampleFilter->SetInput(image);
  resampleFilter->SetSize(image->GetLargestPossibleRegion().GetSize());
  resampleFilter->Update();

  WriterType::Pointer outputWriter = WriterType::New();
  outputWriter->SetFileName("output.png");
  outputWriter->SetInput(resampleFilter->GetOutput());
  outputWriter->Update();

  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}

void
CreateImage(ImageType::Pointer image)
{
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(101);

  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a white square
  for (unsigned int r = 40; r < 60; r++)
  {
    for (unsigned int c = 40; c < 60; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Draw a white border
  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] == 0 ||
        imageIterator.GetIndex()[0] == static_cast<int>(image->
↪GetLargestPossibleRegion().GetSize()[0]) - 1 ||
        imageIterator.GetIndex()[1] == 0 ||
        imageIterator.GetIndex()[1] == static_cast<int>(image->
↪GetLargestPossibleRegion().GetSize()[1]) - 1)
    {
      imageIterator.Set(255);
    }
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TParametersValueType** = float, unsigned int **NDimensions** = 3>
**class ScaleTransform** : **public** itk::MatrixOffsetTransformBase<*TParametersValueType*, *NDimensions*, *NDimensions*>
Scale transformation of a vector space (e.g. space coordinates)

The same functionality could be obtained by using the Affine transform, but with a large difference in performance since the affine transform will use a matrix multiplication using a diagonal matrix.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Scale An Image

Subclassed by itk::ScaleLogarithmicTransform< TParametersValueType, NDimensions >

See itk::ScaleTransform for additional documentation.

### Translate a Vector Image

### Synopsis

Translate a vector image.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkTranslationTransform.h"
#include "itkImageFileReader.h"
#include "itkResampleImageFilter.h"
#include "itkCovariantVector.h"
#include "itkNumericTraits.h"

int
main(int, char *[])
{
  using VectorType = itk::CovariantVector<double, 3>;
  using VectorImageType = itk::Image<VectorType, 2>;

  VectorImageType::Pointer image = VectorImageType::New();
  itk::Index<2>            start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(10);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
```

(continues on next page)

(continued from previous page)

```
  image->Allocate();
  image->FillBuffer(itk::NumericTraits<VectorType>::ZeroValue());

  itk::TranslationTransform<double, 2>::Pointer          transform = itk::
↪TranslationTransform<double, 2>::New();
  itk::TranslationTransform<double, 2>::OutputVectorType translation;
  translation[0] = 10;
  translation[1] = 20;
  transform->Translate(translation);

  using ResampleFilterType = itk::ResampleImageFilter<VectorImageType,␣
↪VectorImageType>;
  ResampleFilterType::Pointer vectorResampleFilter = ResampleFilterType::New();
  vectorResampleFilter->SetInput(image);
  vectorResampleFilter->SetSize(image->GetLargestPossibleRegion().GetSize());
  vectorResampleFilter->SetTransform(transform);
  vectorResampleFilter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class TranslationTransform** : **public** itk::*Transform<TParametersValueType, NDimensions, NDimensions>*
Translation transformation of a vector space (e.g. space coordinates)

The same functionality could be obtained by using the Affine transform, but with a large difference in performance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Vector Image
- Global Registration Of Two Images
- Mutual Information

See itk::TranslationTransform for additional documentation.

### Translate Image

### Synopsis

Translate one itk::Image

### Results

Fig. 79: Input image

Fig. 80: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkTranslationTransform.h"
#include "itkResampleImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
```

(continues on next page)

(continued from previous page)

```
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <translation X> <translation Y>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  reader->UpdateOutputInformation();

  using TransformType = itk::TranslationTransform<double, Dimension>;

  TransformType::OutputVectorType vector;
  vector[0] = std::stod(argv[3]);
  vector[1] = std::stod(argv[4]);

  TransformType::Pointer translation = TransformType::New();
  translation->Translate(vector);

  using ResampleImageFilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  ResampleImageFilterType::Pointer resampleFilter = ResampleImageFilterType::New();
  resampleFilter->SetTransform(translation.GetPointer());
  resampleFilter->SetInput(reader->GetOutput());
  resampleFilter->SetSize(reader->GetOutput()->GetLargestPossibleRegion().GetSize());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(resampleFilter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class TranslationTransform** : **public** itk::*Transform*<*TParametersValueType*, *NDimensions*, *NDimensions*>
Translation transformation of a vector space (e.g. space coordinates)

The same functionality could be obtained by using the Affine transform, but with a large difference in performance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Vector Image
- Global Registration Of Two Images
- Mutual Information

See itk::TranslationTransform for additional documentation.

## 3.3 External

## 3.4 Filtering

### 3.4.1 AnisotropicSmoothing

#### Compute Curvature Anisotropic Diffusion

#### Synopsis

Perform anisotropic diffusion on an image.

#### Results

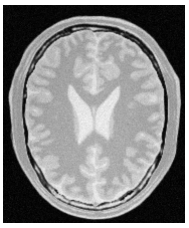


Fig. 81: Input image

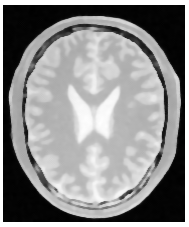


Fig. 82: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Curvature Anisotropic Diffusion.
↪")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_iterations", type=int)
parser.add_argument("time_step", type=float)
parser.add_argument("conductance", type=float)
args = parser.parse_args()

InputPixelType = itk.F
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.CurvatureAnisotropicDiffusionImageFilter[
    InputImageType, InputImageType
]
curvatureAnisotropicDiffusionFilter = FilterType.New()

curvatureAnisotropicDiffusionFilter.SetInput(reader.GetOutput())
curvatureAnisotropicDiffusionFilter.SetNumberOfIterations(args.number_of_iterations)
curvatureAnisotropicDiffusionFilter.SetTimeStep(args.time_step)
curvatureAnisotropicDiffusionFilter.SetConductanceParameter(args.conductance)

RescaleFilterType = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType]
rescaler = RescaleFilterType.New()
rescaler.SetInput(curvatureAnisotropicDiffusionFilter.GetOutput())

outputPixelTypeMinimum = itk.NumericTraits[OutputPixelType].min()
outputPixelTypeMaximum = itk.NumericTraits[OutputPixelType].max()
```

(continues on next page)

(continued from previous page)

```
rescaler.SetOutputMinimum(outputPixelTypeMinimum)
rescaler.SetOutputMaximum(outputPixelTypeMaximum)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCurvatureAnisotropicDiffusionImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " <numberOfIterations> <timeStep> <conductance>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using InputPixelType = float;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  const int            numberOfIterations = std::stoi(argv[3]);
  const InputPixelType timeStep = std::stod(argv[4]);
  const InputPixelType conductance = std::stod(argv[5]);

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::CurvatureAnisotropicDiffusionImageFilter<InputImageType,␣
↪InputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfIterations(numberOfIterations);
```

(continues on next page)

(continued from previous page)

```
  filter->SetTimeStep(timeStep);
  filter->SetConductanceParameter(conductance);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(rescaler->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CurvatureAnisotropicDiffusionImageFilter** : **public** itk::AnisotropicDiffusionImageFilter<*TInputImage*, *TO*

This filter performs anisotropic diffusion on a scalar itk::Image using the modified curvature diffusion equation (MCDE).

For detailed information on anisotropic diffusion and the MCDE see itkAnisotropicDiffusionFunction and itkCurvatureNDAnisotropicDiffusionFunction.

The default time step for this filter is set to the maximum theoretically stable value: 0.5 / 2^N, where N is the dimensionality of the image. For a 2D image, this means valid time steps are below 0.1250. For a 3D image, valid time steps are below 0.0625.

**Inputs and Outputs** The input and output to this filter must be a scalar itk::Image with numerical pixel types (float or double). A user defined type which correctly defines arithmetic operations with floating point accuracy should also give correct results.

**Parameters** Please first read all the documentation found in AnisotropicDiffusionImageFilter and AnisotropicDiffusionFunction. Also see CurvatureNDAnisotropicDiffusionFunction.

**See** AnisotropicDiffusionImageFilter

**See** AnisotropicDiffusionFunction

**See** CurvatureNDAnisotropicDiffusionFunction

See itk::CurvatureAnisotropicDiffusionImageFilter for additional documentation.

### Compute Curvature Flow

### Synopsis

Denoise an image using curvature driven flow.

### Results

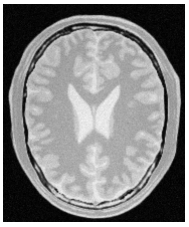


Fig. 83: Input image

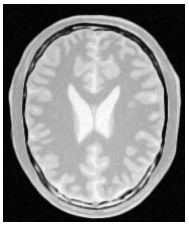


Fig. 84: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Curvature Flow.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_iterations", type=int)
parser.add_argument("time_step", type=float)
args = parser.parse_args()

InputPixelType = itk.F
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]
```

(continues on next page)

(continued from previous page)

```
ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.CurvatureFlowImageFilter[InputImageType, InputImageType]
curvatureFlowFilter = FilterType.New()

curvatureFlowFilter.SetInput(reader.GetOutput())
curvatureFlowFilter.SetNumberOfIterations(args.number_of_iterations)
curvatureFlowFilter.SetTimeStep(args.time_step)

RescaleFilterType = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType]
rescaler = RescaleFilterType.New()
rescaler.SetInput(curvatureFlowFilter.GetOutput())

outputPixelTypeMinimum = itk.NumericTraits[OutputPixelType].min()
outputPixelTypeMaximum = itk.NumericTraits[OutputPixelType].max()

rescaler.SetOutputMinimum(outputPixelTypeMinimum)
rescaler.SetOutputMaximum(outputPixelTypeMaximum)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCurvatureFlowImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " <numberOfIterations> <timeStep>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using InputPixelType = float;
```

(continues on next page)

(continued from previous page)

```
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  const int            numberOfIterations = std::stoi(argv[3]);
  const InputPixelType timeStep = std::stod(argv[4]);

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::CurvatureFlowImageFilter<InputImageType, InputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfIterations(numberOfIterations);
  filter->SetTimeStep(timeStep);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(rescaler->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CurvatureFlowImageFilter** : **public** itk::DenseFiniteDifferenceImageFilter<*TInputImage*, *TOutputImage*>
Denoise an image using curvature driven flow.

CurvatureFlowImageFilter implements a curvature driven image denoising algorithm. Iso-brightness contours in the grayscale input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = \kappa |\nabla I|$$

where $\kappa$ is the curvature.

The advantage of this approach is that sharp boundaries are preserved with smoothing occurring only within a region. However, it should be noted that continuous application of this scheme will result in the eventual removal of all information as each contour shrinks to zero and disappear.

Note that unlike level set segmentation algorithms, the image to be denoised is already the level set and can be set directly as the input using the SetInput() method.

This filter has two parameters: the number of update iterations to be performed and the timestep between each update.

The timestep should be "small enough" to ensure numerical stability. Stability is guarantee when the timestep meets the CFL (Courant-Friedrichs-Levy) condition. Broadly speaking, this condition ensures that each contour does not move more than one grid position at each timestep. In the literature, the timestep is typically user specified and have to manually tuned to the application.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a CurvatureFlowFunction object. A zero flux Neumann boundary condition when computing derivatives near the data boundary.

This filter may be streamed. To support streaming this filter produces a padded output which takes into account edge effects. The size of the padding is m_NumberOfIterations on each edge. Users of this filter should only make use of the center valid central region.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a floating point type. This filter works for any dimensional images.

Input/Output Restrictions: TInputImage and TOutputImage must have the same dimension. TOutputImage's pixel type must be a real number type.

**See** DenseFiniteDifferenceImageFilter

**See** CurvatureFlowFunction

**See** MinMaxCurvatureFlowImageFilter

**See** BinaryMinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image Using Curvature Flow
- Smooth RGB Image Using Curvature Flow

Subclassed by itk::MinMaxCurvatureFlowImageFilter< TInputImage, TOutputImage >

See itk::CurvatureFlowImageFilter for additional documentation.

### Compute Gradient Anisotropic Diffusion

### Synopsis

Perform anisotropic diffusion on an image.

### Results

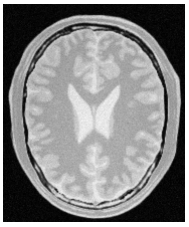


Fig. 85: Input image

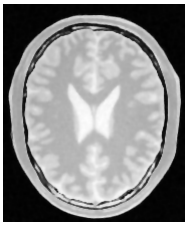


Fig. 86: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Gradient Anisotropic Diffusion.
↪")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_iterations", type=int)
parser.add_argument("time_step", type=float)
parser.add_argument("conductance", type=float)
args = parser.parse_args()

InputPixelType = itk.F
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
```

(continues on next page)

(continued from previous page)

```
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.GradientAnisotropicDiffusionImageFilter[InputImageType,␣
↪InputImageType]
gradientAnisotropicDiffusionFilter = FilterType.New()

gradientAnisotropicDiffusionFilter.SetInput(reader.GetOutput())
gradientAnisotropicDiffusionFilter.SetNumberOfIterations(args.number_of_iterations)
gradientAnisotropicDiffusionFilter.SetTimeStep(args.time_step)
gradientAnisotropicDiffusionFilter.SetConductanceParameter(args.conductance)

RescaleFilterType = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType]
rescaler = RescaleFilterType.New()
rescaler.SetInput(gradientAnisotropicDiffusionFilter.GetOutput())

outputPixelTypeMinimum = itk.NumericTraits[OutputPixelType].min()
outputPixelTypeMaximum = itk.NumericTraits[OutputPixelType].max()

rescaler.SetOutputMinimum(outputPixelTypeMinimum)
rescaler.SetOutputMaximum(outputPixelTypeMaximum)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGradientAnisotropicDiffusionImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " <numberOfIterations> <timeStep> <conductance>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Dimension = 2;

  using InputPixelType = float;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  const int            numberOfIterations = std::stoi(argv[3]);
  const InputPixelType timeStep = std::stod(argv[4]);
  const InputPixelType conductance = std::stod(argv[5]);

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::GradientAnisotropicDiffusionImageFilter<InputImageType,␣
↪InputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfIterations(numberOfIterations);
  filter->SetTimeStep(timeStep);
  filter->SetConductanceParameter(conductance);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(rescaler->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class GradientAnisotropicDiffusionImageFilter** : **public** itk::AnisotropicDiffusionImageFilter<*TInputImage*, *TOu*
This filter performs anisotropic diffusion on a scalar itk::Image using the classic Perona-Malik, gradient magnitude based equation.

For detailed information on anisotropic diffusion, see itkAnisotropicDiffusionFunction and itkGradientNDAnisotropicDiffusionFunction.

**Inputs and Outputs** The input to this filter should be a scalar itk::Image of any dimensionality. The output image will be a diffused copy of the input.

**Parameters** Please see the description of parameters given in itkAnisotropicDiffusionImageFilter.

**See** AnisotropicDiffusionImageFilter

**See** AnisotropicDiffusionFunction

**See** GradientAnisotropicDiffusionFunction

See itk::GradientAnisotropicDiffusionImageFilter for additional documentation.

### Perona Malik Anisotropic Diffusion on Grayscale Image

### Synopsis

Perona Malik Anisotropic Diffusion for scalar valued images.

### Results

Fig. 87: Before anisotropic diffusion (left) and after anisotropic diffusion (right).

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Compute Perona Malik Anisotropic Diffusion."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_iterations", type=int)
parser.add_argument("conductance", type=float)
args = parser.parse_args()


Dimension = 2
InputPixelType = itk.UC
InputImageType = itk.Image[InputPixelType, Dimension]
OutputPixelType = itk.F
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

CastFilterType = itk.CastImageFilter[InputImageType, OutputImageType]
castfilter = CastFilterType.New()
castfilter.SetInput(reader)

FilterType = itk.GradientAnisotropicDiffusionImageFilter[
    OutputImageType, OutputImageType
]
gradientfilter = FilterType.New()
gradientfilter.SetInput(castfilter.GetOutput())
gradientfilter.SetNumberOfIterations(args.number_of_iterations)
gradientfilter.SetTimeStep(0.125)
gradientfilter.SetConductanceParameter(args.conductance)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(gradientfilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGradientAnisotropicDiffusionImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << " <OutputFileName>";
    std::cerr << " <NumberOfIterations> ";
    std::cerr << " <Conductance>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using OutputPixelType = float;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;
  using FilterType = itk::GradientAnisotropicDiffusionImageFilter<InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfIterations(std::stoi(argv[3]));
  filter->SetTimeStep(0.125);
  filter->SetConductanceParameter(std::stod(argv[4]));

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class GradientAnisotropicDiffusionImageFilter** : **public** itk::AnisotropicDiffusionImageFilter<*TInputImage*, *TOu*
This filter performs anisotropic diffusion on a scalar itk::Image using the classic Perona-Malik, gradient magnitude based equation.

For detailed information on anisotropic diffusion, see itkAnisotropicDiffusionFunction and itkGradientNDAnisotropicDiffusionFunction.

**Inputs and Outputs** The input to this filter should be a scalar itk::Image of any dimensionality. The output image will be a diffused copy of the input.

**Parameters** Please see the description of parameters given in itkAnisotropicDiffusionImageFilter.

**See** AnisotropicDiffusionImageFilter

**See** AnisotropicDiffusionFunction

**See** GradientAnisotropicDiffusionFunction

See itk::GradientAnisotropicDiffusionImageFilter for additional documentation.

### Smooth Image While Preserving Edges

### Synopsis

Smooth an image while preserving edges.

### Results

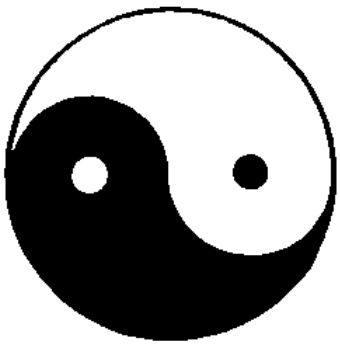


Fig. 88: Input image.

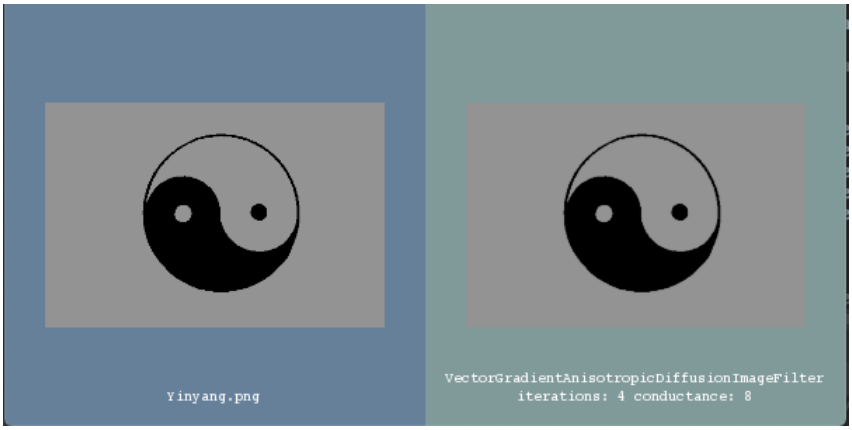



Fig. 89: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkCastImageFilter.h"
#include "itkImageFileReader.h"
#include "itkVectorGradientAnisotropicDiffusionImageFilter.h"
#include "itkVectorToRGBImageAdaptor.h"
#include "itkRGBToVectorImageAdaptor.h"
#include "itkCastImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " InputFileName";
    std::cerr << " [NumberOfIterations] ";
    std::cerr << " [Conductance]" << std::endl;
    return EXIT_FAILURE;
  }

  // 0) Parse arguments
  std::string inputFileName = argv[1];

  using FloatImageType = itk::Image<itk::Vector<float, 3>, 2>;
  using RGBImageType = itk::Image<itk::RGBPixel<float>, 2>;
```

(continues on next page)

(continued from previous page)

```
  // 1) Read the RGB image
  using ReaderType = itk::ImageFileReader<RGBImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // 2) Cast to Vector image for processing
  using AdaptorInputType = itk::RGBToVectorImageAdaptor<RGBImageType>;
  AdaptorInputType::Pointer adaptInput = AdaptorInputType::New();
  adaptInput->SetImage(reader->GetOutput());
  using CastInputType = itk::CastImageFilter<AdaptorInputType, FloatImageType>;
  CastInputType::Pointer castInput = CastInputType::New();
  castInput->SetInput(adaptInput);

  // 3) Smooth the image
  using VectorGradientAnisotropicDiffusionImageFilterType =
    itk::VectorGradientAnisotropicDiffusionImageFilter<FloatImageType, FloatImageType>
↪;
  VectorGradientAnisotropicDiffusionImageFilterType::Pointer filter =
    VectorGradientAnisotropicDiffusionImageFilterType::New();
  filter->SetInput(castInput->GetOutput());
  filter->SetTimeStep(0.125);
  if (argc > 2)
  {
    filter->SetNumberOfIterations(atoi(argv[2]));
  }
  if (argc > 3)
  {
    filter->SetConductanceParameter(atof(argv[3]));
  }

  // 4) Cast the Vector image to an RGB image for display
  using AdaptorOutputType = itk::VectorToRGBImageAdaptor<FloatImageType>;
  AdaptorOutputType::Pointer adaptOutput = AdaptorOutputType::New();
  adaptOutput->SetImage(filter->GetOutput());
  using CastOutputType = itk::CastImageFilter<AdaptorOutputType, RGBImageType>;
  CastOutputType::Pointer castOutput = CastOutputType::New();
  castOutput->SetInput(adaptOutput);

  // 5) Display the input and smoothed images
#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFileName));

  std::stringstream desc;
  desc << "VectorGradientAnisotropicDiffusionImageFilter\niterations: " << filter->
↪GetNumberOfIterations()
       << " conductance: " << filter->GetConductanceParameter();
  viewer.AddRGBImage(castOutput->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class VectorGradientAnisotropicDiffusionImageFilter** : **public** itk::AnisotropicDiffusionImageFilter<*TInputIm*

This filter performs anisotropic diffusion on a vector itk::Image using the anisotropic diffusion function implemented implemented in itkVectorGradientNDAnisotropicDiffusionFunction. For detailed information on anisotropic diffusion see itkAnisotropicDiffusionFunction, itkVectorGradientNDAnisotropicDiffusionFunction, and itkGradientAnisotropicDiffusionFunction.

The maximum allowable time step for this filter is 1/2^N, where N is the dimensionality of the image. For 2D images any value below 0.250 is stable, and for 3D images, any value below 0.125 is stable.

**Inputs and Outputs** The input to this filter must be an itk::Image with pixel type which is either an itk::Vector, or a subclass of an itk::Vector. Additionally, the component type of the vector should be a numerical type (float or double, or a user defined type which correctly defines arithmetic operations with floating point accuracy). The output image type also has these requirements.

**Parameters** Please read all the documentation found in AnisotropicDiffusionImageFilter and AnisotropicDiffusionFunction. Also see VectorGradientNDAnisotropicDiffusionFunction.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image While Preserving Edges

See itk::VectorGradientAnisotropicDiffusionImageFilter for additional documentation.

### Smooth Image While Preserving Edges (Curvature)

### Synopsis

Smooth an image while preserving edges.

### Results

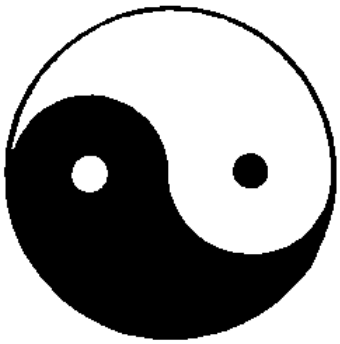


Fig. 90: Input image.

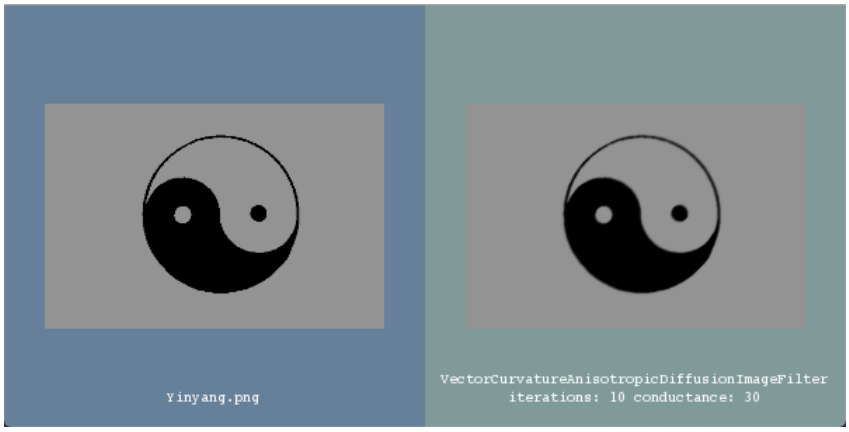



Fig. 91: Output In VTK Window

## Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkVectorCurvatureAnisotropicDiffusionImageFilter.h"
#include "itkVectorToRGBImageAdaptor.h"
#include "itkRGBToVectorImageAdaptor.h"
#include "itkCastImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " InputFileName";
    std::cerr << " [NumberOfIterations] ";
    std::cerr << " [Conductance]" << std::endl;
    return EXIT_FAILURE;
  }

  // 0) Parse arguments
  std::string inputFileName = argv[1];

  using FloatImageType = itk::Image<itk::Vector<float, 3>, 2>;
  using RGBImageType = itk::Image<itk::RGBPixel<float>, 2>;
```

(continues on next page)

(continued from previous page)

```
  // 1) Read the RGB image
  using ReaderType = itk::ImageFileReader<RGBImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // 2) Cast to Vector image for processing
  using AdaptorInputType = itk::RGBToVectorImageAdaptor<RGBImageType>;
  AdaptorInputType::Pointer adaptInput = AdaptorInputType::New();
  adaptInput->SetImage(reader->GetOutput());
  using CastInputType = itk::CastImageFilter<AdaptorInputType, FloatImageType>;
  CastInputType::Pointer castInput = CastInputType::New();
  castInput->SetInput(adaptInput);

  // 3) Smooth the image
  using VectorCurvatureAnisotropicDiffusionImageFilterType =
    itk::VectorCurvatureAnisotropicDiffusionImageFilter<FloatImageType,␣
↪FloatImageType>;
  VectorCurvatureAnisotropicDiffusionImageFilterType::Pointer filter =
    VectorCurvatureAnisotropicDiffusionImageFilterType::New();
  filter->SetInput(castInput->GetOutput());
  filter->SetTimeStep(0.125);
  if (argc > 2)
  {
    filter->SetNumberOfIterations(atoi(argv[2]));
  }
  if (argc > 3)
  {
    filter->SetConductanceParameter(atof(argv[3]));
  }

  // 4) Cast the Vector image to an RGB image for display
  using AdaptorOutputType = itk::VectorToRGBImageAdaptor<FloatImageType>;
  AdaptorOutputType::Pointer adaptOutput = AdaptorOutputType::New();
  adaptOutput->SetImage(filter->GetOutput());
  using CastOutputType = itk::CastImageFilter<AdaptorOutputType, RGBImageType>;
  CastOutputType::Pointer castOutput = CastOutputType::New();
  castOutput->SetInput(adaptOutput);

  // 5) Display the input and smoothed images
#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFileName));

  std::stringstream desc;
  desc << "VectorCurvatureAnisotropicDiffusionImageFilter\niterations: " << filter->
↪GetNumberOfIterations()
       << " conductance: " << filter->GetConductanceParameter();
  viewer.AddRGBImage(castOutput->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class VectorCurvatureAnisotropicDiffusionImageFilter** : **public** itk::AnisotropicDiffusionImageFilter<*TInputI*

This filter performs anisotropic diffusion on a vector itk::Image using the modified curvature diffusion equation (MCDE) implemented in itkVectorCurvatureNDAnisotropicDiffusionFunction. For detailed information on anisotropic diffusion and the MCDE see itkAnisotropicDiffusionFunction, itkVectorCurvatureNDAnisotropicDiffusionFunction, and itkCurvatureNDAnisotropicDiffusionFunction.

The default time step for this filter is set to the maximum theoretically stable value: 0.5 / 2^N, where N is the dimensionality of the image. For a 2D image, this means valid time steps are below 0.1250. For a 3D image, valid time steps are below 0.0625.

**Inputs and Outputs** The input to this filter must be an itk::Image with pixel type which is either an itk::Vector, or a subclass of an itk::Vector. Additionally, the component type of the vector should be a numerical type (float or double, or a user defined type which correctly defines arithmetic operations with floating point accuracy). The output image type also has these requirements.

**Parameters** Please first read all the documentation found in AnisotropicDiffusionImageFilter and AnisotropicDiffusionFunction. Also see VectorCurvatureNDAnisotropicDiffusionFunction.

**See** AnisotropicDiffusionImageFilter

**See** AnisotropicDiffusionFunction

**See** CurvatureNDAnisotropicDiffusionFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image While Preserving Edges (Curvature)

See itk::VectorCurvatureAnisotropicDiffusionImageFilter for additional documentation.

### 3.4.2 AntiAlias

#### Smooth Binary Image Before Surface Extraction

#### Synopsis

This example introduces the use of the *AntiAliasBinaryImageFilter*. This filter expects a binary mask as input. With level sets, it smooths the image by keeping the edge of the structure within a one pixel distance from the original location. It is usually desirable to run this filter before extracting an isocontour with surface extraction methods.

#### Results

#### Code

#### Python

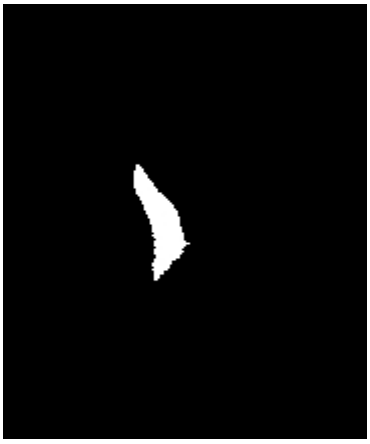


Fig. 92: Input image

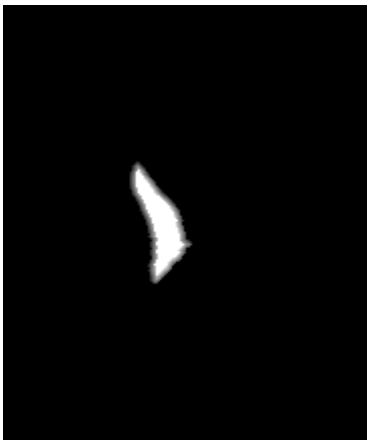


Fig. 93: Output image

```python
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Smooth Binary Image Before Surface Extraction."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("maximum_RMS_error", type=float)
parser.add_argument("number_of_iterations", type=int)
parser.add_argument("number_of_layers", type=int)
args = parser.parse_args()

PixelType = itk.F
Dimension = 2
ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

AntiAliasFilterType = itk.AntiAliasBinaryImageFilter[ImageType, ImageType]
antialiasfilter = AntiAliasFilterType.New()
antialiasfilter.SetInput(reader.GetOutput())
antialiasfilter.SetMaximumRMSError(args.maximum_RMS_error)
antialiasfilter.SetNumberOfIterations(args.number_of_iterations)
antialiasfilter.SetNumberOfLayers(args.number_of_layers)

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(antialiasfilter.GetOutput())

writer.Update()
```

### C++

```cpp
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkAntiAliasBinaryImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <inputImage> <outputImage>";
    std::cerr << " [maximumRMSError]";
    std::cerr << " [numberOfIterations]";
    std::cerr << " [numberOfLayers]" << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  double       maximumRMSError = 0.001;
  if (argc > 3)
  {
    maximumRMSError = std::stod(argv[3]);
  }
  int numberOfIterations = 50;
  if (argc > 4)
  {
    numberOfIterations = std::stoi(argv[4]);
  }
  int numberOfLayers = 2;
  if (argc > 5)
  {
    numberOfLayers = std::stoi(argv[5]);
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = float;

  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::AntiAliasBinaryImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetMaximumRMSError(maximumRMSError);
  filter->SetNumberOfIterations(numberOfIterations);
  filter->SetNumberOfLayers(numberOfLayers);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class AntiAliasBinaryImageFilter** : **public** itk::SparseFieldLevelSetImageFilter<*TInputImage*, *TOutputImage*>
A method for estimation of a surface from a binary volume.

This filter implements a surface-fitting method for estimation of a surface from a binary volume. This process can be used to reduce aliasing artifacts which result in visualization of binary partitioned surfaces.

The binary volume (filter input) is used as a set of constraints in an iterative relaxation process of an estimated ND surface. The surface is described implicitly as the zero level set of a volume $\phi$ and allowed to deform under curvature flow. A set of constraints is imposed on this movement as follows:

$$u_{i,j,k}^{n+1} = \begin{cases} \max(u_{i,j,k}^n + \Delta t H_{i,j,k}^n, 0) & B_{i,j,k} = 1 \\ \min(u_{i,j,k}^n + \Delta t H_{i,j,k}^n, 0) & B_{i,j,k} = -1 \end{cases}$$

where $u_{i,j,k}^n$ is the value of $\phi$ at discrete index $(i, j, k)$ and iteration $n$, $H$ is the gradient magnitude times mean curvature of $\phi$, and $B$ is the binary input volume, with 1 denoting an inside pixel and -1 denoting an outside pixel.

**NOTES** This implementation uses a sparse field level set solver instead of the narrow band implementation described in the reference below, which may introduce some differences in how fast and how accurately (in terms of RMS error) the solution converges.

**REFERENCES**

Whitaker, Ross. "Reducing Aliasing Artifacts In Iso-Surfaces of Binary

Volumes" IEEE Volume Visualization and Graphics Symposium, October 2000, pp.23-32.

**PARAMETERS** The MaximumRMSChange parameter is used to determine when the solution has converged. A lower value will result in a tighter-fitting solution, but will require more computations. Too low a value could put the solver into an infinite loop. Values should always be less than 1.0. A value of 0.07 is a good starting estimate.

The MaximumIterations parameter can be used to halt the solution after a specified number of iterations.

**INPUT** The input is an N-dimensional image of any type. It is assumed to be a binary image. The filter will use an isosurface value that is halfway between the min and max values in the image. A signed data type is *not* necessary for the input.

**OUTPUT** The filter will output a level set image of real, signed values. The zero crossings of this (N-dimensional) image represent the position of the isosurface value of interest. Values outside the zero level set are negative and values inside the zero level set are positive values.

**IMPORTANT!** The output image type you use to instantiate this filter should be a real valued scalar type. In other words: doubles or floats.

**USING THIS FILTER** The filter is relatively straightforward to use. Tests and examples exist to illustrate. The important thing is to understand the input and output types so you can properly interpret your results.

In the common case, the only parameter that will need to be set is the MaximumRMSChange parameter, which determines when the solver halts.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Binary Image Before Surface Extraction

See itk::AntiAliasBinaryImageFilter for additional documentation.

### 3.4.3 Binary Mathematical Morphology

#### Closing Binary Image

#### Synopsis

Closing a binary image.

#### Results

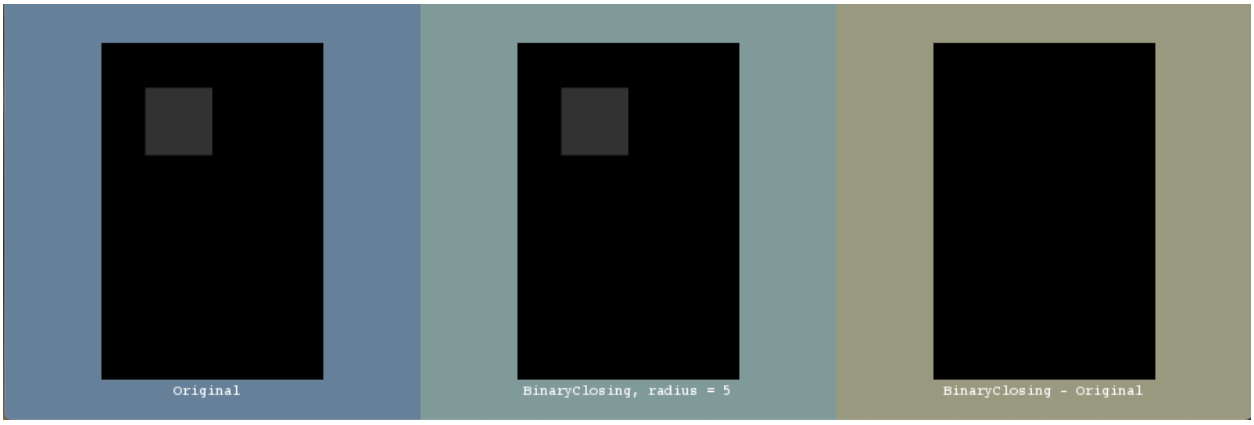



Fig. 94: Output In VTK Window

Output:

```
Radius: 5
```

#### Code

#### C++

```
#include "itkImage.h"
#include "itkBinaryMorphologicalClosingImageFilter.h"
#include "itkImageFileReader.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkSubtractImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
}
```

(continues on next page)

(continued from previous page)

```
static void
CreateImage(ImageType * const image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image;

  if (argc == 1)
  {
    image = ImageType::New();
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  unsigned int radius = 5;
  if (argc == 3)
  {
    std::stringstream ss(argv[2]);
    ss >> radius;
  }
  std::cout << "Radius: " << radius << std::endl;
  using StructuringElementType = itk::BinaryBallStructuringElement<ImageType::
↪PixelType, ImageType::ImageDimension>;
  StructuringElementType structuringElement;
  structuringElement.SetRadius(radius);
  structuringElement.CreateStructuringElement();

  using BinaryMorphologicalClosingImageFilterType =
    itk::BinaryMorphologicalClosingImageFilter<ImageType, ImageType,␣
↪StructuringElementType>;
  BinaryMorphologicalClosingImageFilterType::Pointer closingFilter =␣
↪BinaryMorphologicalClosingImageFilterType::New();
  closingFilter->SetInput(image);
  closingFilter->SetKernel(structuringElement);
  closingFilter->Update();

  using SubtractType = itk::SubtractImageFilter<ImageType>;
  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(closingFilter->GetOutput());
  diff->SetInput2(image);

#ifdef ENABLE_QUICKVIEW
  QuickView          viewer;
  std::stringstream desc;
  desc << "Original ";
  viewer.AddImage(image.GetPointer(), true, desc.str());

  std::stringstream desc2;
```

(continues on next page)

(continued from previous page)

```
  desc2 << "BinaryClosing, radius = " << radius;
  viewer.AddImage(closingFilter->GetOutput(), true, desc2.str());

  std::stringstream desc3;
  desc3 << "BinaryClosing - Original";
  viewer.AddImage(diff->GetOutput(), true, desc3.str());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}


void
CreateImage(ImageType * const image)
{
  // Create an image with 2 connected components
  itk::Index<2> corner = { { 0, 0 } };

  itk::Size<2> size;
  unsigned int NumRows = 200;
  unsigned int NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  itk::ImageRegion<2> region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      itk::Index<2> pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 50);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class BinaryMorphologicalClosingImageFilter** : **public** itk::KernelImageFilter<*TInputImage*, *TOutputImage*, *TKern*

binary morphological closing of an image.

This filter removes small (i.e., smaller than the structuring element) holes and tube like structures in the interior or at the boundaries of the image. The morphological closing of an image "f" is defined as: Closing(f) = Erosion(Dilation(f)).

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel.

This code was contributed in the Insight Journal paper: "Binary morphological closing and opening image filters" by Lehmann G. https://www.insight-journal.org/browse/publication/58

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** MorphologyImageFilter, GrayscaleDilateImageFilter, GrayscaleErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Closing Binary Image

See itk::BinaryMorphologicalClosingImageFilter for additional documentation.

### Dilate a Binary Image

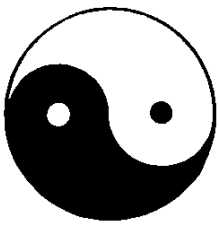
**See also:**

dilation; erosion

### Synopsis

Dilate regions by using a specified kernel, also known as a structuring element. In this example, the white regions are enlarged.

### Results

| | |
|---|---|
|  |  |
| Input binary image. | Dilated output image. |

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Dilate A Binary Image.")
```

(continues on next page)

(continued from previous page)

```
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

StructuringElementType = itk.FlatStructuringElement[Dimension]
structuringElement = StructuringElementType.Ball(args.radius)

DilateFilterType = itk.BinaryDilateImageFilter[
    ImageType, ImageType, StructuringElementType
]
dilateFilter = DilateFilterType.New()
dilateFilter.SetInput(reader.GetOutput())
dilateFilter.SetKernel(structuringElement)
dilateFilter.SetForegroundValue(255)

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(dilateFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkFlatStructuringElement.h"
#include "itkBinaryDilateImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <inputImage> <outputImage> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char *        inputImage = argv[1];
  const char *        outputImage = argv[2];
  const unsigned int radiusValue = std::stoi(argv[3]);
```

(continues on next page)

(continued from previous page)

```
  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);
  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using BinaryDilateImageFilterType = itk::BinaryDilateImageFilter<ImageType,␣
↪ImageType, StructuringElementType>;

  BinaryDilateImageFilterType::Pointer dilateFilter = BinaryDilateImageFilterType::
↪New();
  dilateFilter->SetInput(reader->GetOutput());
  dilateFilter->SetKernel(structuringElement);
  dilateFilter->SetForegroundValue(255); // Value to dilate

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(dilateFilter->GetOutput());
  writer->SetFileName(outputImage);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class BinaryDilateImageFilter** : **public** itk::BinaryMorphologyImageFilter<*TInputImage*, *TOutputImage*, *TKernel*>

Fast binary dilation of a single intensity value in the image.

BinaryDilateImageFilter is a binary dilation morphologic operation on the foreground of an image. Only the value designated by the intensity value "SetForegroundValue()" (alias as SetDilateValue()) is considered as foreground, and other intensity values are considered background.

Grayscale images can be processed as binary images by selecting a "ForegroundValue" (alias "DilateValue"). Pixel values matching the dilate value are considered the "foreground" and all other pixels are "background". This is useful in processing segmented images where all pixels in segment #1 have value 1 and pixels in segment #2 have value 2, etc. A particular "segment number" can be processed. ForegroundValue defaults to the maximum possible value of the PixelType.

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel. A reasonable choice of structuring element is itk::BinaryBallStructuringElement.

This implementation is based on the papers:

L.Vincent "Morphological transformations of binary images with

arbitrary structuring elements", and

N.Nikopoulos et al. "An efficient algorithm for 3d binary morphological transformations with 3d structuring elements for arbitrary size and shape". IEEE Transactions on Image Processing. Vol. 9. No. 3. 2000. pp. 283-286.

**See** ImageToImageFilter BinaryErodeImageFilter BinaryMorphologyImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Dilate A Binary Image

Subclassed by itk::FastIncrementalBinaryDilateImageFilter< TInputImage, TOutputImage, TKernel >

See itk::BinaryDilateImageFilter for additional documentation.

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class BinaryBallStructuringElement** : **public** itk::Neighborhood<*TPixel*, *VDimension*, *TAllocator*>

A Neighborhood that represents a ball structuring element (ellipsoid) with binary elements.

This class defines a Neighborhood whose elements are either off or on depending on whether they are outside or inside an ellipsoid whose radii match the radii of the Neighborhood. This class can be used as a structuring element for the Morphology image filters.

A BinaryBallStructuringElement has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D BinaryBallStructuringElement object with a radius of 2x3 has sides of length 5x7.

BinaryBallStructuringElement objects always have an unambiguous center because their side lengths are always odd.

Internally, this class carries out all of its computations using the FlatStructuringElement. It is preferable to use that class instead of this one because FlatStructuringElement is more flexible.

**See** Neighborhood

**See** MorphologyImageFilter

**See** BinaryDilateImageFilter

**See** BinaryErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Binary Ball Structuring Element

See itk::BinaryBallStructuringElement for additional documentation.

### Erode a Binary Image

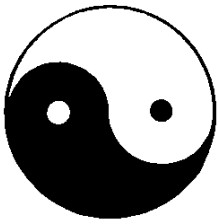
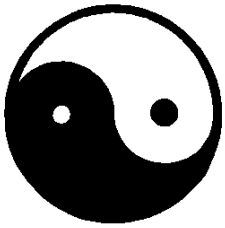
**See also:**

erosion; dilation

### Synopsis

Erode regions by using a specified kernel, also known as a structuring element. In this example, the white regions shrink.

### Results

| | |
|---|---|
|  |  |
| Input binary image. | Eroded output image. |

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Erode A Binary Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

StructuringElementType = itk.FlatStructuringElement[Dimension]
structuringElement = StructuringElementType.Ball(args.radius)
```

(continues on next page)

(continued from previous page)

```
ErodeFilterType = itk.BinaryErodeImageFilter[
    ImageType, ImageType, StructuringElementType
]
erodeFilter = ErodeFilterType.New()
erodeFilter.SetInput(reader.GetOutput())
erodeFilter.SetKernel(structuringElement)
erodeFilter.SetForegroundValue(255)  # Intensity value to erode
erodeFilter.SetBackgroundValue(0)  # Replacement value for eroded voxels

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(erodeFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkBinaryErodeImageFilter.h"
#include "itkFlatStructuringElement.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <inputImage> <outputImage> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char *       inputImage = argv[1];
  const char *       outputImage = argv[2];
  const unsigned int radiusValue = std::stoi(argv[3]);

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);
  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using BinaryErodeImageFilterType = itk::BinaryErodeImageFilter<ImageType, ImageType,
↪ StructuringElementType>;
```

(continues on next page)

(continued from previous page)

```
  BinaryErodeImageFilterType::Pointer erodeFilter = BinaryErodeImageFilterType::New();
  erodeFilter->SetInput(reader->GetOutput());
  erodeFilter->SetKernel(structuringElement);
  erodeFilter->SetForegroundValue(255); // Intensity value to erode
  erodeFilter->SetBackgroundValue(0);   // Replacement value for eroded voxels

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(erodeFilter->GetOutput());
  writer->SetFileName(outputImage);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class BinaryErodeImageFilter** : **public** itk::BinaryMorphologyImageFilter<*TInputImage*, *TOutputImage*, *TKernel*>

Fast binary erosion of a single intensity value in the image.

BinaryErodeImageFilter is a binary erosion morphologic operation on the foreground of an image. Only the value designated by the intensity value "SetForegroundValue()" (alias as SetErodeValue()) is considered as foreground, and other intensity values are considered background.

Grayscale images can be processed as binary images by selecting a "ForegroundValue" (alias "ErodeValue"). Pixel values matching the erode value are considered the "foreground" and all other pixels are "background". This is useful in processing segmented images where all pixels in segment #1 have value 1 and pixels in segment #2 have value 2, etc. A particular "segment number" can be processed. ForegroundValue defaults to the maximum possible value of the PixelType. The eroded pixels will receive the BackgroundValue (defaults to NumericTraits::NonpositiveMin() ).

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel. A reasonable choice of structuring element is itk::BinaryBallStructuringElement.

This implementation is based on the papers:

L.Vincent "Morphological transformations of binary images with

arbitrary structuring elements", and

N.Nikopoulos et al. "An efficient algorithm for 3d binary morphological transformations with 3d structuring elements for arbitrary size and shape". IEEE Transactions on Image Processing. Vol. 9. No. 3. 2000. pp. 283-286.

**See** ImageToImageFilter BinaryDilateImageFilter BinaryMorphologyImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Erode A Binary Image

See itk::BinaryErodeImageFilter for additional documentation.

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class BinaryBallStructuringElement** : **public** itk::Neighborhood<*TPixel*, *VDimension*, *TAllocator*>

A Neighborhood that represents a ball structuring element (ellipsoid) with binary elements.

This class defines a Neighborhood whose elements are either off or on depending on whether they are outside or inside an ellipsoid whose radii match the radii of the Neighborhood. This class can be used as a structuring element for the Morphology image filters.

A BinaryBallStructuringElement has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D BinaryBallStructuringElement object with a radius of 2x3 has sides of length 5x7.

BinaryBallStructuringElement objects always have an unambiguous center because their side lengths are always odd.

Internally, this class carries out all of its computations using the FlatStructuringElement. It is preferable to use that class instead of this one because FlatStructuringElement is more flexible.

**See** Neighborhood

**See** MorphologyImageFilter

**See** BinaryDilateImageFilter

**See** BinaryErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Binary Ball Structuring Element

See itk::BinaryBallStructuringElement for additional documentation.

### Opening a Binary Image

### Synopsis

Opening a binary image.

### Results

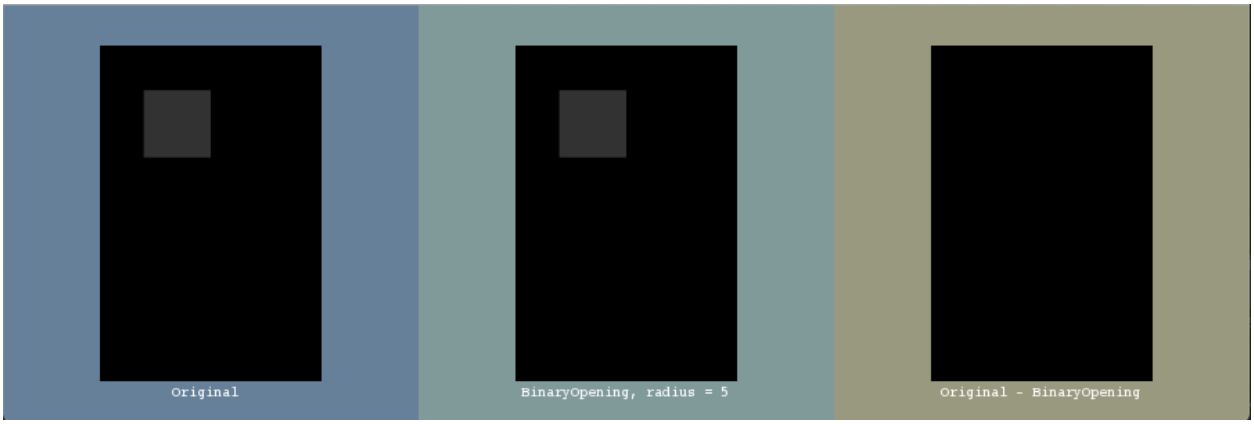



Fig. 95: Output In VTK Window

Output:

```
Radius: 5
```

### Code

### C++

```
#include "itkImage.h"
#include "itkBinaryMorphologicalOpeningImageFilter.h"
#include "itkImageFileReader.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkSubtractImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
}

static void
CreateImage(ImageType * const image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image;

  if (argc == 1)
  {
    image = ImageType::New();
    CreateImage(image);
```

(continues on next page)

(continued from previous page)

```
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  unsigned int radius = 5;
  if (argc == 3)
  {
    std::stringstream ss(argv[2]);
    ss >> radius;
  }
  std::cout << "Radius: " << radius << std::endl;
  using StructuringElementType = itk::BinaryBallStructuringElement<ImageType::
↪PixelType, ImageType::ImageDimension>;
  StructuringElementType structuringElement;
  structuringElement.SetRadius(radius);
  structuringElement.CreateStructuringElement();

  using BinaryMorphologicalOpeningImageFilterType =
    itk::BinaryMorphologicalOpeningImageFilter<ImageType, ImageType,␣
↪StructuringElementType>;
  BinaryMorphologicalOpeningImageFilterType::Pointer openingFilter =␣
↪BinaryMorphologicalOpeningImageFilterType::New();
  openingFilter->SetInput(image);
  openingFilter->SetKernel(structuringElement);
  openingFilter->Update();

  using SubtractType = itk::SubtractImageFilter<ImageType>;
  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(image);
  diff->SetInput2(openingFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView          viewer;
  std::stringstream desc;
  desc << "Original ";
  viewer.AddImage(image.GetPointer(), true, desc.str());

  std::stringstream desc2;
  desc2 << "BinaryOpening, radius = " << radius;
  viewer.AddImage(openingFilter->GetOutput(), true, desc2.str());

  std::stringstream desc3;
  desc3 << "Original - BinaryOpening";
  viewer.AddImage(diff->GetOutput(), true, desc3.str());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}


void
```

(continues on next page)

(continued from previous page)

```
CreateImage(ImageType * const image)
{
  // Create an image with 2 connected components
  itk::Index<2> corner = { { 0, 0 } };

  itk::Size<2> size;
  unsigned int NumRows = 200;
  unsigned int NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  itk::ImageRegion<2> region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      itk::Index<2> pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 50);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class BinaryMorphologicalOpeningImageFilter** : **public** itk::KernelImageFilter<*TInputImage*, *TOutputImage*, *TKern*

binary morphological opening of an image.

This filter removes small (i.e., smaller than the structuring element) structures in the interior or at the boundaries of the image. The morphological opening of an image "f" is defined as: Opening(f) = Dilatation(Erosion(f)).

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel.

This code was contributed in the Insight Journal paper: "Binary morphological closing and opening image filters" by Lehmann G. https://www.insight-journal.org/browse/publication/58

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** MorphologyImageFilter, GrayscaleDilateImageFilter, GrayscaleErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Opening A Binary Image

See itk::BinaryMorphologicalOpeningImageFilter for additional documentation.

### Prune Binary Image

### Synopsis

Prune a binary image.

### Results

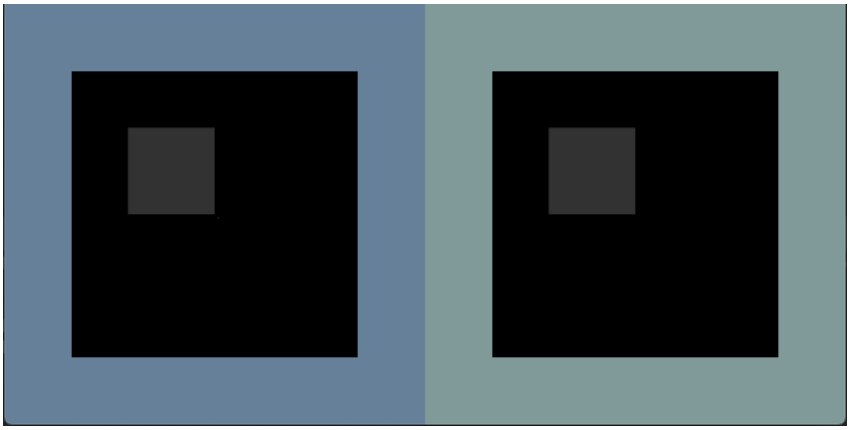


Fig. 96: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkBinaryPruningImageFilter.h"
#include "itkImageFileReader.h"
#include "itkBinaryBallStructuringElement.h"
#include "itkImageFileWriter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

template <typename TImage>
void
CreateImage(TImage * const image);

int
main(int argc, char * argv[])
{
  //   std::cerr << "Usage: " << std::endl;
  //   std::cerr << argv[0] << " InputImageFile OutputImageFile [iteration]" << std::
↪endl;

  using ImageType = itk::Image<unsigned char, 2>;
  ImageType::Pointer image;
```

(continues on next page)

(continued from previous page)

```
  unsigned int iteration = 1;

  if (argc < 3)
  {
    image = ImageType::New();
    CreateImage(image.GetPointer());
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    image = reader->GetOutput();
    std::stringstream ssIteration(argv[2]);
  }

  using BinaryPruningImageFilterType = itk::BinaryPruningImageFilter<ImageType,␣
↪ImageType>;
  BinaryPruningImageFilterType::Pointer pruneFilter = BinaryPruningImageFilterType::
↪New();
  pruneFilter->SetInput(image);
  pruneFilter->SetIteration(iteration);
  pruneFilter->GetOutput();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image.GetPointer());
  viewer.AddImage(pruneFilter->GetOutput());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

template <typename TImage>
void
CreateImage(TImage * const image)
{
  // This function creates a 2D image consisting of a black background,
  // a large square of a non-zero pixel value, and a single "erroneous" pixel
  // near the square.
  typename TImage::IndexType corner = { { 0, 0 } };

  typename TImage::SizeType size = { { 200, 200 } };

  typename TImage::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (int r = 40; r < 100; r++)
  {
    for (int c = 40; c < 100; c++)
    {
      typename TImage::IndexType pixelIndex = { { r, c } };
      image->SetPixel(pixelIndex, 50);
```

(continues on next page)

(continued from previous page)

```
    }
  }

  typename TImage::IndexType pixelIndex = { { 102, 102 } };

  image->SetPixel(pixelIndex, 50);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryPruningImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
This filter removes "spurs" of less than a certain length in the input image.

This class is parameterized over the type of the input image and the type of the output image.

The input is assumed to be a binary image.

This filter is a sequential pruning algorithm and known to be computational time dependable of the image size. The algorithm is the N-dimensional version of that given for two dimensions in:

Rafael C. Gonzales and Richard E. Woods. Digital Image Processing. Addison Wesley, 491-494, (1993).

**See** MorphologyImageFilter

**See** BinaryErodeImageFilter

**See** BinaryDilateImageFilter

**See** BinaryThinningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Prune Binary Image

See itk::BinaryPruningImageFilter for additional documentation.

### Thin Image

### Synopsis

Skeletonix/thin an image.

### Results

Fig. 97: input.png

Fig. 98: output.png

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Thin Image.")
parser.add_argument("input_image", nargs="?")
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2
ImageType = itk.Image[PixelType, Dimension]

if args.input_image:
    image = itk.imread(args.input_image)

else:
    # Create an image
    start = itk.Index[Dimension]()
    start.Fill(0)

    size = itk.Size[Dimension]()
    size.Fill(100)

    region = itk.ImageRegion[Dimension]()
    region.SetIndex(start)
    region.SetSize(size)

    image = ImageType.New(Regions=region)
    image.Allocate()
    image.FillBuffer(0)
```

(continues on next page)

(continued from previous page)

```
    # Draw a 5 pixel wide line
    image[50:55, 20:80] = 255

    # Write Image
    itk.imwrite(image, "input.png")

image = itk.binary_thinning_image_filter(image)

# Rescale the image so that it can be seen (the output is 0 and 1, we want 0 and 255)
image = itk.rescale_intensity_image_filter(image, output_minimum=0, output_
↪maximum=255)

# Write Image
itk.imwrite(image, "outputPython.png")
```

### C++

```
#include "itkBinaryThinningImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
WriteImage(const ImageType::Pointer image, const std::string & fileName);
static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  if (argc == 2)
  {
    using ImageReader = itk::ImageFileReader<ImageType>;
    ImageReader::Pointer reader = ImageReader::New();
    std::string          fileName = argv[1];
    reader->SetFileName(fileName);
    reader->Update();
    image = reader->GetOutput();
  }
  else
  {
    CreateImage(image);
    WriteImage(image, "input.png");
  }

  using BinaryThinningImageFilterType = itk::BinaryThinningImageFilter<ImageType,␣
↪ImageType>;
  BinaryThinningImageFilterType::Pointer binaryThinningImageFilter =␣
↪BinaryThinningImageFilterType::New();
  binaryThinningImageFilter->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  binaryThinningImageFilter->Update();

  // Rescale the image so that it can be seen (the output is 0 and 1, we want 0 and
↪255)
  using RescaleType = itk::RescaleIntensityImageFilter<ImageType, ImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(binaryThinningImageFilter->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(255);
  rescaler->Update();

  WriteImage(rescaler->GetOutput(), "output.png");

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Draw a 5 pixel wide line
  for (unsigned int i = 20; i < 80; ++i)
  {
    for (unsigned int j = 50; j < 55; ++j)
    {
      itk::Index<2> index;
      index[0] = i;
      index[1] = j;
      image->SetPixel(index, 255);
    }
  }
}

void
WriteImage(const ImageType::Pointer image, const std::string & fileName)
{
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(fileName);
  writer->SetInput(image);
  writer->Update();
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryThinningImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

This filter computes one-pixel-wide edges of the input image.

This class is parameterized over the type of the input image and the type of the output image.

The input is assumed to be a binary image. If the foreground pixels of the input image do not have a value of 1, they are rescaled to 1 internally to simplify the computation.

The filter will produce a skeleton of the object. The output background values are 0, and the foreground values are 1.

This filter is a sequential thinning algorithm and known to be computational time dependable on the image size. The algorithm corresponds with the 2D implementation described in:

Rafael C. Gonzales and Richard E. Woods. Digital Image Processing. Addison Wesley, 491-494, (1993).

To do: Make this filter ND.

**See** MorphologyImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Thin Image

See itk::BinaryThinningImageFilter for additional documentation.

### 3.4.4 Colormap

#### Apply a Color Map to an Image

#### Synopsis

Apply a color map to an image

#### Results

#### Code

#### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Apply A Colormap To An Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()
```

(continues on next page)

Fig. 99: Input image

Fig. 100: Output image

(continued from previous page)

```
PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

RGBPixelType = itk.RGBPixel[PixelType]
RGBImageType = itk.Image[RGBPixelType, Dimension]

RGBFilterType = itk.ScalarToRGBColormapImageFilter[ImageType, RGBImageType]
rgbfilter = RGBFilterType.New()
rgbfilter.SetInput(reader.GetOutput())
rgbfilter.SetColormap(itk.ScalarToRGBColormapImageFilterEnums.RGBColormapFilter_Hot)

WriterType = itk.ImageFileWriter[RGBImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(rgbfilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkRGBPixel.h"
#include "itkScalarToRGBColormapImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
```

(continues on next page)

(continued from previous page)

```
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, Dimension>;

  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<ImageType, RGBImageType>;
  RGBFilterType::Pointer rgbfilter = RGBFilterType::New();
  rgbfilter->SetInput(reader->GetOutput());
  rgbfilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::RGBColormapFilter::
↪Hot);

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(rgbfilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ScalarToRGBColormapImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Implements pixel-wise intensity->rgb mapping operation on one image.

This class is parameterized over the type of the input image and the type of the output image.

The input image's scalar pixel values are mapped into a color map. The color map is specified by passing the SetColormap function one of the predefined maps. The following selects the "RGBColormapFilterEnum::Hot" colormap:

```
RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
colormapImageFilter->SetColormap( RGBFilterType::Hot );
```

You can also specify a custom color map. This is done by creating a CustomColormapFunction, and then creating lists of values for the red, green, and blue channel. An example of setting the red channel of a colormap with only 2 colors is given below. The blue and green channels should be specified in the same manner.

```
// Create the custom colormap
using ColormapType = itk::Function::CustomColormapFunction<RealImageType::
↪PixelType,
RGBImageType::PixelType>;
ColormapType::Pointer colormap = ColormapType::New();
// Setup the red channel of the colormap
ColormapType::ChannelType redChannel;
redChannel.push_back(0); redChannel.push_back(255);
colormap->SetRedChannel( channel );
```

The range of values present in the input image is the range that is mapped to the entire range of colors.

This code was contributed in the Insight Journal paper: "Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK" by Tustison N., Zhang H., Lehmann G., Yushkevich P., Gee J. https://www.insight-journal.org/browse/publication/285

**See** BinaryFunctionImageFilter TernaryFunctionImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Custom Colormap
- Apply A Colormap To An Image

See itk::ScalarToRGBColormapImageFilter for additional documentation.

### Create a Custom Color Map

### Synopsis

Create and apply a custom colormap.

---

**Note:** Python wrapping for CustomColormapFunction and the MersenneTwisterRandomVariateGenerator was introduced in ITK 4.8.0.

---

### Results

Fig. 101: Input image

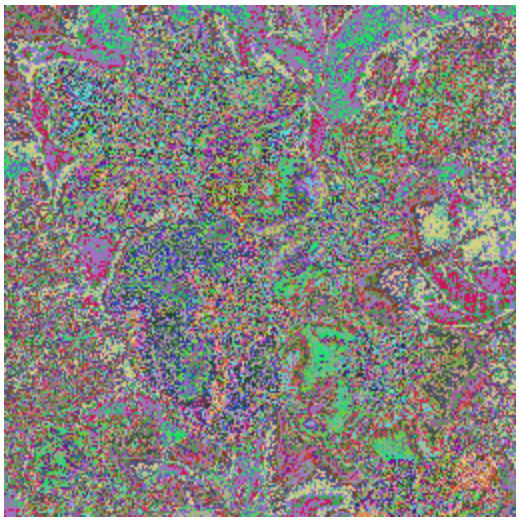


Fig. 102: Output image

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.8.0"):
    print("ITK 4.8.0 is required (see example documentation).")
    sys.exit(1)

parser = argparse.ArgumentParser(description="Create A Custom Colormap.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

RGBPixelType = itk.RGBPixel[PixelType]
RGBImageType = itk.Image[RGBPixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

ColormapType = itk.CustomColormapFunction[PixelType, RGBPixelType]
colormap = ColormapType.New()
```

(continues on next page)

(continued from previous page)

```
random = itk.MersenneTwisterRandomVariateGenerator.New()
random.SetSeed(0)

redChannel = []
greenChannel = []
blueChannel = []

for i in range(255):
    redChannel.append(random.GetUniformVariate(0.0, 1.0))
    greenChannel.append(random.GetUniformVariate(0.0, 1.0))
    blueChannel.append(random.GetUniformVariate(0.0, 1.0))

colormap.SetRedChannel(redChannel)
colormap.SetGreenChannel(greenChannel)
colormap.SetBlueChannel(blueChannel)

ColormapFilterType = itk.ScalarToRGBColormapImageFilter[ImageType, RGBImageType]
colormapFilter1 = ColormapFilterType.New()

colormapFilter1.SetInput(reader.GetOutput())
colormapFilter1.SetColormap(colormap)

WriterType = itk.ImageFileWriter[RGBImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(colormapFilter1.GetOutput())

writer.Update()
```

### C++

```
#include "itkCustomColormapFunction.h"
#include "itkScalarToRGBColormapImageFilter.h"

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCustomColormapFunction.h"
#include "itkScalarToRGBColormapImageFilter.h"
#include "itkRGBPixel.h"
#include "itkMersenneTwisterRandomVariateGenerator.h"


int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputFileName> <OutputFileName>" << std::endl;

    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  using PixelType = unsigned char;
  using RGBPixelType = itk::RGBPixel<unsigned char>;

  using RGBImageType = itk::Image<RGBPixelType, 2>;
  using ImageType = itk::Image<PixelType, 2>;


  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using ColormapType = itk::Function::CustomColormapFunction<PixelType, RGBPixelType>;
  ColormapType::Pointer colormap = ColormapType::New();

  ColormapType::ChannelType redChannel;
  ColormapType::ChannelType greenChannel;
  ColormapType::ChannelType blueChannel;

  itk::Statistics::MersenneTwisterRandomVariateGenerator::Pointer random =
    itk::Statistics::MersenneTwisterRandomVariateGenerator::New();

  random->SetSeed(0);

  for (unsigned int i = 0; i < 255; ++i)
  {
    redChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(0., 1.)));

    greenChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(0., 1.)));

    blueChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(0., 1.)));
  }

  colormap->SetRedChannel(redChannel);
  colormap->SetGreenChannel(greenChannel);
  colormap->SetBlueChannel(blueChannel);

  using ColormapFilterType = itk::ScalarToRGBColormapImageFilter<ImageType,␣
↪RGBImageType>;
  ColormapFilterType::Pointer colormapFilter1 = ColormapFilterType::New();

  colormapFilter1->SetInput(reader->GetOutput());
  colormapFilter1->SetColormap(colormap);

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(colormapFilter1->GetOutput());
  writer->SetFileName(argv[2]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
```

(continues on next page)

(continued from previous page)

```
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TScalar**, typename **TRGBPixel**>
**class CustomColormapFunction** : **public** itk::Function::ColormapFunction<*TScalar*, *TRGBPixel*>
Function object which maps a scalar value into an RGB colormap value.

This code was contributed in the Insight Journal paper:

**Author** Nicholas Tustison, Hui Zhang, Gaetan Lehmann, Paul Yushkevich and James C. Gee

"Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK" https://www.insight-journal.org/browse/publication/285

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Custom Colormap
- Apply A Colormap To An Image

See itk::Function::CustomColormapFunction for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class ScalarToRGBColormapImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Implements pixel-wise intensity->rgb mapping operation on one image.

This class is parameterized over the type of the input image and the type of the output image.

The input image's scalar pixel values are mapped into a color map. The color map is specified by passing the SetColormap function one of the predefined maps. The following selects the "RGBColormapFilterEnum::Hot" colormap:

```
RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
colormapImageFilter->SetColormap( RGBFilterType::Hot );
```

You can also specify a custom color map. This is done by creating a CustomColormapFunction, and then creating lists of values for the red, green, and blue channel. An example of setting the red channel of a colormap with only 2 colors is given below. The blue and green channels should be specified in the same manner.

```
// Create the custom colormap
using ColormapType = itk::Function::CustomColormapFunction<RealImageType::
↪PixelType,
RGBImageType::PixelType>;
ColormapType::Pointer colormap = ColormapType::New();
// Setup the red channel of the colormap
ColormapType::ChannelType redChannel;
```

(continues on next page)

(continued from previous page)

```
redChannel.push_back(0); redChannel.push_back(255);
colormap->SetRedChannel( channel );
```

The range of values present in the input image is the range that is mapped to the entire range of colors.

This code was contributed in the Insight Journal paper: "Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK" by Tustison N., Zhang H., Lehmann G., Yushkevich P., Gee J. https://www.insight-journal.org/browse/publication/285

**See** BinaryFunctionImageFilter TernaryFunctionImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Custom Colormap
- Apply A Colormap To An Image

See itk::ScalarToRGBColormapImageFilter for additional documentation.

**class MersenneTwisterRandomVariateGenerator** : **public** itk::Statistics::RandomVariateGeneratorBase
MersenneTwisterRandom random variate generator.

It is recommended to create a separate object in each thread. By default, each instantiated class will have a different seed created by the GetNextSeed method. The creation of the initial seeds are initialized once from the time. For deterministic behavior, the individual instances' seeds should be manual set to separate values in each thread.

It is no longer recommended to use this class using a "Singleton-like" GetInstance method for the global instance of this class. This usage may result in unsafe concurrent access to the global instance.

This notice was included with the original implementation. The only changes made were to obfuscate the author's email addresses.

**Warning** This class's instance methods are NEITHER reentrant nor concurrent thread-safe, except where marked as thread-safe. That is to say you can still use separate objects concurrently.

MersenneTwister.h Mersenne Twister random number generator a C++ class MTRand Based on code by Makoto Matsumoto, Takuji Nishimura, and Shawn Cokus Richard J. Wagner v1.0 15 May 2003 rjwagner at writeme dot com

The Mersenne Twister is an algorithm for generating random numbers. It was designed with consideration of the flaws in various other generators. The period, 2^19937-1, and the order of equidistribution, 623 dimensions, are far greater. The generator is also fast; it avoids multiplication and division, and it benefits from caches and pipelines. For more information see the inventors' web page at http:*www.math.keio.ac.jp/~matumoto/emt.html

Reference M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally

Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30.



Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

a. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

b. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

c. The names of its contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The original code included the following notice:

```
When you use this, send an email to: matumoto at math dot keio dot ac dot jp
with an appropriate reference to your work.
```

It would be nice to CC: rjwagner at writeme dot com and Cokus at math dot washington dot edu when you write.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mersenne Twister Random Number Generator

See itk::Statistics::MersenneTwisterRandomVariateGenerator for additional documentation.

### Map Scalars Into Jet Colormap

### Synopsis

Map scalars into a jet colormap.

### Results

Output:

```
0: 0  0  142
0.5: 134  255  119
1: 126  0  0
```

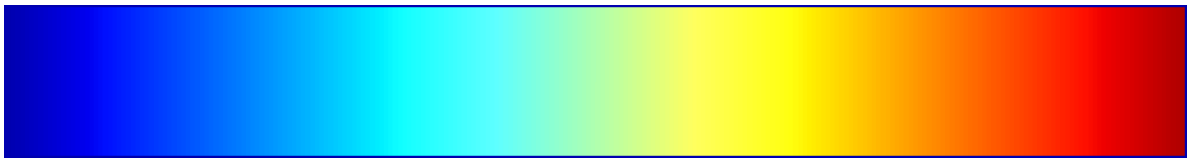
### Code

### C++

```
#include "itkJetColormapFunction.h"
#include "itkRGBPixel.h"

int
main(int, char *[])
{
  using PixelType = itk::RGBPixel<unsigned char>;
  using ColormapType = itk::Function::JetColormapFunction<float, PixelType>;
  ColormapType::Pointer colormap = ColormapType::New();

  colormap->SetMinimumInputValue(0.0);
  colormap->SetMaximumInputValue(1.0);
  std::cout << "0: " << colormap->  operator()(0.0f) << std::endl;
  std::cout << "0.5: " << colormap->operator()(0.5f) << std::endl;
  std::cout << "1: " << colormap->  operator()(1.0f) << std::endl;
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TScalar**, typename **TRGBPixel**>
**class JetColormapFunction** : **public** itk::Function::ColormapFunction<*TScalar*, *TRGBPixel*>
Function object which maps a scalar value into an RGB colormap value.



This code was contributed in the Insight Journal paper:

**Author** Nicholas Tustison, Hui Zhang, Gaetan Lehmann, Paul Yushkevich and James C. Gee

"Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK" https://www.insight-journal.org/browse/publication/285

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Map Scalars Into Jet Colormap

See itk::Function::JetColormapFunction for additional documentation.

### 3.4.5 Convolution

#### Color Normalized Correlation

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

#### Synopsis

Color normalized correlation.

#### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

#### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkNormalizedCorrelationImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"
#include "itkImageKernelOperator.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"
#include "itkCovariantVector.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

#include <iostream>
#include <string>

// Vector types
using FloatVectorType = itk::CovariantVector<float, 3>;
using UnsignedCharVectorType = itk::CovariantVector<unsigned char, 3>;

// Vector image types
using FloatVectorImageType = itk::Image<FloatVectorType, 2>;
using UnsignedCharVectorImageType = itk::Image<UnsignedCharVectorType, 2>;

// Scalar image types
using FloatImageType = itk::Image<float, 2>;
using UnsignedCharImageType = itk::Image<unsigned char, 2>;
```

(continues on next page)

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
    return EXIT_FAILURE;
  }

  std::string filename = argv[1];

  using ReaderType = itk::ImageFileReader<FloatVectorImageType>;

  // Read the image
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(filename.c_str());
  reader->Update();

  // Extract a small region
  using ExtractFilterType = itk::RegionOfInterestImageFilter<FloatVectorImageType,␣
↪FloatVectorImageType>;

  ExtractFilterType::Pointer extractFilter = ExtractFilterType::New();

  FloatImageType::IndexType start;
  start.Fill(50);

  FloatImageType::SizeType patchSize;
  patchSize.Fill(51);

  FloatImageType::RegionType desiredRegion(start, patchSize);

  extractFilter->SetRegionOfInterest(desiredRegion);
  extractFilter->SetInput(reader->GetOutput());
  extractFilter->Update();

  // Perform normalized correlation
  // <input type, mask type (not used), output type>
  using CorrelationFilterType =
    itk::NormalizedCorrelationImageFilter<FloatVectorImageType, FloatVectorImageType,␣
↪FloatImageType>;

  itk::ImageKernelOperator<FloatVectorType> kernelOperator;
  kernelOperator.SetImageKernel(extractFilter->GetOutput());

  // The radius of the kernel must be the radius of the patch, NOT the size of the␣
↪patch
  itk::Size<2> radius = extractFilter->GetOutput()->GetLargestPossibleRegion().
↪GetSize();
  radius[0] = (radius[0] - 1) / 2;
  radius[1] = (radius[1] - 1) / 2;

  kernelOperator.CreateToRadius(radius);

  CorrelationFilterType::Pointer correlationFilter = CorrelationFilterType::New();
  correlationFilter->SetInput(reader->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  correlationFilter->SetTemplate(kernelOperator);
  correlationFilter->Update();

  using MinimumMaximumImageCalculatorType = itk::MinimumMaximumImageCalculator
↪<FloatImageType>;

  MinimumMaximumImageCalculatorType::Pointer minimumMaximumImageCalculatorFilter =
    MinimumMaximumImageCalculatorType::New();
  minimumMaximumImageCalculatorFilter->SetImage(correlationFilter->GetOutput());
  minimumMaximumImageCalculatorFilter->Compute();

  itk::Index<2> maximumCorrelationPatchCenter = minimumMaximumImageCalculatorFilter->
↪GetIndexOfMaximum();
  std::cout << "Maximum: " << maximumCorrelationPatchCenter << std::endl;

  // Note that the best correlation is at the center of the patch we extracted (ie.␣
↪(75, 75) rather than the corner
  // (50,50)

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  using WriterType = itk::ImageFileWriter<UnsignedCharVectorImageType>;
  {
    RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
    rescaleFilter->SetInput(correlationFilter->GetOutput());
    rescaleFilter->SetOutputMinimum(0);
    rescaleFilter->SetOutputMaximum(255);
    rescaleFilter->Update();

    WriterType::Pointer writer = WriterType::New();
    writer->SetInput(rescaleFilter->GetOutput());
    writer->SetFileName("correlation.png");
    writer->Update();
  }

  {
    RescaleFilterType::Pointer rescaleFilter = RescaleVectorFilterType::New();
    rescaleFilter->SetInput(extractFilter->GetOutput());
    rescaleFilter->SetOutputMinimum(0);
    rescaleFilter->SetOutputMaximum(255);
    rescaleFilter->Update();

    WriterType::Pointer writer = WriterType::New();
    writer->SetInput(rescaleFilter->GetOutput());
    writer->SetFileName("patch.png");
    writer->Update();
  }

  // Extract the best matching patch
  FloatImageType::IndexType bestPatchStart;
  bestPatchStart[0] = maximumCorrelationPatchCenter[0] - radius[0];
  bestPatchStart[1] = maximumCorrelationPatchCenter[1] - radius[1];

  FloatImageType::RegionType bestPatchRegion(bestPatchStart, patchSize);

  ExtractFilterType::Pointer bestPatchExtractFilter = ExtractFilterType::New();
  bestPatchExtractFilter->SetRegionOfInterest(bestPatchRegion);
```

(continues on next page)

(continued from previous page)

```
  bestPatchExtractFilter->SetInput(reader->GetOutput());
  bestPatchExtractFilter->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput());
  viewer.AddImage(extractFilter->GetOutput());
  viewer.AddImage(correlationFilter->GetOutput());
  viewer.AddImage(bestPatchExtractFilter->GetOutput());
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage**, typename **TOperatorValueType** = **t**
**class NormalizedCorrelationImageFilter** : **public** itk::*NeighborhoodOperatorImageFilter*<*TInputImage*, *TOutputIma*

Computes the normalized correlation of an image and a template.

This filter calculates the normalized correlation between an image and the template. Normalized correlation is frequently use in feature detection because it is invariant to local changes in contrast.

The filter can be given a mask. When presented with an input image and a mask, the normalized correlation is only calculated at those pixels under the mask.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalized Correlation
- Normalized Correlation Of Masked Image
- Color Normalized Operation

See itk::NormalizedCorrelationImageFilter for additional documentation.

### Convolve Image With Kernel

### Synopsis

Convolve an image with a kernel.

### Results

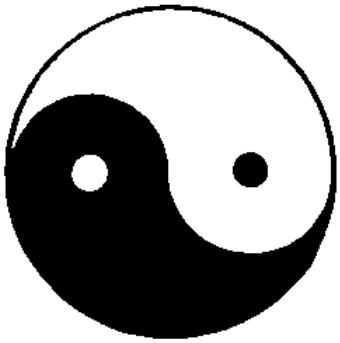


Fig. 103: Input image.

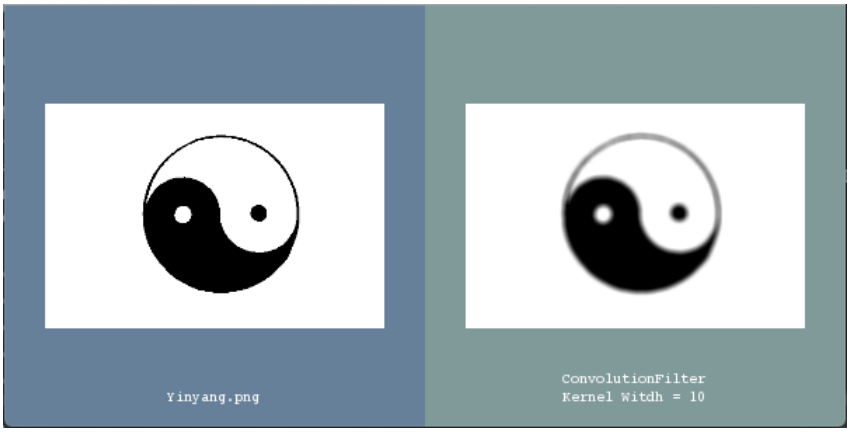



Fig. 104: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConvolutionImageFilter.h"
#include "itkImageRegionIterator.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using ImageType = itk::Image<float, 2>;

static void
CreateKernel(ImageType::Pointer kernel, unsigned int width);

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: ";
    std::cerr << argv[0] << "inputImageFile [width]" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  unsigned int width = 3;
  if (argc > 2)
  {
    width = std::stoi(argv[2]);
  }

  ImageType::Pointer kernel = ImageType::New();
  CreateKernel(kernel, width);

  using ReaderType = itk::ImageFileReader<ImageType>;
  using FilterType = itk::ConvolutionImageFilter<ImageType>;

  // Create and setup a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  // Convolve image with kernel.
  FilterType::Pointer convolutionFilter = FilterType::New();
  convolutionFilter->SetInput(reader->GetOutput());
#if ITK_VERSION_MAJOR >= 4
  convolutionFilter->SetKernelImage(kernel);
#else
  convolutionFilter->SetImageKernelInput(kernel);
#endif
#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage<ImageType>(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));
```

(continues on next page)

(continued from previous page)

```
  std::stringstream desc;
  desc << "ConvolutionFilter\n"
       << "Kernel Witdh = " << width;
  viewer.AddImage<ImageType>(convolutionFilter->GetOutput(), true, desc.str());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateKernel(ImageType::Pointer kernel, unsigned int width)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(width);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  kernel->SetRegions(region);
  kernel->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(kernel, region);

  while (!imageIterator.IsAtEnd())
  {
    // imageIterator.Set(255);
    imageIterator.Set(1);

    ++imageIterator;
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TKernelImage** = *TInputImage*, typename **TOutputImage** = *TInputImage*>
**class ConvolutionImageFilter** : **public** itk::ConvolutionImageFilterBase<*TInputImage*, *TKernelImage*, *TOutputImage*>

Convolve a given image with an arbitrary image kernel.

This filter operates by centering the flipped kernel at each pixel in the image and computing the inner product between pixel values in the image and pixel values in the kernel. The center of the kernel is defined as $\lfloor (2 * i + s - 1)/2 \rfloor$ where $i$ is the index and $s$ is the size of the largest possible region of the kernel image. For kernels with odd sizes in all dimensions, this corresponds to the center pixel. If a dimension of the kernel image has an even size, then the center index of the kernel in that dimension will be the largest integral index that is less than the continuous index of the image center.

The kernel can optionally be normalized to sum to 1 using NormalizeOn(). Normalization is off by default.

This code was contributed in the Insight Journal paper:

**Warning** This filter ignores the spacing, origin, and orientation of the kernel image and treats them as identical to those in the input image.

"Image Kernel Convolution" by Tustison N., Gee J. http://insight-journal.org/browse/publication/208

**Author** Nicholas J. Tustison

**Author** James C. Gee

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convolve Image With Kernel

See itk::ConvolutionImageFilter for additional documentation.

### Normalized Correlation

### Synopsis

Normalized correlation.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkNormalizedCorrelationImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"
#include "itkImageKernelOperator.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"
#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

#include <iostream>
#include <string>

using FloatImageType = itk::Image<float, 2>;
using UnsignedCharImageType = itk::Image<unsigned char, 2>;

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Required: filename" << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  std::string filename = argv[1];

  using ReaderType = itk::ImageFileReader<FloatImageType>;

  // Read the image
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(filename.c_str());
  reader->Update();

  // Extract a small region
  using ExtractFilterType = itk::RegionOfInterestImageFilter<FloatImageType,␣
↪FloatImageType>;

  ExtractFilterType::Pointer extractFilter = ExtractFilterType::New();

  FloatImageType::IndexType start;
  start.Fill(50);

  FloatImageType::SizeType patchSize;
  patchSize.Fill(51);

  FloatImageType::RegionType desiredRegion(start, patchSize);

  extractFilter->SetRegionOfInterest(desiredRegion);
  extractFilter->SetInput(reader->GetOutput());
  extractFilter->Update();

  // Perform normalized correlation
  // <input type, mask type (not used), output type>
  using CorrelationFilterType = itk::NormalizedCorrelationImageFilter<FloatImageType,␣
↪FloatImageType, FloatImageType>;

  itk::ImageKernelOperator<float> kernelOperator;
  kernelOperator.SetImageKernel(extractFilter->GetOutput());

  // The radius of the kernel must be the radius of the patch, NOT the size of the␣
↪patch
  itk::Size<2> radius = extractFilter->GetOutput()->GetLargestPossibleRegion().
↪GetSize();
  radius[0] = (radius[0] - 1) / 2;
  radius[1] = (radius[1] - 1) / 2;

  kernelOperator.CreateToRadius(radius);

  CorrelationFilterType::Pointer correlationFilter = CorrelationFilterType::New();
  correlationFilter->SetInput(reader->GetOutput());
  correlationFilter->SetTemplate(kernelOperator);
  correlationFilter->Update();

  using MinimumMaximumImageCalculatorType = itk::MinimumMaximumImageCalculator
↪<FloatImageType>;

  MinimumMaximumImageCalculatorType::Pointer minimumMaximumImageCalculatorFilter =
    MinimumMaximumImageCalculatorType::New();
```

(continues on next page)

(continued from previous page)

```
  minimumMaximumImageCalculatorFilter->SetImage(correlationFilter->GetOutput());
  minimumMaximumImageCalculatorFilter->Compute();

  itk::Index<2> maximumCorrelationPatchCenter = minimumMaximumImageCalculatorFilter->
↪GetIndexOfMaximum();
  std::cout << "Maximum: " << maximumCorrelationPatchCenter << std::endl;

  // Note that the best correlation is at the center of the patch we extracted (ie.␣
↪(75, 75) rather than the corner
  // (50,50)

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  {
    RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
    rescaleFilter->SetInput(correlationFilter->GetOutput());
    rescaleFilter->SetOutputMinimum(0);
    rescaleFilter->SetOutputMaximum(255);
    rescaleFilter->Update();

    WriterType::Pointer writer = WriterType::New();
    writer->SetInput(rescaleFilter->GetOutput());
    writer->SetFileName("correlation.png");
    writer->Update();
  }

  {
    RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
    rescaleFilter->SetInput(extractFilter->GetOutput());
    rescaleFilter->SetOutputMinimum(0);
    rescaleFilter->SetOutputMaximum(255);
    rescaleFilter->Update();

    WriterType::Pointer writer = WriterType::New();
    writer->SetInput(rescaleFilter->GetOutput());
    writer->SetFileName("patch.png");
    writer->Update();
  }

  // Extract the best matching patch
  FloatImageType::IndexType bestPatchStart;
  bestPatchStart[0] = maximumCorrelationPatchCenter[0] - radius[0];
  bestPatchStart[1] = maximumCorrelationPatchCenter[1] - radius[1];

  FloatImageType::RegionType bestPatchRegion(bestPatchStart, patchSize);

  ExtractFilterType::Pointer bestPatchExtractFilter = ExtractFilterType::New();
  bestPatchExtractFilter->SetRegionOfInterest(bestPatchRegion);
  bestPatchExtractFilter->SetInput(reader->GetOutput());
  bestPatchExtractFilter->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput());
  viewer.AddImage(extractFilter->GetOutput());
  viewer.AddImage(correlationFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  viewer.AddImage(bestPatchExtractFilter->GetOutput());
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage**, typename **TOperatorValueType** = **t**
**class NormalizedCorrelationImageFilter** : **public** itk::*NeighborhoodOperatorImageFilter<TInputImage*, *TOutputIma*
Computes the normalized correlation of an image and a template.

This filter calculates the normalized correlation between an image and the template. Normalized correlation is frequently use in feature detection because it is invariant to local changes in contrast.

The filter can be given a mask. When presented with an input image and a mask, the normalized correlation is only calculated at those pixels under the mask.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalized Correlation
- Normalized Correlation Of Masked Image
- Color Normalized Operation

See itk::NormalizedCorrelationImageFilter for additional documentation.

### Normalized Correlation of Masked Image

### Synopsis

Normalized correlation of a masked image.

### Results

.

Fig. 105: mask.png

Fig. 106: image1.png

Fig. 107: image2.png

Output:

```
Maximum location fixed: [20, 6]
Maximum value: 1
```

### Code

### C++

```
// Maximum value is inf??

#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkNormalizedCorrelationImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"
#include "itkImageKernelOperator.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"

#include <iostream>
#include <string>

using ImageType = itk::Image<unsigned char, 2>;
using MaskType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

void
CreateMask(MaskType * const mask);
void
CreateImage(ImageType * const image);
void
CreateImageOfSquare(ImageType * const image, const itk::Index<2> & cornerOfSquare);
```

(continues on next page)

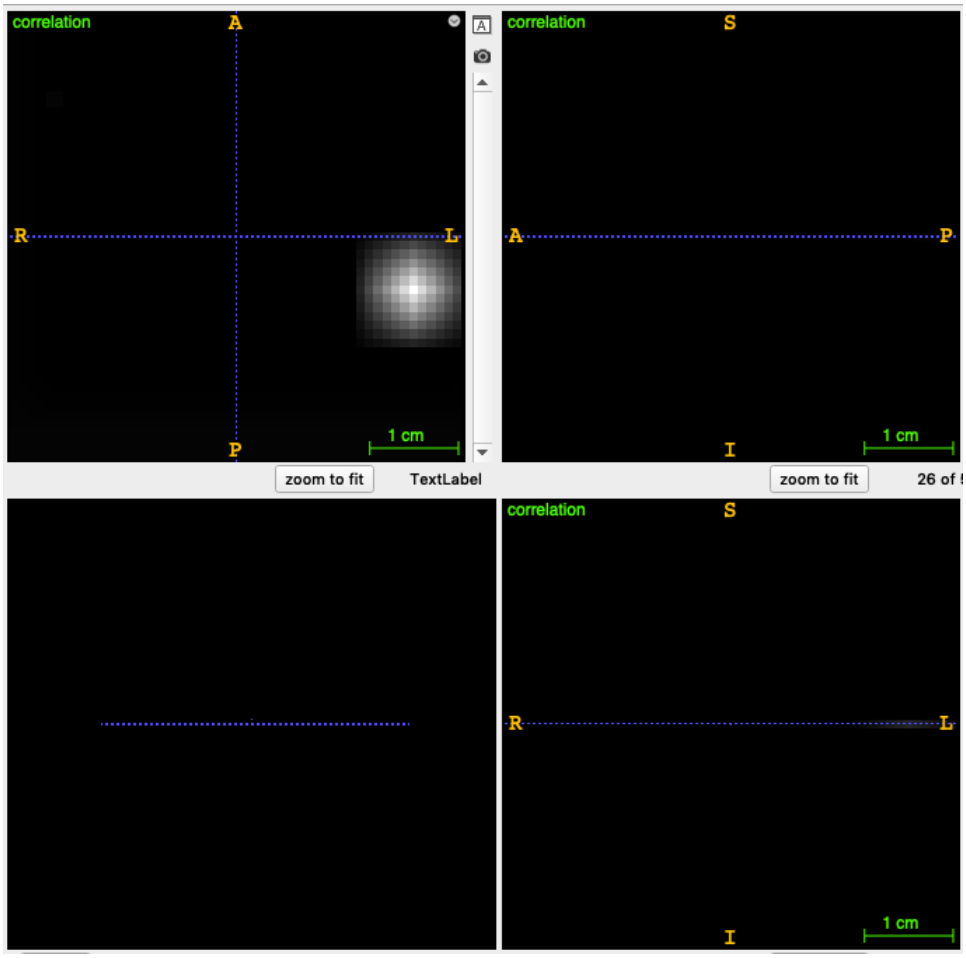



Fig. 108: correlation.mha

Fig. 109: correlation.png

(continued from previous page)

```
template <typename TImage>
void
WriteImage(const TImage * const image, const std::string & filename);

int
main(int, char *[])
{
  // Setup mask
  MaskType::Pointer mask = MaskType::New();
  CreateMask(mask);

  WriteImage(mask.GetPointer(), "mask.png");

  // Setup image1
  ImageType::Pointer image1 = ImageType::New();
  itk::Index<2>      cornerOfSquare1;
  cornerOfSquare1[0] = 3;
  cornerOfSquare1[1] = 8;
  CreateImageOfSquare(image1, cornerOfSquare1);
  WriteImage(image1.GetPointer(), "image1.png");

  // Setup image2
  itk::Index<2> offset;
  offset[0] = 20;
  offset[1] = 6;

  ImageType::Pointer image2 = ImageType::New();
  itk::Index<2>      cornerOfSquare2;
  cornerOfSquare2[0] = cornerOfSquare1[0] + offset[0];
  cornerOfSquare2[1] = cornerOfSquare1[1] + offset[1];
  CreateImageOfSquare(image2, cornerOfSquare2);

  WriteImage(image2.GetPointer(), "image2.png");

  // Create a kernel from an image
  itk::ImageKernelOperator<unsigned char> kernelOperator;
  kernelOperator.SetImageKernel(image1);

  // The radius of the kernel must be the radius of the patch, NOT the size of the␣
↪patch
  itk::Size<2> radius;
  radius[0] = image1->GetLargestPossibleRegion().GetSize()[0] / 2;
  radius[1] = image1->GetLargestPossibleRegion().GetSize()[1] / 2;

  if (radius[0] % 2 == 0 || radius[1] % 2 == 0)
  {
    std::cerr << "Input must have odd dimensions!" << std::endl;
    return EXIT_FAILURE;
  }

  kernelOperator.CreateToRadius(radius);

  // Perform normalized correlation
  // <input type, mask type, output type>
  using CorrelationFilterType =
    itk::NormalizedCorrelationImageFilter<ImageType, MaskType, FloatImageType,␣
↪unsigned char>;
```

(continues on next page)

(continued from previous page)

```
  CorrelationFilterType::Pointer correlationFilter = CorrelationFilterType::New();
  correlationFilter->SetInput(image2);
  correlationFilter->SetMaskImage(mask);
  correlationFilter->SetTemplate(kernelOperator);
  correlationFilter->Update();

  WriteImage(correlationFilter->GetOutput(), "correlation.mha");

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪ImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(correlationFilter->GetOutput());
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();
  WriteImage(rescaleFilter->GetOutput(), "correlation.png");

  using MinimumMaximumImageCalculatorType = itk::MinimumMaximumImageCalculator
↪<FloatImageType>;
  MinimumMaximumImageCalculatorType::Pointer minimumMaximumImageCalculatorFilter =
    MinimumMaximumImageCalculatorType::New();
  minimumMaximumImageCalculatorFilter->SetImage(correlationFilter->GetOutput());
  minimumMaximumImageCalculatorFilter->Compute();

  itk::Index<2> maximumCorrelationPatchCenter = minimumMaximumImageCalculatorFilter->
↪GetIndexOfMaximum();

  // This is the value we expect!
  std::cout << "Maximum location fixed: " << maximumCorrelationPatchCenter - radius <
↪< std::endl;

  // If the images can be perfectly aligned, the value is 1
  std::cout << "Maximum value: " << minimumMaximumImageCalculatorFilter->GetMaximum()
↪<< std::endl;

  return EXIT_SUCCESS;
}

void
CreateMask(MaskType * const mask)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  mask->SetRegions(region);
  mask->Allocate();
  mask->FillBuffer(255); // Make the whole mask "valid"

  // unsigned int squareSize = 8;
  unsigned int squareSize = 3;

  itk::Index<2> cornerOfSquare = { { 3, 8 } };
```

(continues on next page)

(continued from previous page)

```
  // Remove pixels from the mask in a small square. The correlationw will not be␣
↪computed at these pixels.
  itk::ImageRegionIterator<MaskType> maskIterator(mask, region);

  while (!maskIterator.IsAtEnd())
  {
    if (maskIterator.GetIndex()[0] > cornerOfSquare[0] && maskIterator.GetIndex()[0]
↪< cornerOfSquare[0] + squareSize &&
        maskIterator.GetIndex()[1] > cornerOfSquare[1] && maskIterator.GetIndex()[1]
↪< cornerOfSquare[1] + squareSize)
    {
      maskIterator.Set(0);
    }

    ++maskIterator;
  }
}

void
CreateImage(ImageType * const image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);
}

void
CreateImageOfSquare(ImageType * const image, const itk::Index<2> & cornerOfSquare)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  unsigned int squareSize = 8;

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > cornerOfSquare[0] &&
```

(continues on next page)

(continued from previous page)

```
        imageIterator.GetIndex()[0] < cornerOfSquare[0] + squareSize &&
        imageIterator.GetIndex()[1] > cornerOfSquare[1] && imageIterator.
↪GetIndex()[1] < cornerOfSquare[1] + squareSize)
    {
      imageIterator.Set(255);
    }

    ++imageIterator;
  }
}

template <typename TImage>
void
WriteImage(const TImage * const image, const std::string & filename)
{
  using WriterType = itk::ImageFileWriter<TImage>;
  typename WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(filename);
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage**, typename **TOperatorValueType** = **t**
**class NormalizedCorrelationImageFilter** : **public** itk::*NeighborhoodOperatorImageFilter<TInputImage*, *TOutputIma*

Computes the normalized correlation of an image and a template.

This filter calculates the normalized correlation between an image and the template. Normalized correlation is frequently use in feature detection because it is invariant to local changes in contrast.

The filter can be given a mask. When presented with an input image and a mask, the normalized correlation is only calculated at those pixels under the mask.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalized Correlation
- Normalized Correlation Of Masked Image
- Color Normalized Operation

See itk::NormalizedCorrelationImageFilter for additional documentation.

### Normalized Correlation Using FFT

### Synopsis

Normalized correlation using the FFT.

### Results

Fig. 110: fixedImage.png

Fig. 111: movingImage.png

Output:

```
Maximum location: [45, 44]
Maximum location fixed: [5, 6]
Maximum value: 1
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkFFTNormalizedCorrelationImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"
#include "itkImageKernelOperator.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"

#include <iostream>
#include <string>

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;
} // namespace

// using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image, const itk::Index<2> & cornerOfSquare);
```

(continues on next page)

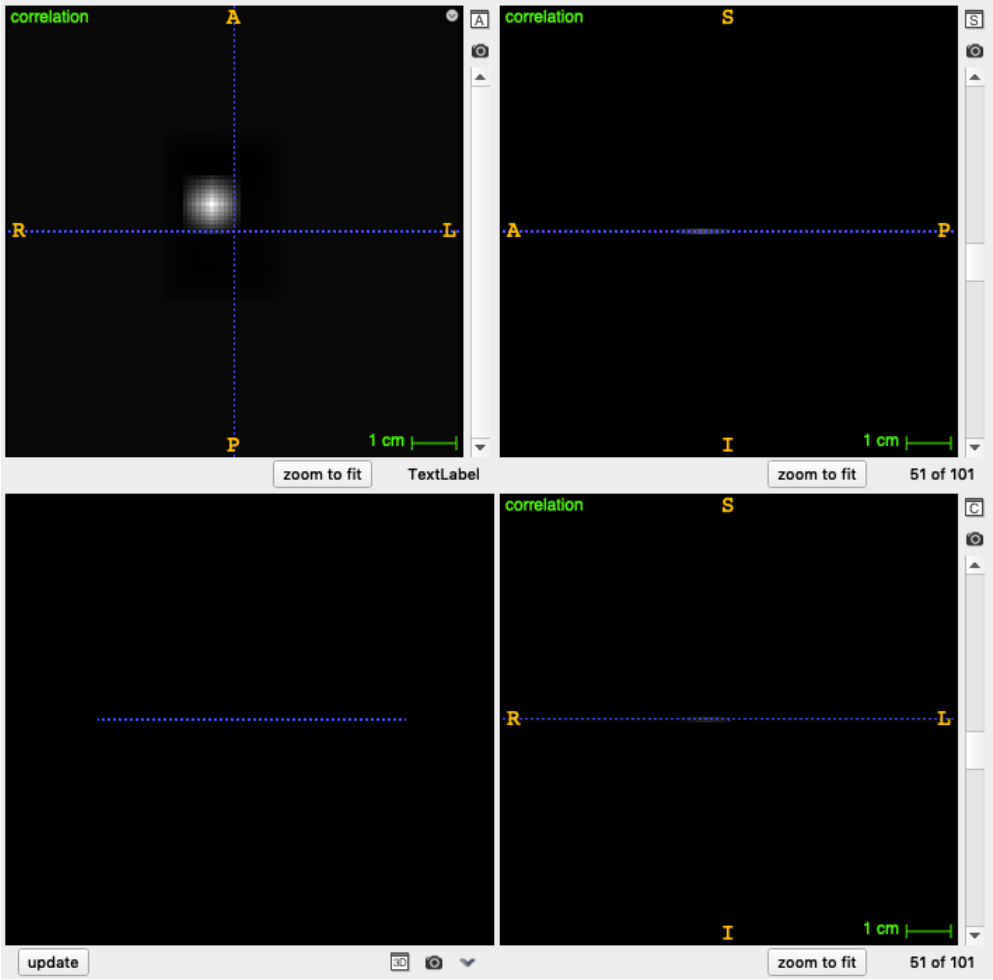



Fig. 112: correlation.mha

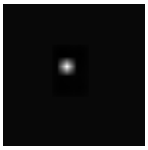


Fig. 113: correlation.png

(continued from previous page)

```
template <typename TImage>
static void
WriteImage(TImage * const image, const std::string & filename);

int
main(int, char *[])
{
  itk::Index<2> offset;
  offset[0] = 5;
  offset[1] = 6;

  ImageType::Pointer fixedImage = ImageType::New();
  itk::Index<2>      cornerOfFixedSquare;
  cornerOfFixedSquare[0] = 3;
  cornerOfFixedSquare[1] = 8;
  CreateImage(fixedImage, cornerOfFixedSquare);
  WriteImage(fixedImage.GetPointer(), "fixedImage.png");

  ImageType::Pointer movingImage = ImageType::New();
  itk::Index<2>      cornerOfMovingSquare;
  cornerOfMovingSquare[0] = cornerOfFixedSquare[0] + offset[0];
  cornerOfMovingSquare[1] = cornerOfFixedSquare[1] + offset[1];
  CreateImage(movingImage, cornerOfMovingSquare);
  WriteImage(movingImage.GetPointer(), "movingImage.png");

  // Perform normalized correlation
  using CorrelationFilterType = itk::FFTNormalizedCorrelationImageFilter<ImageType,␣
↪FloatImageType>;
  CorrelationFilterType::Pointer correlationFilter = CorrelationFilterType::New();
  correlationFilter->SetFixedImage(fixedImage);
  correlationFilter->SetMovingImage(movingImage);
  correlationFilter->Update();

  WriteImage(correlationFilter->GetOutput(), "correlation.mha");

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪ImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(correlationFilter->GetOutput());
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();
  WriteImage(rescaleFilter->GetOutput(), "correlation.png");

  using MinimumMaximumImageCalculatorType = itk::MinimumMaximumImageCalculator
↪<FloatImageType>;
  MinimumMaximumImageCalculatorType::Pointer minimumMaximumImageCalculatorFilter =
    MinimumMaximumImageCalculatorType::New();
  minimumMaximumImageCalculatorFilter->SetImage(correlationFilter->GetOutput());
  minimumMaximumImageCalculatorFilter->Compute();

  itk::Index<2> maximumCorrelationPatchCenter = minimumMaximumImageCalculatorFilter->
↪GetIndexOfMaximum();

  itk::Size<2> outputSize = correlationFilter->GetOutput()->
↪GetLargestPossibleRegion().GetSize();
```

(continues on next page)

(continued from previous page)

```
  itk::Index<2> maximumCorrelationPatchCenterFixed;
  maximumCorrelationPatchCenterFixed[0] = outputSize[0] / 2 -␣
↪maximumCorrelationPatchCenter[0];
  maximumCorrelationPatchCenterFixed[1] = outputSize[1] / 2 -␣
↪maximumCorrelationPatchCenter[1];

  std::cout << "Maximum location: " << maximumCorrelationPatchCenter << std::endl;
  std::cout << "Maximum location fixed: " << maximumCorrelationPatchCenterFixed
            << std::endl; // This is the value we expect!
  std::cout << "Maximum value: " << minimumMaximumImageCalculatorFilter->GetMaximum()
            << std::endl; // If the images can be perfectly aligned, the value is 1

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image, const itk::Index<2> & cornerOfSquare)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  ImageType::IndexValueType squareSize = 8;

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > cornerOfSquare[0] &&
        imageIterator.GetIndex()[0] < cornerOfSquare[0] + squareSize &&
        imageIterator.GetIndex()[1] > cornerOfSquare[1] && imageIterator.
↪GetIndex()[1] < cornerOfSquare[1] + squareSize)
    {
      imageIterator.Set(255);
    }

    ++imageIterator;
  }
}

template <typename TImage>
void
WriteImage(TImage * const image, const std::string & filename)
{
  using WriterType = itk::ImageFileWriter<TImage>;
  typename WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(filename);
  writer->SetInput(image);
  writer->Update();
```

(continues on next page)

(continued from previous page)

```
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class FFTNormalizedCorrelationImageFilter** : **public** itk::*MaskedFFTNormalizedCorrelationImageFilter<TInputIm*
Calculate normalized cross correlation using FFTs.

This filter calculates the normalized cross correlation (NCC) of two images using FFTs instead of spatial correlation. It is much faster than spatial correlation for reasonably large structuring elements. This filter is a subclass of the more general MaskedFFTNormalizedCorrelationImageFilter and operates by essentially setting the masks in that algorithm to images of ones. As described in detail in the references below, there is no computational overhead to utilizing the more general masked algorithm because the FFTs of the images of ones are still necessary for the computations.

Inputs: Two images are required as inputs, fixedImage and movingImage. In the context of correlation, inputs are often defined as: "image" and "template". In this filter, the fixedImage plays the role of the image, and the movingImage plays the role of the template. However, this filter is capable of correlating any two images and is not restricted to small movingImages (templates).

Optional parameters: The RequiredNumberOfOverlappingPixels enables the user to specify how many voxels of the two images must overlap; any location in the correlation map that results from fewer than this number of voxels will be set to zero. Larger values zero-out pixels on a larger border around the correlation image. Thus, larger values remove less stable computations but also limit the capture range. If RequiredNumberOfOverlappingPixels is set to 0, the default, no zeroing will take place.

Image size: fixedImage and movingImage need not be the same size. Furthermore, whereas some algorithms require that the "template" be smaller than the "image" because of errors in the regions where the two are not fully overlapping, this filter has no such restriction.

Image spacing: Since the computations are done in the pixel domain, all input images must have the same spacing.

Outputs; The output is an image of RealPixelType that is the NCC of the two images and its values range from -1.0 to 1.0. The size of this NCC image is, by definition, size(fixedImage) + size(movingImage) - 1.

Example filter usage:

```
using FilterType = itk::FFTNormalizedCorrelationImageFilter< ShortImageType,␣
↪DoubleImageType >;
FilterType::Pointer filter = FilterType::New();
filter->SetFixedImage( fixedImage );
filter->SetMovingImage( movingImage );
filter->SetRequiredNumberOfOverlappingPixels(20);
filter->Update();
```

References: 1) D. Padfield. "Masked object registration in the Fourier domain." Transactions on Image Processing. 2) D. Padfield. "Masked FFT registration". In Proc. Computer Vision and Pattern Recognition, 2010.

**Warning** The pixel type of the output image must be of real type (float or double). ConceptChecking is used to enforce the output pixel type. You will get a compilation error if the pixel type of the output image is not float or double.

**Author** : Dirk Padfield, GE Global Research, padfield@research.ge.com

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalized Correlation Using FFT

See itk::FFTNormalizedCorrelationImageFilter for additional documentation.

### Normalized Correlation Using FFT With Mask Images for Input Images

### Synopsis

Normalized correlation using the FFT with optional mask images for both input images.

### Results

Fig. 114: fixedImage.png

Fig. 115: movingImage.png

Output:

```
Maximum location: [45, 44]
Maximum location fixed: [5, 6]
Maximum value: 1
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkFFTNormalizedCorrelationImageFilter.h"
#include "itkRegionOfInterestImageFilter.h"
#include "itkImageKernelOperator.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkMinimumMaximumImageCalculator.h"

#include <iostream>
#include <string>

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
```

(continues on next page)

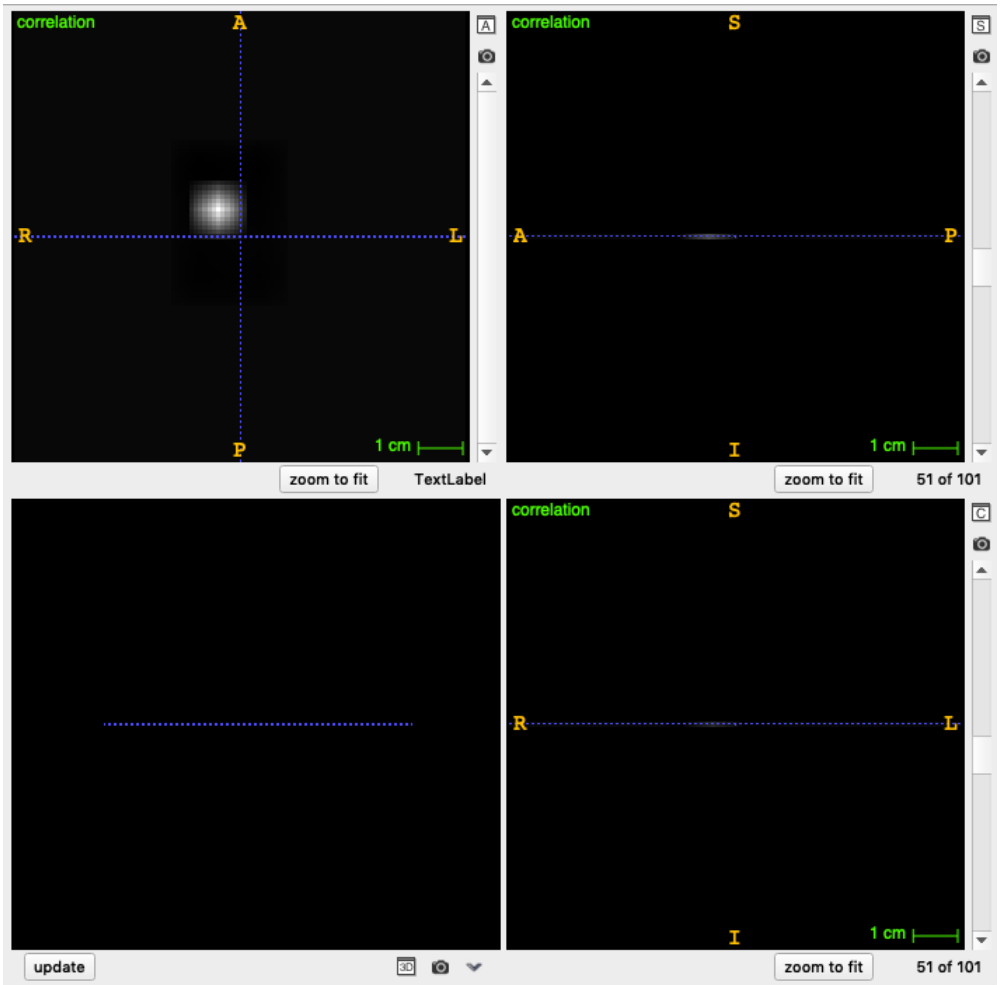



Fig. 116: correlation.mha

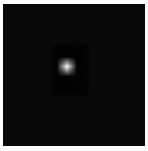


Fig. 117: correlation.png

(continued from previous page)

```
using FloatImageType = itk::Image<float, 2>;
using MaskType = itk::Image<unsigned char, 2>;
} // namespace

static void
CreateMask(MaskType * const mask);
static void
CreateImage(ImageType::Pointer image, const itk::Index<2> & cornerOfSquare);

template <typename TImage>
void
WriteImage(TImage * const image, const std::string & filename);

int
main(int, char *[])
{
  itk::Index<2> offset;
  offset[0] = 5;
  offset[1] = 6;

  // Setup mask
  MaskType::Pointer mask = MaskType::New();
  CreateMask(mask);

  ImageType::Pointer fixedImage = ImageType::New();
  itk::Index<2>      cornerOfFixedSquare;
  cornerOfFixedSquare[0] = 3;
  cornerOfFixedSquare[1] = 8;
  CreateImage(fixedImage, cornerOfFixedSquare);
  WriteImage(fixedImage.GetPointer(), "fixedImage.png");

  ImageType::Pointer movingImage = ImageType::New();
  itk::Index<2>      cornerOfMovingSquare;
  cornerOfMovingSquare[0] = cornerOfFixedSquare[0] + offset[0];
  cornerOfMovingSquare[1] = cornerOfFixedSquare[1] + offset[1];
  CreateImage(movingImage, cornerOfMovingSquare);
  WriteImage(movingImage.GetPointer(), "movingImage.png");

  // Perform normalized correlation
  using CorrelationFilterType = itk::FFTNormalizedCorrelationImageFilter<ImageType,␣
↪FloatImageType>;
  CorrelationFilterType::Pointer correlationFilter = CorrelationFilterType::New();
  correlationFilter->SetFixedImage(fixedImage);
  correlationFilter->SetMovingImage(movingImage);
  correlationFilter->SetMovingImageMask(mask);
  // correlationFilter->SetFixedImageMask(mask);
  correlationFilter->Update();

  WriteImage(correlationFilter->GetOutput(), "correlation.mha");

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪ImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(correlationFilter->GetOutput());
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  WriteImage(rescaleFilter->GetOutput(), "correlation.png");

  using MinimumMaximumImageCalculatorType = itk::MinimumMaximumImageCalculator
↪<FloatImageType>;
  MinimumMaximumImageCalculatorType::Pointer minimumMaximumImageCalculatorFilter =
    MinimumMaximumImageCalculatorType::New();
  minimumMaximumImageCalculatorFilter->SetImage(correlationFilter->GetOutput());
  minimumMaximumImageCalculatorFilter->Compute();

  itk::Index<2> maximumCorrelationPatchCenter = minimumMaximumImageCalculatorFilter->
↪GetIndexOfMaximum();

  itk::Size<2> outputSize = correlationFilter->GetOutput()->
↪GetLargestPossibleRegion().GetSize();

  itk::Index<2> maximumCorrelationPatchCenterFixed;
  maximumCorrelationPatchCenterFixed[0] = outputSize[0] / 2 -␣
↪maximumCorrelationPatchCenter[0];
  maximumCorrelationPatchCenterFixed[1] = outputSize[1] / 2 -␣
↪maximumCorrelationPatchCenter[1];

  std::cout << "Maximum location: " << maximumCorrelationPatchCenter << std::endl;
  std::cout << "Maximum location fixed: " << maximumCorrelationPatchCenterFixed
            << std::endl; // This is the value we expect!
  std::cout << "Maximum value: " << minimumMaximumImageCalculatorFilter->GetMaximum()
            << std::endl; // If the images can be perfectly aligned, the value is 1

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image, const itk::Index<2> & cornerOfSquare)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  ImageType::IndexValueType squareSize = 8;

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > cornerOfSquare[0] &&
        imageIterator.GetIndex()[0] < cornerOfSquare[0] + squareSize &&
        imageIterator.GetIndex()[1] > cornerOfSquare[1] && imageIterator.
↪GetIndex()[1] < cornerOfSquare[1] + squareSize)
    {
      imageIterator.Set(255);
```

(continues on next page)

(continued from previous page)

```
    }

    ++imageIterator;
  }
}

template <typename TImage>
void
WriteImage(TImage * const image, const std::string & filename)
{
  using WriterType = itk::ImageFileWriter<TImage>;
  typename WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(filename);
  writer->SetInput(image);
  writer->Update();
}


void
CreateMask(MaskType * const mask)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(51);

  ImageType::RegionType region(start, size);

  mask->SetRegions(region);
  mask->Allocate();
  mask->FillBuffer(255); // Make the whole mask "valid"

  // unsigned int squareSize = 8;
  ImageType::IndexValueType squareSize = 3;

  itk::Index<2> cornerOfSquare = { { 3, 8 } };

  // Remove pixels from the mask in a small square. The correlationw will not be␣
↪computed at these pixels.
  itk::ImageRegionIterator<MaskType> maskIterator(mask, region);

  while (!maskIterator.IsAtEnd())
  {
    if (maskIterator.GetIndex()[0] > cornerOfSquare[0] && maskIterator.GetIndex()[0]
↪< cornerOfSquare[0] + squareSize &&
        maskIterator.GetIndex()[1] > cornerOfSquare[1] && maskIterator.GetIndex()[1]
↪< cornerOfSquare[1] + squareSize)
    {
      maskIterator.Set(0);
    }

    ++maskIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TInputImage*>
**class MaskedFFTNormalizedCorrelationImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImag*

Calculate masked normalized cross correlation using FFTs.

This filter calculates the masked normalized cross correlation (NCC) of two images under masks using FFTs instead of spatial correlation. It is much faster than spatial correlation for reasonably large structuring elements. This filter is not equivalent to simply masking the images first and then correlating them; the latter approach yields incorrect results because the zeros in the images still affect the metric in the correlation process. This filter implements the masked NCC correctly so that the masked-out regions are completely ignored. The fundamental difference is described in detail in the references below. If the masks are set to images of all ones, the result of this filter is the same as standard NCC.

Inputs: Two images are required as inputs, fixedImage and movingImage, and two are optional, fixedMask and movingMask. In the context of correlation, inputs are often defined as: "image" and "template". In this filter, the fixedImage plays the role of the image, and the movingImage plays the role of the template. However, this filter is capable of correlating any two images and is not restricted to small movingImages (templates). In the fixedMask and movingMask, non-zero positive values indicate locations of useful information in the corresponding image, whereas zero and negative values indicate locations that should be masked out (ignored). Internally, the masks are converted to have values of only 0 and 1. For each optional mask that is not set, the filter internally creates an image of ones, which is equivalent to not masking the image. Thus, if both masks are not set, the result will be equivalent to unmasked NCC. For example, if only a mask for the fixed image is needed, the movingMask can either not be set or can be set to an image of ones.

Optional parameters: The RequiredNumberOfOverlappingPixels enables the user to specify the minimum number of voxels of the two masks that must overlap; any location in the correlation map that results from fewer than this number of voxels will be set to zero. Larger values zero-out pixels on a larger border around the correlation image. Thus, larger values remove less stable computations but also limit the capture range. If RequiredNumberOfOverlappingPixels is set to 0, the default, no zeroing will take place.

The RequiredFractionOfOverlappingPixels enables the user to specify a fraction of the maximum number of overlapping pixels that need to overlap; any location in the correlation map that results from fewer than the product of this fraction and the internally computed maximum number of overlapping pixels will be set to zero. The value ranges between 0.0 and 1.0. This is very useful when the user does does not know beforehand the maximum number of pixels of the masks that will overlap. For example, when the masks have strange shapes, it is difficult to predict how the correlation of the masks will interact and what the maximum overlap will be. It is also useful when the mask shapes or sizes change because it is relative to the internally computed maximum of the overlap. Larger values zero-out pixels on a larger border around the correlation image. Thus, larger values remove less stable computations but also limit the capture range. Experiments have shown that a value between 0.1 and 0.6 works well for images with significant overlap and between 0.05 and 0.1 for images with little overlap (such as in stitching applications). If RequiredFractionOfOverlappingPixels is set to 0, the default, no zeroing will take place.

The user can either specify RequiredNumberOfOverlappingPixels or RequiredFractionOfOverlappingPixels (or both or none). Internally, the number of required pixels resulting from both of these methods is calculated and the one that gives the largest number of pixels is chosen. Since these both default to 0, if a user only sets one, the other is ignored.

Image size: fixedImage and movingImage need not be the same size, but fixedMask must be the same size as fixedImage, and movingMask must be the same size as movingImage. Furthermore, whereas some algorithms require that the "template" be smaller than the "image" because of errors in the regions where the two are not fully overlapping, this filter has no such restriction.

Image spacing: Since the computations are done in the pixel domain, all input images must have the same spacing.

Outputs; The output is an image of RealPixelType that is the masked NCC of the two images and its values range from -1.0 to 1.0. The size of this NCC image is, by definition, size(fixedImage) + size(movingImage) - 1.

Example filter usage:

```
using FilterType = itk::MaskedFFTNormalizedCorrelationImageFilter< ShortImageType,
↪ DoubleImageType >;
FilterType::Pointer filter = FilterType::New();
filter->SetFixedImage( fixedImage );
filter->SetMovingImage( movingImage );
filter->SetFixedImageMask( fixedMask );
filter->SetMovingImageMask( movingMask );
filter->SetRequiredNumberOfOverlappingPixels(20);
filter->Update();
```

References: 1) D. Padfield. "Masked object registration in the Fourier domain." Transactions on Image Processing. 2) D. Padfield. "Masked FFT registration". In Proc. Computer Vision and Pattern Recognition, 2010.

**Warning** The pixel type of the output image must be of real type (float or double). ConceptChecking is used to enforce the output pixel type. You will get a compilation error if the pixel type of the output image is not float or double.

**Author** : Dirk Padfield, GE Global Research, padfield@research.ge.com

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalized Correlation Using FFT With Mask Images For Input Images

See itk::MaskedFFTNormalizedCorrelationImageFilter for additional documentation.

### 3.4.6 CurvatureFlow

#### Binary Min and Max Curvature Flow of Binary Image

#### Synopsis

BinaryMinMaxCurvatureFlow a binary image.

#### Results

#### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkBinaryMinMaxCurvatureFlowImageFilter.h"
#include "itkSubtractImageFilter.h"

#include "itksys/SystemTools.hxx"
```

(continues on next page)

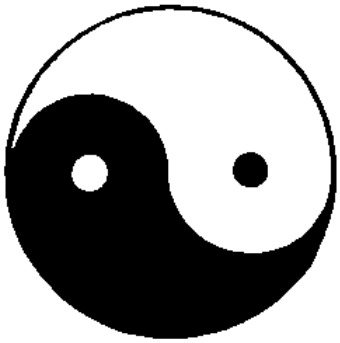


Fig. 118: Input image.

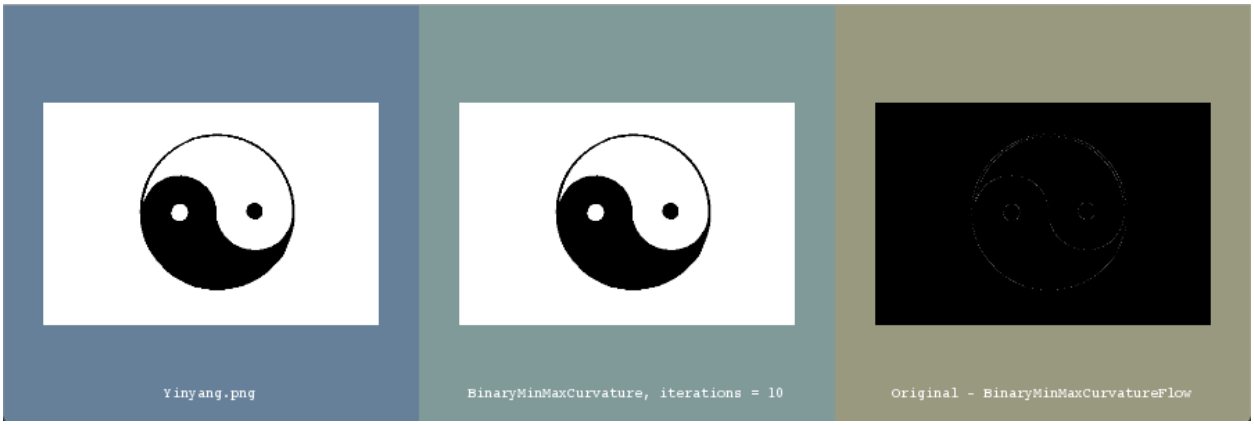



Fig. 119: Output In VTK Window

(continued from previous page)

```
#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << argv[0] << " InputFileName [NumberOfIterations]" << std ::endl;
    return EXIT_FAILURE;
  }

  std::string inputFileName = argv[1];

  unsigned int numberOfIterations = 2;
  if (argc > 2)
  {
    numberOfIterations = std::stoi(argv[2]);
  }

  constexpr unsigned int Dimension = 2;

  using InputPixelType = float;
  using OutputPixelType = float;

  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<InputImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::BinaryMinMaxCurvatureFlowImageFilter<InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetThreshold(255);
  filter->SetNumberOfIterations(numberOfIterations);

  using SubtractType = itk::SubtractImageFilter<OutputImageType>;
  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(filter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFileName));

  std::stringstream desc;
  desc << "BinaryMinMaxCurvature, iterations = " << numberOfIterations;
  viewer.AddImage(filter->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - BinaryMinMaxCurvatureFlow";
```

(continues on next page)

(continued from previous page)

```
  viewer.AddImage(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryMinMaxCurvatureFlowImageFilter** : **public** itk::*MinMaxCurvatureFlowImageFilter<TInputImage*, *TOut*

Denoise a binary image using min/max curvature flow.

BinaryMinMaxCurvatureFlowImageFilter implements a curvature driven image denoising algorithm. This filter assumes that the image is essentially binary: consisting of two classes. Iso-brightness contours in the input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = F_{\text{minmax}}|\nabla I|$$

where $F_{\text{minmax}} = \min(\kappa, 0)$ if $\text{Avg}_{\text{stencil}}(x)$ is less than or equal to $T_{threshold}$ and $\max(\kappa, 0)$, otherwise. $\kappa$ is the mean curvature of the iso-brightness contour at point $x$.

In min/max curvature flow, movement is turned on or off depending on the scale of the noise one wants to remove. Switching depends on the average image value of a region of radius $R$ around each point. The choice of $R$, the stencil radius, governs the scale of the noise to be removed.

The threshold value $T_{threshold}$ is a user specified value which discriminates between the two pixel classes.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a BinaryMinMaxCurvatureFlowFunction object. A zero flux Neumann boundary condition is used when computing derivatives near the data boundary.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a real type. This filter works for any dimensional images.

**See** BinaryMinMaxCurvatureFlowFunction

**See** CurvatureFlowImageFilter

**See** MinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Binary Min And Max Curvature Flow Of Binary Image

See itk::BinaryMinMaxCurvatureFlowImageFilter for additional documentation.

### Smooth Image Using Curvature Flow

### Synopsis

Smooth an image using curvature flow.

### Results

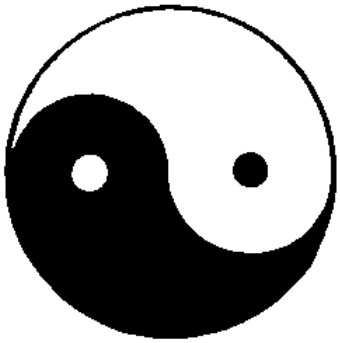


Fig. 120: Input image.


Fig. 121: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkCastImageFilter.h"
#include "itkCurvatureFlowImageFilter.h"
#include "itkSubtractImageFilter.h"
#include "itkImageFileReader.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0];
    std::cerr << " inputImage [iterations]" << std::endl;
    return EXIT_FAILURE;
  }

  int iterations = 5;
  if (argc > 2)
  {
    iterations = std::stoi(argv[2]);
  }

  using InternalPixelType = float;
  constexpr unsigned int Dimension = 2;
  using InternalImageType = itk::Image<InternalPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InternalImageType>;

  ReaderType::Pointer reader = ReaderType::New();

  reader->SetFileName(argv[1]);

  using CurvatureFlowImageFilterType = itk::CurvatureFlowImageFilter
↪<InternalImageType, InternalImageType>;

  CurvatureFlowImageFilterType::Pointer smoothing = CurvatureFlowImageFilterType::
↪New();

  smoothing->SetInput(reader->GetOutput());
  smoothing->SetNumberOfIterations(iterations);
  smoothing->SetTimeStep(0.125);

  using SubtractImageFilterType = itk::SubtractImageFilter<InternalImageType>;
  SubtractImageFilterType::Pointer diff = SubtractImageFilterType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(smoothing->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
```

(continues on next page)

(continued from previous page)

```
  viewer.AddImage<InternalImageType>(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  std::stringstream desc;
  desc << "CurvatureFlow\niterations = " << iterations;
  viewer.AddImage<InternalImageType>(smoothing->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - CurvatureFlow";
  viewer.AddImage<InternalImageType>(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CurvatureFlowImageFilter** : **public** itk::DenseFiniteDifferenceImageFilter<*TInputImage*, *TOutputImage*>

Denoise an image using curvature driven flow.

CurvatureFlowImageFilter implements a curvature driven image denoising algorithm. Iso-brightness contours in the grayscale input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = \kappa|\nabla I|$$

where $\kappa$ is the curvature.

The advantage of this approach is that sharp boundaries are preserved with smoothing occurring only within a region. However, it should be noted that continuous application of this scheme will result in the eventual removal of all information as each contour shrinks to zero and disappear.

Note that unlike level set segmentation algorithms, the image to be denoised is already the level set and can be set directly as the input using the SetInput() method.

This filter has two parameters: the number of update iterations to be performed and the timestep between each update.

The timestep should be "small enough" to ensure numerical stability. Stability is guarantee when the timestep meets the CFL (Courant-Friedrichs-Levy) condition. Broadly speaking, this condition ensures that each contour does not move more than one grid position at each timestep. In the literature, the timestep is typically user specified and have to manually tuned to the application.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a CurvatureFlowFunction object. A zero flux Neumann boundary condition when computing derivatives near the data boundary.

This filter may be streamed. To support streaming this filter produces a padded output which takes into account edge effects. The size of the padding is m_NumberOfIterations on each edge. Users of this filter should only make use of the center valid central region.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a floating point type. This filter works for any dimensional images.

Input/Output Restrictions: TInputImage and TOutputImage must have the same dimension. TOutputImage's pixel type must be a real number type.

**See** DenseFiniteDifferenceImageFilter

**See** CurvatureFlowFunction

**See** MinMaxCurvatureFlowImageFilter

**See** BinaryMinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image Using Curvature Flow
- Smooth RGB Image Using Curvature Flow

Subclassed by itk::MinMaxCurvatureFlowImageFilter< TInputImage, TOutputImage >

See itk::CurvatureFlowImageFilter for additional documentation.

### Smooth Image Using Min Max Curvature Flow

### Synopsis

Smooth an image using min/max curvature flow.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkCastImageFilter.h"
#include "itkMinMaxCurvatureFlowImageFilter.h"
#include "itkSubtractImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0];
```

(continues on next page)

Fig. 122: Input image.


Fig. 123: Output In VTK Window

(continued from previous page)

```
    std::cerr << " inputImage [iterations]" << std::endl;
    return EXIT_FAILURE;
  }

  std::string inputFileName = argv[1];

  int iterations = 5;
  if (argc > 2)
  {
    std::stringstream ss(argv[2]);
    ss >> iterations;
  }
  std::string inputFilename = argv[1];

  using PixelType = float;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  using MinMaxCurvatureFlowImageFilterType = itk::MinMaxCurvatureFlowImageFilter
↪<ImageType, ImageType>;
  using SubtractType = itk::SubtractImageFilter<ImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);

  MinMaxCurvatureFlowImageFilterType::Pointer minMaxCurvatureFlowImageFilter =
    MinMaxCurvatureFlowImageFilterType::New();
  minMaxCurvatureFlowImageFilter->SetInput(reader->GetOutput());
  minMaxCurvatureFlowImageFilter->SetNumberOfIterations(iterations);
  minMaxCurvatureFlowImageFilter->SetTimeStep(0.125);

  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(minMaxCurvatureFlowImageFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFilename));

  std::stringstream desc;
  desc << "MinMaxCurvatureFlow, iterations = " << iterations;
  viewer.AddImage(minMaxCurvatureFlowImageFilter->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - Median";
  viewer.AddImage(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MinMaxCurvatureFlowImageFilter** : **public** itk::*CurvatureFlowImageFilter<TInputImage, TOutputImage>*

Denoise an image using min/max curvature flow.

MinMaxCurvatureFlowImageFilter implements a curvature driven image denoising algorithm. Iso-brightness contours in the grayscale input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = F_{\text{minmax}}|\nabla I|$$

where $F_{\text{minmax}} = \max(\kappa, 0)$ if $\text{Avg}_{\text{stencil}}(x)$ is less than or equal to $T_{threshold}$ and $\min(\kappa, 0)$, otherwise. $\kappa$ is the mean curvature of the iso-brightness contour at point $x$.

In min/max curvature flow, movement is turned on or off depending on the scale of the noise one wants to remove. Switching depends on the average image value of a region of radius $R$ around each point. The choice of $R$, the stencil radius, governs the scale of the noise to be removed.

The threshold value $T_{threshold}$ is the average intensity obtained in the direction perpendicular to the gradient at point $x$ at the extrema of the local neighborhood.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a MinMaxCurvatureFlowFunction object. A zero flux Neumann boundary condition is used when computing derivatives near the data boundary.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a real type. This filter works for any dimensional images, however for dimensions greater than 3D, an expensive brute-force search is used to compute the local threshold.

**See** MinMaxCurvatureFlowFunction

**See** CurvatureFlowImageFilter

**See** BinaryMinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image Using Min Max Curvature Flow
- SmoothRGBImageUsingMinMaxCurvatureFlow

Subclassed by itk::BinaryMinMaxCurvatureFlowImageFilter< TInputImage, TOutputImage >

See itk::MinMaxCurvatureFlowImageFilter for additional documentation.

### Smooth RGB Image Using Curvature Flow

### Synopsis

Smooth an RGB image using curvature flow.

### Results

Fig. 124: Input image.


Fig. 125: Output In VTK Window

### Code

### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"
#include "itkNthElementImageAdaptor.h"
#include "itkCurvatureFlowImageFilter.h"
#include "itkComposeImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkRGBPixel.h"

#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " InputImageFile [iterations]" << std::endl;
    return EXIT_FAILURE;
  }

  int iterations = 5;
  if (argc > 2)
  {
    std::stringstream ss(argv[2]);
    ss >> iterations;
  }
  std::string inputFilename = argv[1];

  // Setup types
  using ComponentType = float;
  using PixelType = itk::RGBPixel<ComponentType>;
  using RGBImageType = itk::Image<PixelType, 2>;
  using ImageType = itk::Image<ComponentType, 2>;
  using ReaderType = itk::ImageFileReader<RGBImageType>;
  using ImageAdaptorType = itk::NthElementImageAdaptor<RGBImageType, unsigned char>;
  using ComposeType = itk::ComposeImageFilter<ImageType, RGBImageType>;
  using CurvatureFlowType = itk::CurvatureFlowImageFilter<ImageAdaptorType, ImageType>
↪;

  // Create and setup a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);
  reader->Update();

  // Run the filter for each component
  ImageAdaptorType::Pointer rAdaptor = ImageAdaptorType::New();
  rAdaptor->SelectNthElement(0);
```

(continues on next page)

(continued from previous page)

```
  rAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer rCurvatureFilter = CurvatureFlowType::New();
  rCurvatureFilter->SetInput(rAdaptor);
  rCurvatureFilter->SetNumberOfIterations(iterations);
  rCurvatureFilter->Update();

  ImageAdaptorType::Pointer gAdaptor = ImageAdaptorType::New();
  gAdaptor->SelectNthElement(1);
  gAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer gCurvatureFilter = CurvatureFlowType::New();
  gCurvatureFilter->SetInput(gAdaptor);
  gCurvatureFilter->SetNumberOfIterations(iterations);
  gCurvatureFilter->Update();

  ImageAdaptorType::Pointer bAdaptor = ImageAdaptorType::New();
  bAdaptor->SelectNthElement(2);
  bAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer bCurvatureFilter = CurvatureFlowType::New();
  bCurvatureFilter->SetInput(bAdaptor);
  bCurvatureFilter->SetNumberOfIterations(iterations);
  bCurvatureFilter->Update();

  // compose an RGB image from the three filtered images

  ComposeType::Pointer compose = ComposeType::New();
  compose->SetInput1(rCurvatureFilter->GetOutput());
  compose->SetInput2(gCurvatureFilter->GetOutput());
  compose->SetInput3(bCurvatureFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFilename));

  std::stringstream desc;
  desc << "CurvatureFlow iterations = " << iterations;
  viewer.AddRGBImage(compose->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CurvatureFlowImageFilter** : **public** itk::DenseFiniteDifferenceImageFilter<*TInputImage*, *TOutputImage*>
Denoise an image using curvature driven flow.

CurvatureFlowImageFilter implements a curvature driven image denoising algorithm. Iso-brightness contours in the grayscale input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = \kappa |\nabla I|$$

where $\kappa$ is the curvature.

The advantage of this approach is that sharp boundaries are preserved with smoothing occurring only within a region. However, it should be noted that continuous application of this scheme will result in the eventual removal of all information as each contour shrinks to zero and disappear.

Note that unlike level set segmentation algorithms, the image to be denoised is already the level set and can be set directly as the input using the SetInput() method.

This filter has two parameters: the number of update iterations to be performed and the timestep between each update.

The timestep should be "small enough" to ensure numerical stability. Stability is guarantee when the timestep meets the CFL (Courant-Friedrichs-Levy) condition. Broadly speaking, this condition ensures that each contour does not move more than one grid position at each timestep. In the literature, the timestep is typically user specified and have to manually tuned to the application.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a CurvatureFlowFunction object. A zero flux Neumann boundary condition when computing derivatives near the data boundary.

This filter may be streamed. To support streaming this filter produces a padded output which takes into account edge effects. The size of the padding is m_NumberOfIterations on each edge. Users of this filter should only make use of the center valid central region.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a floating point type. This filter works for any dimensional images.

Input/Output Restrictions: TInputImage and TOutputImage must have the same dimension. TOutputImage's pixel type must be a real number type.

**See** DenseFiniteDifferenceImageFilter

**See** CurvatureFlowFunction

**See** MinMaxCurvatureFlowImageFilter

**See** BinaryMinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image Using Curvature Flow

- Smooth RGB Image Using Curvature Flow

Subclassed by itk::MinMaxCurvatureFlowImageFilter< TInputImage, TOutputImage >

See itk::CurvatureFlowImageFilter for additional documentation.

### Smooth RGB Image Using Min Max Curvature Flow

### Synopsis

Smooth an RGB image using min/max curvature flow.

### Results

Fig. 126: Input image.

### Code

### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"
#include "itkNthElementImageAdaptor.h"
#include "itkMinMaxCurvatureFlowImageFilter.h"
#include "itkComposeImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkRGBPixel.h"
```

(continues on next page)


Fig. 127: Output In VTK Window

(continued from previous page)

```
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " InputImageFile [iterations]" << std::endl;
    return EXIT_FAILURE;
  }

  int iterations = 5;
  if (argc > 2)
  {
    std::stringstream ss(argv[2]);
    ss >> iterations;
  }
  std::string inputFilename = argv[1];

  // Setup types
  using ComponentType = float;
  using PixelType = itk::RGBPixel<ComponentType>;
  using RGBImageType = itk::Image<PixelType, 2>;
  using ImageType = itk::Image<ComponentType, 2>;
  using ReaderType = itk::ImageFileReader<RGBImageType>;
  using ImageAdaptorType = itk::NthElementImageAdaptor<RGBImageType, unsigned char>;
  using ComposeType = itk::ComposeImageFilter<ImageType, RGBImageType>;
  using CurvatureFlowType = itk::MinMaxCurvatureFlowImageFilter<ImageAdaptorType,␣
↪ImageType>;

  // Create and setup a reader
```

(continues on next page)

(continued from previous page)

```
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);
  reader->Update();

  // Run the filter for each component
  ImageAdaptorType::Pointer rAdaptor = ImageAdaptorType::New();
  rAdaptor->SelectNthElement(0);
  rAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer rCurvatureFilter = CurvatureFlowType::New();
  rCurvatureFilter->SetInput(rAdaptor);
  rCurvatureFilter->SetNumberOfIterations(iterations);
  rCurvatureFilter->Update();

  ImageAdaptorType::Pointer gAdaptor = ImageAdaptorType::New();
  gAdaptor->SelectNthElement(1);
  gAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer gCurvatureFilter = CurvatureFlowType::New();
  gCurvatureFilter->SetInput(gAdaptor);
  gCurvatureFilter->SetNumberOfIterations(iterations);
  gCurvatureFilter->Update();

  ImageAdaptorType::Pointer bAdaptor = ImageAdaptorType::New();
  bAdaptor->SelectNthElement(2);
  bAdaptor->SetImage(reader->GetOutput());

  CurvatureFlowType::Pointer bCurvatureFilter = CurvatureFlowType::New();
  bCurvatureFilter->SetInput(bAdaptor);
  bCurvatureFilter->SetNumberOfIterations(iterations);
  bCurvatureFilter->Update();

  // compose an RGB image from the three filtered images

  ComposeType::Pointer compose = ComposeType::New();
  compose->SetInput1(rCurvatureFilter->GetOutput());
  compose->SetInput2(gCurvatureFilter->GetOutput());
  compose->SetInput3(bCurvatureFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFilename));

  std::stringstream desc;
  desc << "MinMaxCurvatureFlow iterations = " << iterations;
  viewer.AddRGBImage(compose->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MinMaxCurvatureFlowImageFilter** : **public** itk::*CurvatureFlowImageFilter<TInputImage*, *TOutputImage>*
Denoise an image using min/max curvature flow.

MinMaxCurvatureFlowImageFilter implements a curvature driven image denoising algorithm. Iso-brightness contours in the grayscale input image are viewed as a level set. The level set is then evolved using a curvature-based speed function:

$$I_t = F_{\text{minmax}}|\nabla I|$$

where $F_{\text{minmax}} = \max(\kappa, 0)$ if $\text{Avg}_{\text{stencil}}(x)$ is less than or equal to $T_{threshold}$ and $\min(\kappa, 0)$, otherwise. $\kappa$ is the mean curvature of the iso-brightness contour at point $x$.

In min/max curvature flow, movement is turned on or off depending on the scale of the noise one wants to remove. Switching depends on the average image value of a region of radius $R$ around each point. The choice of $R$, the stencil radius, governs the scale of the noise to be removed.

The threshold value $T_{threshold}$ is the average intensity obtained in the direction perpendicular to the gradient at point $x$ at the extrema of the local neighborhood.

This filter make use of the multi-threaded finite difference solver hierarchy. Updates are computed using a MinMaxCurvatureFlowFunction object. A zero flux Neumann boundary condition is used when computing derivatives near the data boundary.

Reference: "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Chapter 16, Second edition, 1999.

**Warning** This filter assumes that the input and output types have the same dimensions. This filter also requires that the output image pixels are of a real type. This filter works for any dimensional images, however for dimensions greater than 3D, an expensive brute-force search is used to compute the local threshold.

**See** MinMaxCurvatureFlowFunction

**See** CurvatureFlowImageFilter

**See** BinaryMinMaxCurvatureFlowImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Smooth Image Using Min Max Curvature Flow
- SmoothRGBImageUsingMinMaxCurvatureFlow

Subclassed by itk::BinaryMinMaxCurvatureFlowImageFilter< TInputImage, TOutputImage >

See itk::MinMaxCurvatureFlowImageFilter for additional documentation.

### 3.4.7 DistanceMap

#### Approximate Distance Map of Binary Image

#### Synopsis

Compute the distance map from objects in a binary image.

#### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

#### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkApproximateSignedDistanceMapImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int argc, char * argv[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  if (argc < 2)
  {
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<UnsignedCharImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }
```

(continues on next page)

(continued from previous page)

```
  using ApproximateSignedDistanceMapImageFilterType =
    itk::ApproximateSignedDistanceMapImageFilter<UnsignedCharImageType,␣
↪FloatImageType>;
  ApproximateSignedDistanceMapImageFilterType::Pointer␣
↪approximateSignedDistanceMapImageFilter =
    ApproximateSignedDistanceMapImageFilterType::New();
  approximateSignedDistanceMapImageFilter->SetInput(image);
  approximateSignedDistanceMapImageFilter->SetInsideValue(255);
  approximateSignedDistanceMapImageFilter->SetOutsideValue(0);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "Approximate Signed Distance";
  viewer.AddImage(approximateSignedDistanceMapImageFilter->GetOutput(), true, desc.
↪str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a line of white pixels
  for (unsigned int i = 40; i < 60; ++i)
  {
    itk::Index<2> pixel;
    pixel.Fill(i);
    image->SetPixel(pixel, 255);
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ApproximateSignedDistanceMapImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Create a map of the approximate signed distance from the boundaries of a binary image.

The ApproximateSignedDistanceMapImageFilter takes as input a binary image and produces a signed distance map. Each pixel value in the output contains the approximate distance from that pixel to the nearest "object" in the binary image. This filter differs from the DanielssonDistanceMapImageFilter in that it calculates the distance to the "object edge" for pixels within the object.

Negative values in the output indicate that the pixel at that position is within an object in the input image. The absolute value of a negative pixel represents the approximate distance to the nearest object boundary pixel.

WARNING: This filter requires that the output type be floating-point. Otherwise internal calculations will not be performed to the appropriate precision, resulting in completely incorrect (read: zero-valued) output.

The distances computed by this filter are Chamfer distances, which are only an approximation to Euclidian distances, and are not as exact approximations as those calculated by the DanielssonDistanceMapImageFilter. On the other hand, this filter is faster.

This filter requires that an "inside value" and "outside value" be set as parameters. The "inside value" is the intensity value of the binary image which corresponds to objects, and the "outside value" is the intensity of the background. (A typical binary image often represents objects as black (0) and background as white (usually 255), or vice-versa.) Note that this filter is slightly faster if the inside value is less than the outside value. Otherwise an extra iteration through the image is required.

This filter uses the FastChamferDistanceImageFilter and the IsoContourDistanceImageFilter internally to perform the distance calculations.

**See** DanielssonDistanceMapImageFilter

**See** SignedDanielssonDistanceMapImageFilter

**See** SignedMaurerDistanceMapImageFilter

**See** FastChamferDistanceImageFilter

**See** IsoContourDistanceImageFilter

**Author** Zach Pincus

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Approximate Distance Map Of Binary Image

See itk::ApproximateSignedDistanceMapImageFilter for additional documentation.

### Maurer Distance Map of Binary Image

### Synopsis

Compute a distance map from objects in a binary image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkSignedMaurerDistanceMapImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int argc, char * argv[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  if (argc < 2)
  {
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<UnsignedCharImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  using SignedMaurerDistanceMapImageFilterType =
```

(continues on next page)

(continued from previous page)

```
    itk::SignedMaurerDistanceMapImageFilter<UnsignedCharImageType, FloatImageType>;
  SignedMaurerDistanceMapImageFilterType::Pointer distanceMapImageFilter =
    SignedMaurerDistanceMapImageFilterType::New();
  distanceMapImageFilter->SetInput(image);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "Signed Maurer Distance";
  viewer.AddImage(distanceMapImageFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a line of white pixels
  for (unsigned int i = 40; i < 60; ++i)
  {
    itk::Index<2> pixel;
    pixel.Fill(i);
    image->SetPixel(pixel, 255);
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SignedMaurerDistanceMapImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
This filter calculates the Euclidean distance transform of a binary image in linear time for arbitrary dimensions.

The inside is considered as having negative distances. Outside is treated as having positive distances. To change the convention, use the InsideIsPositive(bool) function.

**Inputs and Outputs** This is an image-to-image filter. The dimensionality is arbitrary. The only dimensionality constraint is that the input and output images be of the same dimensions and size. To maintain integer arithmetic within the filter, the default output is the signed squared distance. This implies that the input image should be of type "unsigned int" or "int" whereas the output image is of type "int". Obviously, if the user wishes to utilize the image spacing or to have a filter with the Euclidean distance (as opposed to the squared distance), output image types of float or double should be used.

Reference: C. R. Maurer, Jr., R. Qi, and V. Raghavan, "A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions", IEEE - Transactions on Pattern Analysis and Machine Intelligence, 25(2): 265-270, 2003.

**Parameters** Set/GetBackgroundValue specifies the background of the value of the input binary image. Normally this is zero and, as such, zero is the default value. Other than that, the usage is completely analogous to the itk::DanielssonDistanceImageFilter class except it does not return the Voronoi map.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Maurer Distance Map Of Binary Image

See itk::SignedMaurerDistanceMapImageFilter for additional documentation.

### Mean Distance Between All Points on Two Curves

### Synopsis

Compute the mean distance between all points of two curves.

### Results

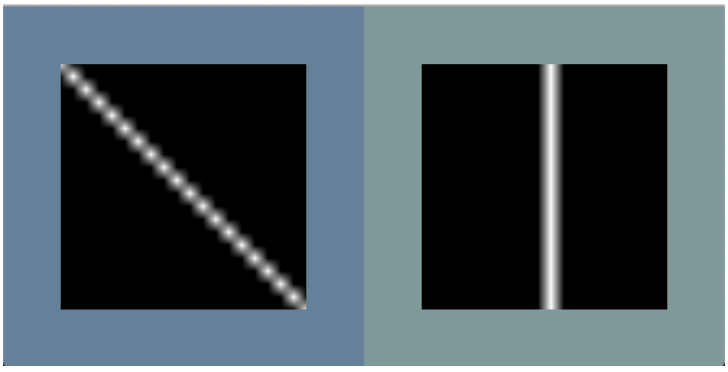


Fig. 128: Output Image Mean distance: 5

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkContourMeanDistanceImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

template <typename TImage>
static void
CreateImage1(TImage * const);

template <typename TImage>
static void
CreateImage2(TImage * const);

int
main(int, char *[])
{
  using ImageType = itk::Image<unsigned char, 2>;

  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1.GetPointer());

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2.GetPointer());

  using ContourMeanDistanceImageFilterType = itk::ContourMeanDistanceImageFilter
↪<ImageType, ImageType>;

  ContourMeanDistanceImageFilterType::Pointer contourMeanDistanceImageFilter =
    ContourMeanDistanceImageFilterType::New();
  contourMeanDistanceImageFilter->SetInput1(image1);
  contourMeanDistanceImageFilter->SetInput2(image2);
  contourMeanDistanceImageFilter->Update();

  std::cout << "Mean distance: " << contourMeanDistanceImageFilter->GetMeanDistance()
↪<< std::endl;

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image1.GetPointer());
  viewer.AddImage(image2.GetPointer());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

template <typename TImage>
void
CreateImage1(TImage * const image)
{
  // Create an image bigger than the input image and that has dimensions which are␣
↪powers of two
```

(continues on next page)

(continued from previous page)

```
  typename TImage::IndexType start = { { 0, 0 } };

  typename TImage::SizeType size = { { 20, 20 } };

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a diagonal white line through the image
  for (typename TImage::IndexValueType i = 0; i < 20; i++)
  {
    for (typename TImage::IndexValueType j = 0; j < 20; j++)
    {
      if (i == j)
      {
        itk::Index<2> pixel = { { i, j } };
        image->SetPixel(pixel, 255);
      }
    }
  }
}

template <typename TImage>
void
CreateImage2(TImage * const image)
{
  typename TImage::IndexType start = { { 0, 0 } };

  typename TImage::SizeType size = { { 20, 20 } };

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a vertical line of white pixels down the center of the image
  for (typename TImage::IndexValueType i = 0; i < 20; i++)
  {
    for (typename TImage::IndexValueType j = 0; j < 20; j++)
    {
      if (i == 10)
      {
        itk::Index<2> pixel = { { i, j } };
        image->SetPixel(pixel, 255);
      }
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**>
**class ContourMeanDistanceImageFilter** : **public** itk::*ImageToImageFilter<TInputImage1, TInputImage1>*
Computes the Mean distance between the boundaries of non-zero regions of two images.

ContourMeanDistanceImageFilter computes the distance between the set non-zero pixels of two images using the following formula:

$$H(A, B) = \max(h(A, B), h(B, A))$$

where

$$h(A, B) = \text{mean}_{a \in A} \min_{b \in B} \|a - b\|$$

is the dir ected Mean distance and $A$ and $B$ are respectively the set of non-zero pixels in the first and second input images.

In particular, this filter uses the ContourDirectedMeanImageFilter inside to compute the two directed distances and then select the largest of the two.

The Mean distance measures the degree of mismatch between two sets and behaves like a metric over the set of all closed bounded sets - with properties of identity, symmetry and triangle inequality.

This filter requires the largest possible region of the first image and the same corresponding region in the second image. It behaves as filter with two input and one output. Thus it can be inserted in a pipeline with other filters. The filter passes the first input through unmodified.

This filter is templated over the two input image type. It assume both image have the same number of dimensions.

**See** ContourDirectedMeanDistanceImageFilter

**Author** Teo Popa, ISIS Center, Georgetown University

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mean Distance Between All Points On Two Curves

See itk::ContourMeanDistanceImageFilter for additional documentation.

### Signed Distance Map of Binary Image

### Synopsis

Compute a distance map from objects in a binary image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkSignedDanielssonDistanceMapImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int argc, char * argv[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  if (argc < 2)
  {
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<UnsignedCharImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  using SignedDanielssonDistanceMapImageFilterType =
    itk::SignedDanielssonDistanceMapImageFilter<UnsignedCharImageType, FloatImageType>
↪;
  SignedDanielssonDistanceMapImageFilterType::Pointer distanceMapImageFilter =
    SignedDanielssonDistanceMapImageFilterType::New();
  distanceMapImageFilter->SetInput(image);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
```

(continues on next page)

(continued from previous page)

```
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "Signed Danielsson Distance";
  viewer.AddImage(distanceMapImageFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a line of white pixels
  for (unsigned int i = 40; i < 60; ++i)
  {
    itk::Index<2> pixel;
    pixel.Fill(i);
    image->SetPixel(pixel, 255);
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TVoronoiImage** = *TInputImage*>
**class SignedDanielssonDistanceMapImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

This filter computes the signed distance map of the input image as an approximation with pixel accuracy to the Euclidean distance.

This class is parameterized over the type of the input image and the type of the output image.

For purposes of evaluating the signed distance map, the input is assumed to be binary composed of pixels with value 0 and non-zero.

The inside is considered as having negative distances. Outside is treated as having positive distances. To change the convention, use the InsideIsPositive(bool) function.

As a convention, the distance is evaluated from the boundary of the ON pixels.

The filter returns

- A signed distance map with the approximation to the euclidean distance.
- A voronoi partition. (See itkDanielssonDistanceMapImageFilter)
- A vector map containing the component of the vector relating the current pixel with the closest point of the closest object to this pixel. Given that the components of the distance are computed in "pixels", the vector is represented by an itk::Offset. That is, physical coordinates are not used. (See itkDanielssonDistanceMapImageFilter)

This filter internally uses the DanielssonDistanceMap filter. This filter is N-dimensional.

**See** itkDanielssonDistanceMapImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Signed Distance Map Of Binary Image

See itk::SignedDanielssonDistanceMapImageFilter for additional documentation.

### 3.4.8 FastMarching

#### Compute Geodesic Distance on Mesh

#### Synopsis

Compute the geodesic distance from a provided seed vertex on a mesh.

#### Results

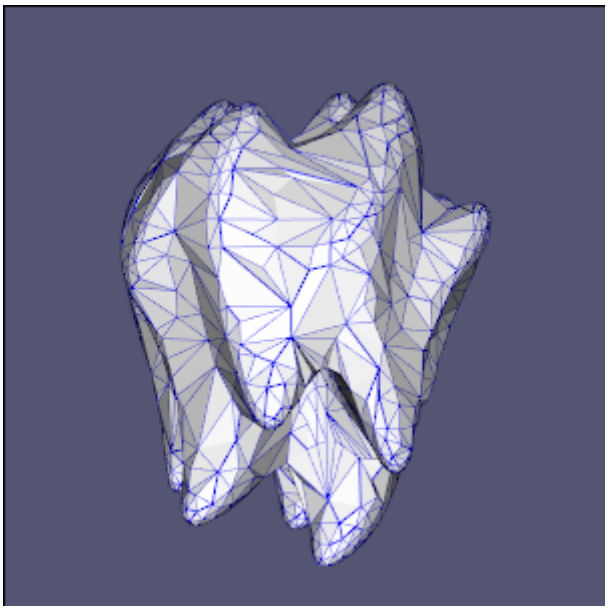


Fig. 129: Input mesh

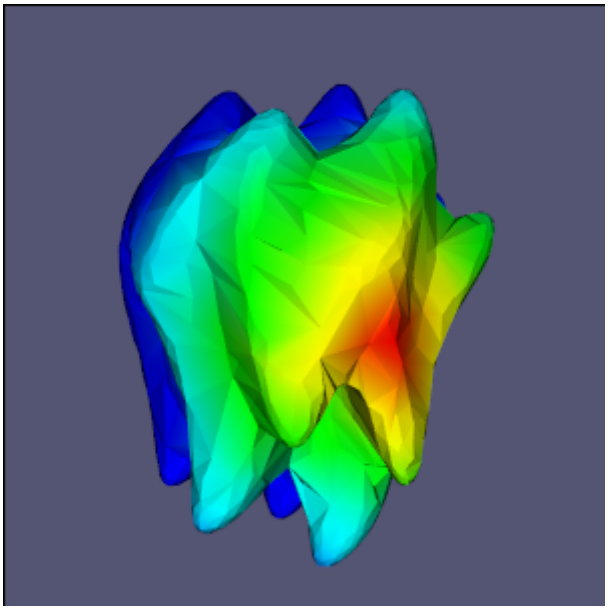


Fig. 130: Output mesh

## Code

### C++

```
#include "itkFastMarchingQuadEdgeMeshFilterBase.h"
#include "itkQuadEdgeMeshExtendedTraits.h"
#include "itkRegularSphereMeshSource.h"
#include "itkFastMarchingThresholdStoppingCriterion.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << argv[0] << std::endl;
    std::cerr << " <input filename> <output filename>" << std::endl;
    return EXIT_FAILURE;
  }
  using PixelType = float;
  using CoordType = double;

  constexpr unsigned int Dimension = 3;

  using Traits = itk::QuadEdgeMeshExtendedTraits<PixelType, // type of data for␣
↪vertices
                                                 Dimension, // geometrical dimension␣
↪of space
                                                 2,         // Mac topological␣
↪dimension of a cell
                                                 CoordType, // type for point␣
↪coordinate
```

(continues on next page)

(continued from previous page)

```
                                                    CoordType, // type for interpolation␣
↪weight
                                                    PixelType, // type of data for cell
                                                    bool,      // type of data for␣
↪primal edges
                                                    bool       // type of data for dual␣
↪edges
                                                    >;

  using MeshType = itk::QuadEdgeMesh<PixelType, Dimension, Traits>;

  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using FastMarchingType = itk::FastMarchingQuadEdgeMeshFilterBase<MeshType, MeshType>
↪;

  MeshType::Pointer mesh = reader->GetOutput();

  MeshType::PointsContainerConstPointer points = mesh->GetPoints();

  MeshType::PointsContainerConstIterator pIt = points->Begin();
  MeshType::PointsContainerConstIterator pEnd = points->End();

  while (pIt != pEnd)
  {
    mesh->SetPointData(pIt->Index(), 1.);
    ++pIt;
  }

  using NodePairType = FastMarchingType::NodePairType;
  using NodePairContainerType = FastMarchingType::NodePairContainerType;

  NodePairContainerType::Pointer trial = NodePairContainerType::New();

  NodePairType nodePair(0, 0.);
  trial->push_back(nodePair);

  using CriterionType = itk::FastMarchingThresholdStoppingCriterion<MeshType,␣
↪MeshType>;
  CriterionType::Pointer criterion = CriterionType::New();
  criterion->SetThreshold(100.);

  FastMarchingType::Pointer fmmFilter = FastMarchingType::New();
  fmmFilter->SetInput(mesh);
  fmmFilter->SetTrialPoints(trial);
  fmmFilter->SetStoppingCriterion(criterion);

  using WriterType = itk::MeshFileWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(fmmFilter->GetOutput());
  writer->SetFileName(argv[2]);

  try
  {
    writer->Update();
```

(continues on next page)

(continued from previous page)

```
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInput**, typename **TOutput**>
**class FastMarchingQuadEdgeMeshFilterBase** : **public** itk::FastMarchingBase<*TInput*, *TOutput*>
Fast Marching Method on QuadEdgeMesh.

The speed function is specified by the input mesh. Data associated to each point is considered as the speed function. The speed function is set using the method SetInput().

If the speed function is constant and of value one, fast marching results is an approximate geodesic function from the initial alive points.

Implementation of this class is based on "Fast Marching Methods on Triangulated Domains", Kimmel, R., and Sethian, J.A., Proc. Nat. Acad. Sci., 95, pp. 8341-8435, 1998.

See itk::FastMarchingQuadEdgeMeshFilterBase for additional documentation.

### Create Distance Map From Seeds

### Synopsis

Create a distance map from given seeds

### Results

### Code

### C++

```
#include "itkFastMarchingImageToNodePairContainerAdaptor.h"
#include "itkFastMarchingImageFilterBase.h"
#include "itkFastMarchingThresholdStoppingCriterion.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << " <OutputFileName>" << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  // create a fastmarching object
  using PixelType = float;
  constexpr unsigned int Dimension = 2;

  using FloatImageType = itk::Image<PixelType, Dimension>;

  using CriterionType = itk::FastMarchingThresholdStoppingCriterion<FloatImageType,␣
↪FloatImageType>;

  using FastMarchingType = itk::FastMarchingImageFilterBase<FloatImageType,␣
↪FloatImageType>;

  CriterionType::Pointer criterion = CriterionType::New();
  criterion->SetThreshold(100.);

  FastMarchingType::Pointer marcher = FastMarchingType::New();
  marcher->SetStoppingCriterion(criterion);

  // specify the size of the output image
  FloatImageType::SizeType size = { { 64, 64 } };
  marcher->SetOutputSize(size);

  // setup a speed image of ones
  FloatImageType::Pointer speedImage = FloatImageType::New();

  FloatImageType::RegionType region;
  region.SetSize(size);

  speedImage->SetLargestPossibleRegion(region);
  speedImage->SetBufferedRegion(region);
  speedImage->Allocate();
  speedImage->FillBuffer(1.0);

  // setup a 'alive image'
  FloatImageType::Pointer AliveImage = FloatImageType::New();
  AliveImage->SetLargestPossibleRegion(region);
  AliveImage->SetBufferedRegion(region);
  AliveImage->Allocate();
  AliveImage->FillBuffer(0.0);

  FloatImageType::OffsetType offset0 = { { 28, 35 } };

  FloatImageType::IndexType index;
  index.Fill(0);
  index += offset0;

  AliveImage->SetPixel(index, 1.0);

  // setup a 'trial image'
  FloatImageType::Pointer TrialImage = FloatImageType::New();
  TrialImage->SetLargestPossibleRegion(region);
  TrialImage->SetBufferedRegion(region);
  TrialImage->Allocate();
```

(continues on next page)

(continued from previous page)

```
  TrialImage->FillBuffer(0.0);

  index[0] += 1;
  TrialImage->SetPixel(index, 1.0);

  index[0] -= 1;
  index[1] += 1;
  TrialImage->SetPixel(index, 1.0);

  index[0] -= 1;
  index[1] -= 1;
  TrialImage->SetPixel(index, 1.0);

  index[0] += 1;
  index[1] -= 1;
  TrialImage->SetPixel(index, 1.0);

  marcher->SetInput(speedImage);

  using AdaptorType = itk::FastMarchingImageToNodePairContainerAdaptor<FloatImageType,
↪ FloatImageType, FloatImageType>;

  AdaptorType::Pointer adaptor = AdaptorType::New();

  adaptor->SetAliveImage(AliveImage.GetPointer());
  adaptor->SetAliveValue(0.0);

  adaptor->SetTrialImage(TrialImage.GetPointer());
  adaptor->SetTrialValue(1.0);
  adaptor->Update();

  marcher->SetAlivePoints(adaptor->GetAlivePoints());
  marcher->SetTrialPoints(adaptor->GetTrialPoints());

  using WriterType = itk::ImageFileWriter<FloatImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(marcher->GetOutput());
  writer->SetFileName(argv[1]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInput**, typename **TOutput**>
**class FastMarchingImageFilterBase** : **public** itk::FastMarchingBase<*TInput*, *TOutput*>

Apply the Fast Marching method to solve an Eikonal equation on an image.

The speed function can be specified as a speed image or a speed constant. The speed image is set using the method SetInput(). If the speed image is nullptr, a constant speed function is used and is specified using method the SetSpeedConstant().

If the speed function is constant and of value one, fast marching results is an approximate distance function from the initial alive points.

There are two ways to specify the output image information (LargestPossibleRegion, Spacing, Origin):

- it is copied directly from the input speed image
- it is specified by the user. Default values are used if the user does not specify all the information.

The output information is computed as follows.

If the speed image is nullptr or if the OverrideOutputInformation is set to true, the output information is set from user specified parameters. These parameters can be specified using methods

- FastMarchingImageFilterBase::SetOutputRegion(),
- FastMarchingImageFilterBase::SetOutputSpacing(),
- FastMarchingImageFilterBase::SetOutputDirection(),
- FastMarchingImageFilterBase::SetOutputOrigin().

Else the output information is copied from the input speed image.

Implementation of this class is based on Chapter 8 of "Level Set Methods and Fast Marching Methods", J.A. Sethian, Cambridge Press, Second edition, 1999.

For an alternative implementation, see itk::FastMarchingImageFilter.

**See** FastMarchingImageFilter

**See** ImageFastMarchingTraits

**See** ImageFastMarchingTraits2

**Template Parameters**

- `TTraits`: traits

Subclassed by itk::FastMarchingExtensionImageFilterBase< TInput, TOutput, TAuxValue, VAuxDimension >, itk::FastMarchingUpwindGradientImageFilterBase< TInput, TOutput >

See itk::FastMarchingImageFilterBase for additional documentation.

### 3.4.9 FFT

#### Compute Forward FFT

#### Synopsis

Compute forward FFT of an image.

#### Results

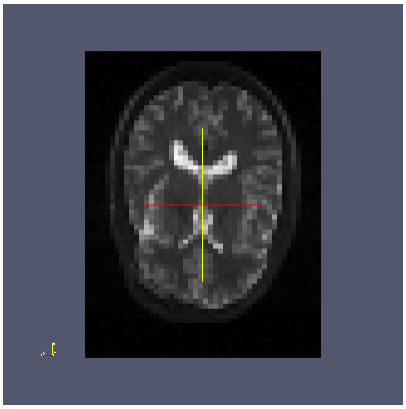


Fig. 131: Input image

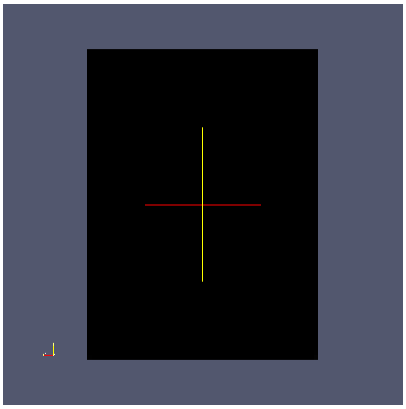


Fig. 132: Output Real image

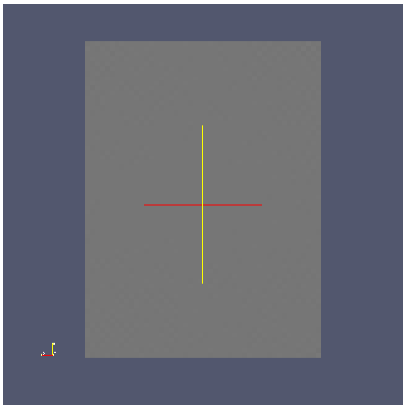


Fig. 133: Output Imaginary image

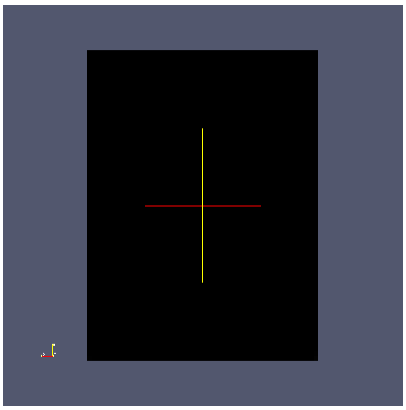


Fig. 134: Output Modulus image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkWrapPadImageFilter.h"
#include "itkForwardFFTImageFilter.h"
#include "itkComplexToRealImageFilter.h"
#include "itkComplexToImaginaryImageFilter.h"
#include "itkComplexToModulusImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <Real filename> <Imaginary filename> <Modulus␣
↪filename>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * realFileName = argv[2];
  const char * imaginaryFileName = argv[3];
  const char * modulusFileName = argv[4];

  constexpr unsigned int Dimension = 3;

  using FloatPixelType = float;
  using FloatImageType = itk::Image<FloatPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<FloatImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using UnsignedCharPixelType = unsigned char;
  using UnsignedCharImageType = itk::Image<UnsignedCharPixelType, Dimension>;

  // Some FFT filter implementations, like VNL's, need the image size to be a
  // multiple of small prime numbers.
  using PadFilterType = itk::WrapPadImageFilter<FloatImageType, FloatImageType>;
  PadFilterType::Pointer padFilter = PadFilterType::New();
  padFilter->SetInput(reader->GetOutput());
  PadFilterType::SizeType padding;
  // Input size is [48, 62, 42].  Pad to [48, 64, 48].
  padding[0] = 0;
  padding[1] = 2;
  padding[2] = 6;
  padFilter->SetPadUpperBound(padding);

  using FFTType = itk::ForwardFFTImageFilter<FloatImageType>;
```

(continues on next page)

(continued from previous page)

```
  FFTType::Pointer fftFilter = FFTType::New();
  fftFilter->SetInput(padFilter->GetOutput());

  using FFTOutputImageType = FFTType::OutputImageType;

  // Extract the real part
  using RealFilterType = itk::ComplexToRealImageFilter<FFTOutputImageType,␣
↪FloatImageType>;
  RealFilterType::Pointer realFilter = RealFilterType::New();
  realFilter->SetInput(fftFilter->GetOutput());

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer realRescaleFilter = RescaleFilterType::New();
  realRescaleFilter->SetInput(realFilter->GetOutput());
  realRescaleFilter->SetOutputMinimum(itk::NumericTraits<UnsignedCharPixelType>::
↪min());
  realRescaleFilter->SetOutputMaximum(itk::NumericTraits<UnsignedCharPixelType>::
↪max());

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer realWriter = WriterType::New();
  realWriter->SetFileName(realFileName);
  realWriter->SetInput(realRescaleFilter->GetOutput());
  try
  {
    realWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  // Extract the imaginary part
  using ImaginaryFilterType = itk::ComplexToImaginaryImageFilter<FFTOutputImageType,␣
↪FloatImageType>;
  ImaginaryFilterType::Pointer imaginaryFilter = ImaginaryFilterType::New();
  imaginaryFilter->SetInput(fftFilter->GetOutput());

  RescaleFilterType::Pointer imaginaryRescaleFilter = RescaleFilterType::New();
  imaginaryRescaleFilter->SetInput(imaginaryFilter->GetOutput());
  imaginaryRescaleFilter->SetOutputMinimum(itk::NumericTraits<UnsignedCharPixelType>::
↪min());
  imaginaryRescaleFilter->SetOutputMaximum(itk::NumericTraits<UnsignedCharPixelType>::
↪max());

  WriterType::Pointer complexWriter = WriterType::New();
  complexWriter->SetFileName(imaginaryFileName);
  complexWriter->SetInput(imaginaryRescaleFilter->GetOutput());
  try
  {
    complexWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  // Compute the magnitude
  using ModulusFilterType = itk::ComplexToModulusImageFilter<FFTOutputImageType,␣
↪FloatImageType>;
  ModulusFilterType::Pointer modulusFilter = ModulusFilterType::New();
  modulusFilter->SetInput(fftFilter->GetOutput());

  RescaleFilterType::Pointer magnitudeRescaleFilter = RescaleFilterType::New();
  magnitudeRescaleFilter->SetInput(modulusFilter->GetOutput());
  magnitudeRescaleFilter->SetOutputMinimum(itk::NumericTraits<UnsignedCharPixelType>::
↪min());
  magnitudeRescaleFilter->SetOutputMaximum(itk::NumericTraits<UnsignedCharPixelType>::
↪max());

  WriterType::Pointer magnitudeWriter = WriterType::New();
  magnitudeWriter->SetFileName(modulusFileName);
  magnitudeWriter->SetInput(magnitudeRescaleFilter->GetOutput());
  try
  {
    magnitudeWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<std::complex<**typename** *TInputImage*::PixelType>, *TInp*
**class ForwardFFTImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for forward Fast Fourier Transform.

This is a base class for the "forward" or "direct" discrete Fourier Transform. This is an abstract base class: the actual implementation is provided by the best child class available on the system when the object is created via the object factory system.

This class transforms a real input image into its full complex Fourier transform. The Fourier transform of a real input image has Hermitian symmetry: $f(\mathbf{x}) = f^*(-\mathbf{x})$. That is, when the result of the transform is split in half along the x-dimension, the values in the second half of the transform are the complex conjugates of values in the first half reflected about the center of the image in each dimension.

This filter works only for real single-component input image types.

The output generated from a ForwardFFTImageFilter is in the dual space or frequency domain. Refer to FrequencyFFTLayoutImageRegionConstIteratorWithIndex for a description of the layout of frequencies generated after a forward FFT. Also see ITKImageFrequency for a set of filters requiring input images in the frequency domain.

**See** InverseFFTImageFilter, FFTComplexToComplexImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Forward FFT

Subclassed by itk::FFTWForwardFFTImageFilter< TInputImage, TOutputImage >, itk::VnlForwardFFTImageFilter< TInputImage, TOutputImage >

See itk::ForwardFFTImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class ComplexToRealImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes pixel-wise the real(x) part of a complex image.

See itk::ComplexToRealImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class ComplexToImaginaryImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes pixel-wise the imaginary part of a complex image.

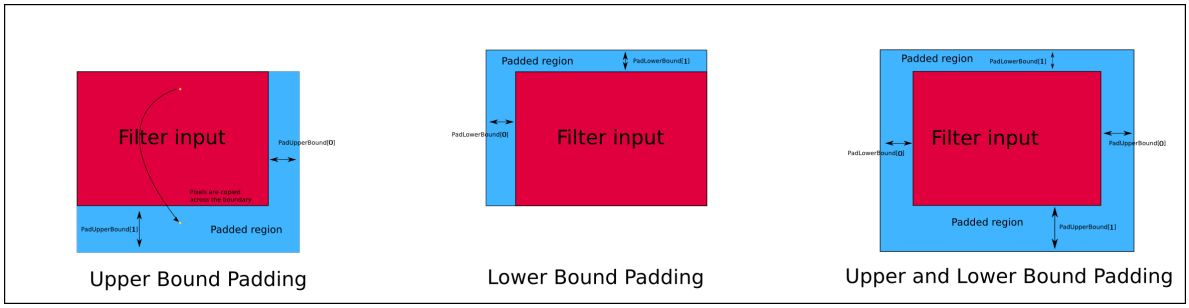
See itk::ComplexToImaginaryImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class ComplexToModulusImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes pixel-wise the Modulus of a complex image.

See itk::ComplexToModulusImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class WrapPadImageFilter** : **public** itk::PadImageFilter<*TInputImage*, *TOutputImage*>
Increase the image size by padding with replicants of the input image value.

WrapPadImageFilter changes the image bounds of an image. Added pixels are filled in with a wrapped replica of the input image. For instance, if the output image needs a pixel that is **two pixels to the left of the LargestPossibleRegion** of the input image, the value assigned will be from the pixel **two pixels inside the right boundary of the LargestPossibleRegion**. The image bounds of the output must be specified.




This filter is implemented as a multithreaded filter. It provides a ThreadedGenerateData() method for its implementation.

**See** MirrorPadImageFilter, ConstantPadImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Pad Image By Wrapping

See itk::WrapPadImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class RescaleIntensityImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::Inte

Applies a linear transformation to the intensity levels of the input Image.

RescaleIntensityImageFilter applies pixel-wise a linear transformation to the intensity values of input image pixels. The linear transformation is defined by the user in terms of the minimum and maximum values that the output image should have.

The following equation gives the mapping of the intensity values

All computations are performed in the precision of the input pixel's RealType. Before assigning the computed value to the output pixel.

$$outputPixel = (inputPixel - inputMin) \cdot \frac{(outputMax - outputMin)}{(inputMax - inputMin)} + outputMin$$

NOTE: In this filter the minimum and maximum values of the input image are computed internally using the MinimumMaximumImageCalculator. Users are not supposed to set those values in this filter. If you need a filter where you can set the minimum and maximum values of the input, please use the IntensityWindowingImage-Filter. If you want a filter that can use a user-defined linear transformation for the intensity, then please use the ShiftScaleImageFilter.

**See** IntensityWindowingImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Rescale An Image

See itk::RescaleIntensityImageFilter for additional documentation.

### Compute Image Spectral Density

### Synopsis

Compute the magnitude of an image's spectral components.

### Results

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkWrapPadImageFilter.h"
#include "itkForwardFFTImageFilter.h"
#include "itkComplexToModulusImageFilter.h"
#include "itkIntensityWindowingImageFilter.h"
#include "itkFFTShiftImageFilter.h"
```

(continues on next page)

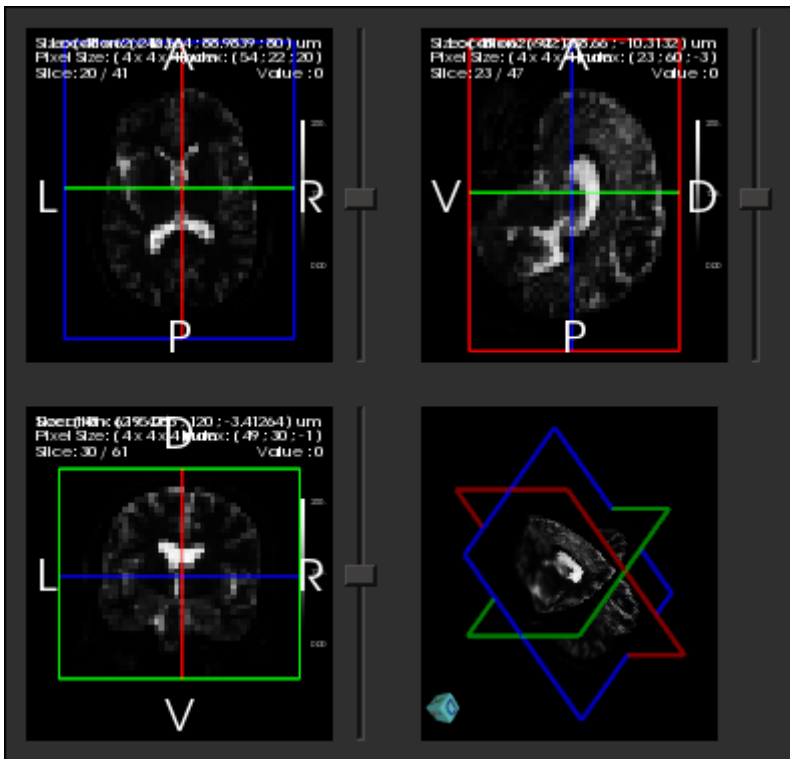



Fig. 135: Input image

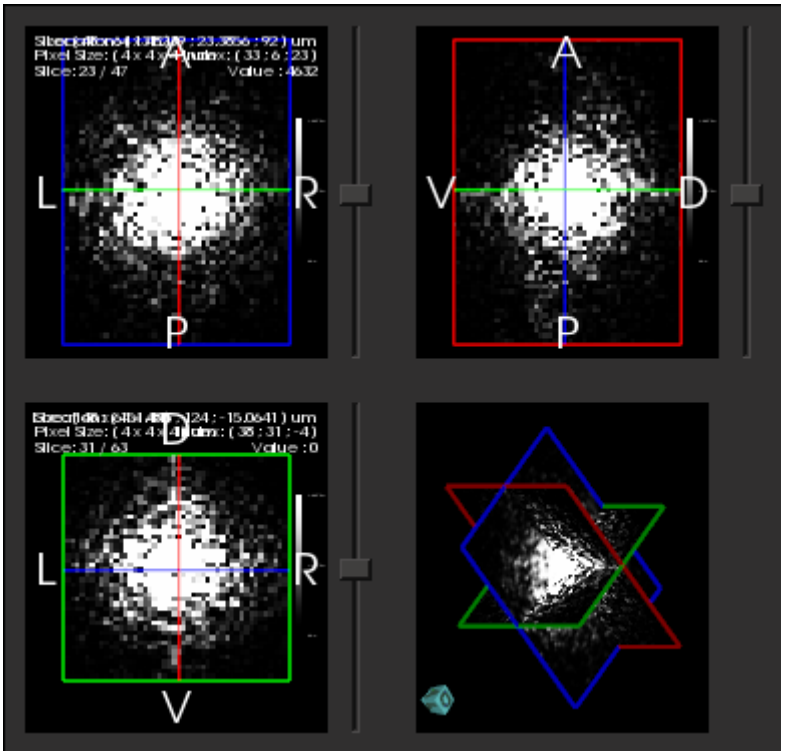



Fig. 136: Output image of spectral density. The DC component is shifted to the center of the image.

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 3;

  using PixelType = float;
  using RealImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<RealImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // Some FFT filter implementations, like VNL's, need the image size to be a
  // multiple of small prime numbers.
  using PadFilterType = itk::WrapPadImageFilter<RealImageType, RealImageType>;
  PadFilterType::Pointer padFilter = PadFilterType::New();
  padFilter->SetInput(reader->GetOutput());
  PadFilterType::SizeType padding;
  // Input size is [48, 62, 42].  Pad to [48, 64, 48].
  padding[0] = 0;
  padding[1] = 2;
  padding[2] = 6;
  padFilter->SetPadUpperBound(padding);

  using ForwardFFTFilterType = itk::ForwardFFTImageFilter<RealImageType>;
  using ComplexImageType = ForwardFFTFilterType::OutputImageType;
  ForwardFFTFilterType::Pointer forwardFFTFilter = ForwardFFTFilterType::New();
  forwardFFTFilter->SetInput(padFilter->GetOutput());

  using ComplexToModulusFilterType = itk::ComplexToModulusImageFilter
↪<ComplexImageType, RealImageType>;
  ComplexToModulusFilterType::Pointer complexToModulusFilter =␣
↪ComplexToModulusFilterType::New();
  complexToModulusFilter->SetInput(forwardFFTFilter->GetOutput());

  // Window and shift the output for visualization.
  using OutputPixelType = unsigned short;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;
  using WindowingFilterType = itk::IntensityWindowingImageFilter<RealImageType,␣
↪OutputImageType>;
  WindowingFilterType::Pointer windowingFilter = WindowingFilterType::New();
  windowingFilter->SetInput(complexToModulusFilter->GetOutput());
  windowingFilter->SetWindowMinimum(0);
  windowingFilter->SetWindowMaximum(20000);
```

(continues on next page)

(continued from previous page)

```
  using FFTShiftFilterType = itk::FFTShiftImageFilter<OutputImageType,␣
↪OutputImageType>;
  FFTShiftFilterType::Pointer fftShiftFilter = FFTShiftFilterType::New();
  fftShiftFilter->SetInput(windowingFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(fftShiftFilter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ComplexToModulusImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes pixel-wise the Modulus of a complex image.

See itk::ComplexToModulusImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class FFTShiftImageFilter** : **public** itk::CyclicShiftImageFilter<*TInputImage*, *TOutputImage*>
Shift the zero-frequency components of a Fourier transform to the center of the image.

The Fourier transform produces an image where the zero frequency components are in the corner of the image, making it difficult to understand. This filter shifts the component to the center of the image.

https://www.insight-journal.org/browse/publication/125

**Note** For images with an odd-sized dimension, applying this filter twice will not produce the same image as the original one without using SetInverse(true) on one (and only one) of the two filters.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ForwardFFTImageFilter, InverseFFTImageFilter

See itk::FFTShiftImageFilter for additional documentation.

### Compute Inverse FFT of Image

### Synopsis

Compute the inverse FFT of an image.

### Results

Fig. 137: ifft.png

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Inverse FFT Of Image.")
parser.add_argument("input_image", nargs="?")
args = parser.parse_args()

dimension = 2
float_image_type = itk.Image[itk.F, dimension]

if not args.input_image:
    corner = itk.Index[dimension]()
    corner.Fill(0)

    size = itk.Size[dimension]()
    size.Fill(200)

    region = itk.ImageRegion[dimension]()
    region.SetIndex(corner)
    region.SetSize(size)

    image = float_image_type.New(Regions=region)
    image.Allocate()

    # Make a square
    image[40:100, 40:100] = 100
```

(continues on next page)

(continued from previous page)

```
else:
    image = itk.imread(args.input_image, pixel_type=itk.F)

# Define some types
unsigned_char_image_type = itk.Image[itk.UC, dimension]

# Compute the FFT
image = itk.forward_fft_image_filter(image)

# Compute the IFFT
image = itk.inverse_fft_image_filter(image)

image = itk.cast_image_filter(image, ttype=(float_image_type, unsigned_char_image_
↪type))

itk.imwrite(image, "ComputeInverseFFTOfImagePython.png")
```

### C++

```
#include "itkImage.h"
#include "itkForwardFFTImageFilter.h"
#include "itkInverseFFTImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileReader.h"
#include "itkCastImageFilter.h"
#include "itkImageFileWriter.h"

using FloatImageType = itk::Image<float, 2>;

static void
CreateImage(FloatImageType * const image);

int
main(int argc, char * argv[])
{
  FloatImageType::Pointer image;
  // Verify input
  if (argc < 2)
  {
    image = FloatImageType::New();
    CreateImage(image);
    // std::cerr << "Required: filename" << std::endl;
    // return EXIT_FAILURE;
  }
  else
  {
    // Read the image
    using ReaderType = itk::ImageFileReader<FloatImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();

    image = reader->GetOutput();
```

(continues on next page)

(continued from previous page)

```
  }

  // Define some types
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;

  // Compute the FFT
  using FFTType = itk::ForwardFFTImageFilter<FloatImageType>;
  FFTType::Pointer fftFilter = FFTType::New();
  fftFilter->SetInput(image);
  fftFilter->Update();

  // Compute the IFFT
  // using IFFTType = itk::InverseFFTImageFilter<FFTType::OutputImageType,␣
↪UnsignedCharImageType>; // This does not work
  // - output type seems to need to be float, but it is just an error, not a concept␣
↪check error...
  using IFFTType = itk::InverseFFTImageFilter<FFTType::OutputImageType,␣
↪FloatImageType>;
  IFFTType::Pointer ifftFilter = IFFTType::New();
  ifftFilter->SetInput(fftFilter->GetOutput());
  ifftFilter->Update();

  using CastFilterType = itk::CastImageFilter<FloatImageType, UnsignedCharImageType>;
  CastFilterType::Pointer castFilter = CastFilterType::New();
  castFilter->SetInput(ifftFilter->GetOutput());
  castFilter->Update();

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("ifft.png");
  writer->SetInput(castFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(FloatImageType * const image)
{
  itk::Index<2> corner = { { 0, 0 } };

  itk::Size<2> size = { { 200, 200 } };

  itk::ImageRegion<2> region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (FloatImageType::IndexValueType r = 40; r < 100; r++)
  {
    for (FloatImageType::IndexValueType c = 40; c < 100; c++)
    {
      FloatImageType::IndexType pixelIndex = { { r, c } };

      image->SetPixel(pixelIndex, 100);
```

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<**typename** *TInputImage*::PixelType::value_type, *TInputIm*
**class InverseFFTImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for inverse Fast Fourier Transform.

This is a base class for the "inverse" or "reverse" Discrete Fourier Transform. This is an abstract base class: the actual implementation is provided by the best child available on the system when the object is created via the object factory system.

This class transforms a full complex image with Hermitian symmetry into its real spatial domain representation. If the input does not have Hermitian symmetry, the imaginary component is discarded.

**See** ForwardFFTImageFilter, InverseFFTImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Inverse FFT Of Image

Subclassed by itk::FFTWInverseFFTImageFilter< TInputImage, TOutputImage >, itk::VnlInverseFFTImageFilter< TInputImage, TOutputImage >

See itk::InverseFFTImageFilter for additional documentation.

### Filter Image in Fourier Domain

### Synopsis

Filter an Image in the Fourier Domain One of the most common image processing operations performed in the Fourier Domain is the masking of the spectrum in order to modulate a range of spatial frequencies from the input image. This operation is typically performed by taking the input image, computing its Fourier transform using a FFT filter, masking the resulting image in the Fourier domain with a mask, and finally taking the result of the masking and computing its inverse Fourier transform.

### Results

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkWrapPadImageFilter.h"
#include "itkForwardFFTImageFilter.h"
#include "itkGaussianImageSource.h"
```

(continues on next page)

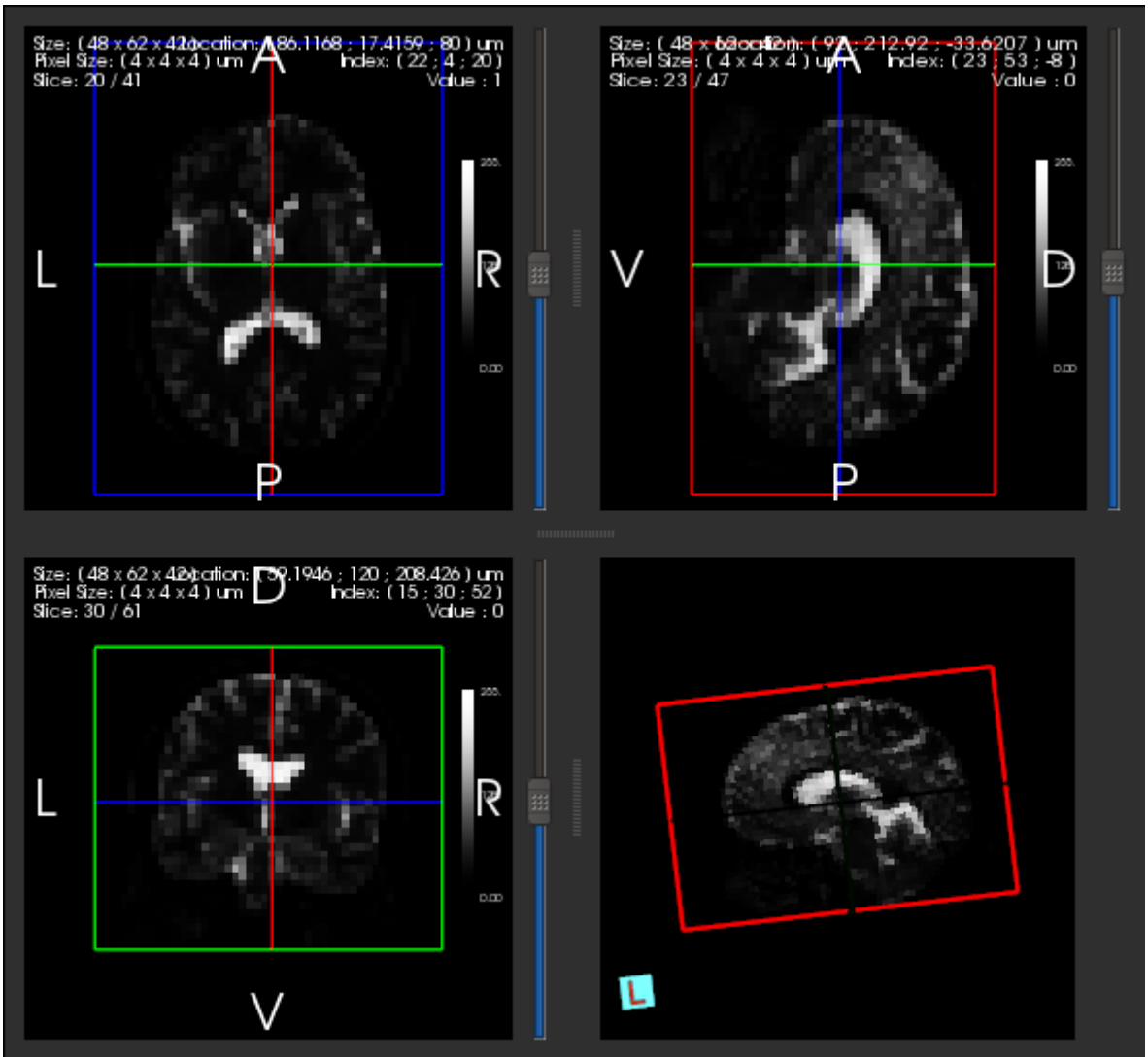



Fig. 138: Input image

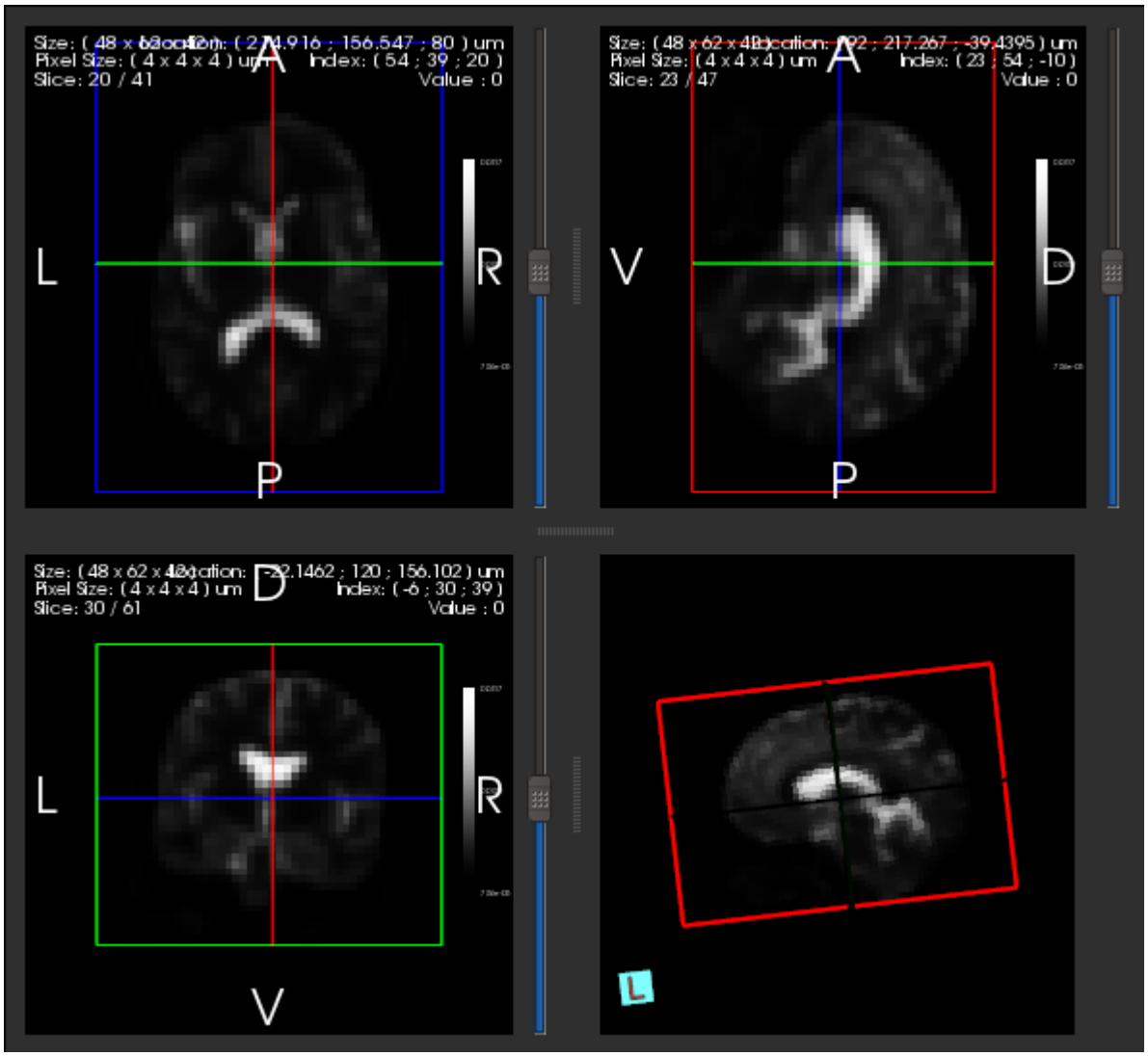



Fig. 139: Output image

(continued from previous page)

```
#include "itkFFTShiftImageFilter.h"
#include "itkMultiplyImageFilter.h"
#include "itkInverseFFTImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " [Sigma]";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  double       sigmaValue = 0.5;
  if (argc > 3)
  {
    sigmaValue = std::stod(argv[3]);
  }

  constexpr unsigned int Dimension = 3;

  using PixelType = float;
  using RealImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<RealImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // Some FFT filter implementations, like VNL's, need the image size to be a
  // multiple of small prime numbers.
  using PadFilterType = itk::WrapPadImageFilter<RealImageType, RealImageType>;
  PadFilterType::Pointer padFilter = PadFilterType::New();
  padFilter->SetInput(reader->GetOutput());
  PadFilterType::SizeType padding;
  // Input size is [48, 62, 42].  Pad to [48, 64, 48].
  padding[0] = 0;
  padding[1] = 2;
  padding[2] = 6;
  padFilter->SetPadUpperBound(padding);

  using ForwardFFTFilterType = itk::ForwardFFTImageFilter<RealImageType>;
  using ComplexImageType = ForwardFFTFilterType::OutputImageType;
  ForwardFFTFilterType::Pointer forwardFFTFilter = ForwardFFTFilterType::New();
  forwardFFTFilter->SetInput(padFilter->GetOutput());

  try
  {
    forwardFFTFilter->UpdateOutputInformation();
  }
  catch (itk::ExceptionObject & error)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  // A Gaussian is used here to create a low-pass filter.
  using GaussianSourceType = itk::GaussianImageSource<RealImageType>;
  GaussianSourceType::Pointer gaussianSource = GaussianSourceType::New();
  gaussianSource->SetNormalized(true);
  ComplexImageType::ConstPointer        transformedInput = forwardFFTFilter->
↪GetOutput();
  const ComplexImageType::RegionType    inputRegion(transformedInput->
↪GetLargestPossibleRegion());
  const ComplexImageType::SizeType      inputSize = inputRegion.GetSize();
  const ComplexImageType::SpacingType   inputSpacing = transformedInput->GetSpacing();
  const ComplexImageType::PointType     inputOrigin = transformedInput->GetOrigin();
  const ComplexImageType::DirectionType inputDirection = transformedInput->
↪GetDirection();
  gaussianSource->SetSize(inputSize);
  gaussianSource->SetSpacing(inputSpacing);
  gaussianSource->SetOrigin(inputOrigin);
  gaussianSource->SetDirection(inputDirection);
  GaussianSourceType::ArrayType sigma;
  GaussianSourceType::PointType mean;
  sigma.Fill(sigmaValue);
  for (unsigned int ii = 0; ii < Dimension; ++ii)
  {
    const double halfLength = inputSize[ii] * inputSpacing[ii] / 2.0;
    sigma[ii] *= halfLength;
    mean[ii] = inputOrigin[ii] + halfLength;
  }
  mean = inputDirection * mean;
  gaussianSource->SetSigma(sigma);
  gaussianSource->SetMean(mean);

  using FFTShiftFilterType = itk::FFTShiftImageFilter<RealImageType, RealImageType>;
  FFTShiftFilterType::Pointer fftShiftFilter = FFTShiftFilterType::New();
  fftShiftFilter->SetInput(gaussianSource->GetOutput());

  using MultiplyFilterType = itk::MultiplyImageFilter<ComplexImageType, RealImageType,
↪ ComplexImageType>;
  MultiplyFilterType::Pointer multiplyFilter = MultiplyFilterType::New();
  multiplyFilter->SetInput1(forwardFFTFilter->GetOutput());
  multiplyFilter->SetInput2(fftShiftFilter->GetOutput());

  using InverseFilterType = itk::InverseFFTImageFilter<ComplexImageType,␣
↪RealImageType>;
  InverseFilterType::Pointer inverseFFTFilter = InverseFilterType::New();
  inverseFFTFilter->SetInput(multiplyFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<RealImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(inverseFFTFilter->GetOutput());
  try
  {
    writer->Update();
  }
```

(continues on next page)

(continued from previous page)

```
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<std::complex<**typename** *TInputImage*::PixelType>, *TInp*
**class ForwardFFTImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for forward Fast Fourier Transform.

This is a base class for the "forward" or "direct" discrete Fourier Transform. This is an abstract base class: the actual implementation is provided by the best child class available on the system when the object is created via the object factory system.

This class transforms a real input image into its full complex Fourier transform. The Fourier transform of a real input image has Hermitian symmetry: $f(\mathbf{x}) = f^*(-\mathbf{x})$. That is, when the result of the transform is split in half along the x-dimension, the values in the second half of the transform are the complex conjugates of values in the first half reflected about the center of the image in each dimension.

This filter works only for real single-component input image types.

The output generated from a ForwardFFTImageFilter is in the dual space or frequency domain. Refer to FrequencyFFTLayoutImageRegionConstIteratorWithIndex for a description of the layout of frequencies generated after a forward FFT. Also see ITKImageFrequency for a set of filters requiring input images in the frequency domain.

**See** InverseFFTImageFilter, FFTComplexToComplexImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Forward FFT

Subclassed by itk::FFTWForwardFFTImageFilter< TInputImage, TOutputImage >, itk::VnlForwardFFTImageFilter< TInputImage, TOutputImage >

See itk::ForwardFFTImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class FFTShiftImageFilter** : **public** itk::CyclicShiftImageFilter<*TInputImage*, *TOutputImage*>

Shift the zero-frequency components of a Fourier transform to the center of the image.

The Fourier transform produces an image where the zero frequency components are in the corner of the image, making it difficult to understand. This filter shifts the component to the center of the image.

https://www.insight-journal.org/browse/publication/125

**Note** For images with an odd-sized dimension, applying this filter twice will not produce the same image as the original one without using SetInverse(true) on one (and only one) of the two filters.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ForwardFFTImageFilter, InverseFFTImageFilter

See itk::FFTShiftImageFilter for additional documentation.

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class MultiplyImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Pixel-wise multiplication of two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiply Two Images
- Multiply Image By Scalar

See itk::MultiplyImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage** = *Image*<**typename** *TInputImage*::PixelType::value_type, *TInputIm*
**class InverseFFTImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for inverse Fast Fourier Transform.

This is a base class for the "inverse" or "reverse" Discrete Fourier Transform. This is an abstract base class: the actual implementation is provided by the best child available on the system when the object is created via the object factory system.

This class transforms a full complex image with Hermitian symmetry into its real spatial domain representation. If the input does not have Hermitian symmetry, the imaginary component is discarded.

**See** ForwardFFTImageFilter, InverseFFTImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Inverse FFT Of Image

Subclassed by itk::FFTWInverseFFTImageFilter< TInputImage, TOutputImage >, itk::VnlInverseFFTImageFilter< TInputImage, TOutputImage >

See itk::InverseFFTImageFilter for additional documentation.

### 3.4.10 ImageCompare

#### Absolute Value Of Difference Between Two Images

#### Synopsis

Compute the absolute value of the difference of corresponding pixels in two images.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkAbsoluteValueDifferenceImageFilter.h"
#include "itkImageRegionIterator.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage1(UnsignedCharImageType::Pointer image);
static void
CreateImage2(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image1 = UnsignedCharImageType::New();
  CreateImage1(image1);

  UnsignedCharImageType::Pointer image2 = UnsignedCharImageType::New();
  CreateImage2(image2);

  using AbsoluteValueDifferenceImageFilterType =
    itk::AbsoluteValueDifferenceImageFilter<UnsignedCharImageType,␣
↪UnsignedCharImageType, FloatImageType>;

  AbsoluteValueDifferenceImageFilterType::Pointer absoluteValueDifferenceFilter =
    AbsoluteValueDifferenceImageFilterType::New();
  absoluteValueDifferenceFilter->SetInput1(image1);
  absoluteValueDifferenceFilter->SetInput2(image2);
  absoluteValueDifferenceFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(UnsignedCharImageType::Pointer image)
{
  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(10);

  UnsignedCharImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();
```

(continues on next page)

(continued from previous page)

```
  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(255);

    ++imageIterator;
  }
}


void
CreateImage2(UnsignedCharImageType::Pointer image)
{
  // Create an image with 2 connected components
  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(10);

  UnsignedCharImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(100);

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**, typename **TOutputImage**>
**class AbsoluteValueDifferenceImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*,

Implements pixel-wise the computation of absolute value difference.

This filter is parameterized over the types of the two input images and the type of the output image.

Numeric conversions (castings) are done by the C++ defaults.

The filter will walk over all the pixels in the two input images, and for each one of them it will do the following:

- Cast the input 1 pixel value to `double`.
- Cast the input 2 pixel value to `double`.
- Compute the difference of the two pixel values.
- Compute the absolute value of the difference.

- Cast the `double` value resulting from the absolute value to the pixel type of the output image.
- Store the casted value into the output image.

The filter expects all images to have the same dimension (e.g. all 2D, or all 3D, or all ND).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Absolute Value Of Difference Between Two Images

See itk::AbsoluteValueDifferenceImageFilter for additional documentation.

### Combine Two Images With Checker Board Pattern

### Synopsis

Combine two images by alternating blocks of a checkerboard pattern.

In this example the first image is black and the second one is white.

### Results

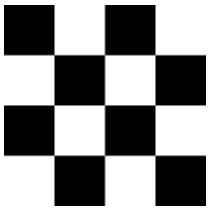


Fig. 140: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Combine Two Images With Checker Board Pattern."
)
parser.add_argument("output_image")
args = parser.parse_args()

dimension = 2
pixel_type = itk.UC
image_type = itk.Image[pixel_type, dimension]
```

(continues on next page)

(continued from previous page)

```
# Create an image
start = itk.Index[dimension]()
start.Fill(0)

size = itk.Size[dimension]()
size.Fill(100)

region = itk.ImageRegion[dimension]()
region.SetSize(size)
region.SetIndex(start)

image1 = image_type.New(Regions=region)
image1.Allocate()
image1.FillBuffer(0)

image2 = image_type.New(Regions=region)
image2.Allocate()
image2.FillBuffer(255)

output = itk.checker_board_image_filter(image1, image2)

itk.imwrite(output, args.output_image)
```

### C++

```
#include "itkImage.h"
#include "itkCheckerBoardImageFilter.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << " <OutputImage>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
```

(continues on next page)

(continued from previous page)

```
  ImageType::Pointer image1 = ImageType::New();
  image1->SetRegions(region);
  image1->Allocate();
  image1->FillBuffer(0);

  ImageType::Pointer image2 = ImageType::New();
  image2->SetRegions(region);
  image2->Allocate();
  image2->FillBuffer(255);

  using CheckerBoardFilterType = itk::CheckerBoardImageFilter<ImageType>;
  CheckerBoardFilterType::Pointer checkerBoardFilter = CheckerBoardFilterType::New();
  checkerBoardFilter->SetInput1(image1);
  checkerBoardFilter->SetInput2(image2);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(checkerBoardFilter->GetOutput());
  writer->SetFileName(argv[1]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class CheckerBoardImageFilter** : **public** itk::*ImageToImageFilter*<*TImage*, *TImage*>

Combines two images in a checkerboard pattern.

CheckerBoardImageFilter takes two input images that must have the same dimension, size, origin and spacing and produces an output image of the same size by combining the pixels from the two input images in a checkerboard pattern. This filter is commonly used for visually comparing two images, in particular for evaluating the results of an image registration process.

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Combine Two Images With Checker Board Pattern

See itk::CheckerBoardImageFilter for additional documentation.

### Squared Difference of Two Images

### Synopsis

Compute the squared difference of corresponding pixels in two images.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkSquaredDifferenceImageFilter.h"
#include "itkImageRegionIterator.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage1(UnsignedCharImageType::Pointer image);
static void
CreateImage2(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image1 = UnsignedCharImageType::New();
  CreateImage1(image1);

  UnsignedCharImageType::Pointer image2 = UnsignedCharImageType::New();
  CreateImage2(image2);

  using SquaredDifferenceImageFilterType =
    itk::SquaredDifferenceImageFilter<UnsignedCharImageType, UnsignedCharImageType,␣
↪FloatImageType>;

  SquaredDifferenceImageFilterType::Pointer squaredDifferenceFilter =␣
↪SquaredDifferenceImageFilterType::New();
  squaredDifferenceFilter->SetInput1(image1);
  squaredDifferenceFilter->SetInput2(image2);
  squaredDifferenceFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(UnsignedCharImageType::Pointer image)
{
  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(10);
```

(continues on next page)

(continued from previous page)

```
  UnsignedCharImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(255);

    ++imageIterator;
  }
}


void
CreateImage2(UnsignedCharImageType::Pointer image)
{
  // Create an image with 2 connected components
  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(10);

  UnsignedCharImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(100);

    ++imageIterator;
  }
}
```

#### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**, typename **TOutputImage**>
**class SquaredDifferenceImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutput*
Implements pixel-wise the computation of squared difference.

This filter is parameterized over the types of the two input images and the type of the output image.

Numeric conversions (castings) are done by the C++ defaults.

The filter will walk over all the pixels in the two input images, and for each one of them it will do the following:

- cast the input 1 pixel value to `double`
- cast the input 2 pixel value to `double`
- compute the difference of the two pixel values
- compute the square of the difference
- cast the `double` value resulting from `sqr()` to the pixel type of the output image
- store the casted value into the output image.

The filter expect all images to have the same dimension (e.g. all 2D, or all 3D, or all ND)

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Squared Difference Of Two Images

See itk::SquaredDifferenceImageFilter for additional documentation.

### 3.4.11 ImageCompose

#### Compose Vector From Three Scalar Images

#### Synopsis

Compose a vector image (with 3 components) from three scalar images.

#### Results

Output:

```
1.2
[1.2, 1.2, 1.2]
```

### Code

### C++

```
#include "itkImageAdaptor.h"
#include "itkImageRegionIterator.h"
#if ITK_VERSION_MAJOR < 4
#  include "itkCompose3DCovariantVectorImageFilter.h"
#else
#  include "itkComposeImageFilter.h"
#endif

using VectorImageType = itk::Image<itk::CovariantVector<float, 3>, 2>;
using ScalarImageType = itk::Image<float, 2>;

static void
CreateImage(ScalarImageType::Pointer image);

int
main(int, char *[])
{
  ScalarImageType::Pointer image = ScalarImageType::New();
  CreateImage(image);

#if ITK_VERSION_MAJOR < 4
  using ComposeCovariantVectorImageFilterType =
    itk::Compose3DCovariantVectorImageFilter<ScalarImageType, VectorImageType>;
#else
  using ComposeCovariantVectorImageFilterType = itk::ComposeImageFilter
↪<ScalarImageType, VectorImageType>;
#endif
  ComposeCovariantVectorImageFilterType::Pointer composeFilter =␣
↪ComposeCovariantVectorImageFilterType::New();

  composeFilter->SetInput1(image);
  composeFilter->SetInput2(image);
  composeFilter->SetInput3(image);
  composeFilter->Update();

  itk::Index<2> index;
  index.Fill(0);

  std::cout << image->GetPixel(index) << std::endl;

  std::cout << composeFilter->GetOutput()->GetPixel(index) << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ScalarImageType::Pointer image)
{
  ScalarImageType::IndexType start;
  start.Fill(0);

  ScalarImageType::SizeType size;
  size.Fill(2);
```

(continues on next page)

(continued from previous page)

```
  ScalarImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ScalarImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(1.2);

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *VectorImage*<**typename** *TInputImage*::PixelType, *TInputImage*::
**class ComposeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

ComposeImageFilter combine several scalar images into a multicomponent image.

ComposeImageFilter combine several scalar images into an itk::Image of vector pixel (itk::Vector, itk::RGBPixel, . . . ), of std::complex pixel, or in an itk::VectorImage.

**Inputs and Usage**

```
filter->SetInput( 0, image0 );
filter->SetInput( 1, image1 );
...
filter->Update();
itk::VectorImage< PixelType, dimension >::Pointer = filter->GetOutput();
```

All input images are expected to have the same template parameters and have the same size and origin.

**See** VectorImage

**See** VectorIndexSelectionCastImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Vector Image From Scalar Images
- Compose Vector From Three Scalar Images
- Convert Real And Imaginary Images To Complex Image

See itk::ComposeImageFilter for additional documentation.

### Convert Real and Imaginary Images to Complex Image

### Synopsis

Convert a real image and an imaginary image to a complex image.

### Results

Output:

```
Image (0x7ff58250cf20)
RTTI typeinfo:   itk::Image<std::__1::complex<float>, 2u>
Reference Count: 1
Modified Time: 25
Debug: Off
Object Name:
Observers:
  none
Source: (0x7ff58250c590)
Source output name: Primary
Release Data: Off
Data Released: False
Global Release Data: Off
PipelineMTime: 22
UpdateMTime: 26
RealTimeStamp: 0 seconds
LargestPossibleRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [0, 0]
BufferedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [0, 0]
RequestedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [0, 0]
Spacing: [1, 1]
Origin: [0, 0]
Direction:
1 0
0 1

IndexToPointMatrix:
1 0
0 1

PointToIndexMatrix:
1 0
0 1

Inverse Direction:
1 0
0 1
```

(continues on next page)

(continued from previous page)

```
PixelContainer:
  ImportImageContainer (0x7ff58250d0e0)
    RTTI typeinfo:   itk::ImportImageContainer<unsigned long, std::__1::complex<float>
↪ >
    Reference Count: 1
    Modified Time: 24
    Debug: Off
    Object Name:
    Observers:
      none
    Pointer: 0x7ff58250c1b0
    Container manages memory: true
    Size: 0
    Capacity: 0
```

### Code

### C++

```
#include "itkImage.h"

#if ITK_VERSION_MAJOR < 4
#  include "itkRealAndImaginaryToComplexImageFilter.h"
#else
#  include "itkComposeImageFilter.h"
#endif

#include <complex>

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  using ImageType = itk::Image<unsigned char, 2>;
  using ComplexImageType = itk::Image<std::complex<float>, 2>;

  ImageType::Pointer realImage = ImageType::New();
  ImageType::Pointer imaginaryImage = ImageType::New();

#if ITK_VERSION_MAJOR < 4
  using RealAndImaginaryToComplexImageFilterType =
    itk::RealAndImaginaryToComplexImageFilter<ImageType, ComplexImageType>;
#else
  using RealAndImaginaryToComplexImageFilterType = itk::ComposeImageFilter<ImageType,␣
↪ComplexImageType>;
#endif
  RealAndImaginaryToComplexImageFilterType::Pointer␣
↪realAndImaginaryToComplexImageFilter =
    RealAndImaginaryToComplexImageFilterType::New();
  realAndImaginaryToComplexImageFilter->SetInput1(realImage);
  realAndImaginaryToComplexImageFilter->SetInput2(imaginaryImage);
  realAndImaginaryToComplexImageFilter->Update();

  ComplexImageType * output = realAndImaginaryToComplexImageFilter->GetOutput();
  output->Print(std::cout);
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *VectorImage*<**typename** *TInputImage*::PixelType, *TInputImage*::
**class ComposeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

ComposeImageFilter combine several scalar images into a multicomponent image.

ComposeImageFilter combine several scalar images into an itk::Image of vector pixel (itk::Vector, itk::RGBPixel, . . . ), of std::complex pixel, or in an itk::VectorImage.

**Inputs and Usage**

```
filter->SetInput( 0, image0 );
filter->SetInput( 1, image1 );
...
filter->Update();
itk::VectorImage< PixelType, dimension >::Pointer = filter->GetOutput();
```

All input images are expected to have the same template parameters and have the same size and origin.

**See** VectorImage

**See** VectorIndexSelectionCastImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Vector Image From Scalar Images
- Compose Vector From Three Scalar Images
- Convert Real And Imaginary Images To Complex Image

See itk::ComposeImageFilter for additional documentation.

## Create Vector Image From Scalar Images

## Synopsis

Create a vector image from a collection of scalar images.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkComposeImageFilter.h"
#include "itkVectorImage.h"

namespace
{
using VectorImageType = itk::VectorImage<unsigned char, 2>;
using ScalarImageType = itk::Image<unsigned char, 2>;
} // namespace

static void
CreateImage(ScalarImageType::Pointer image);

int
main(int, char *[])
{
  ScalarImageType::Pointer image0 = ScalarImageType::New();
  CreateImage(image0);

  ScalarImageType::Pointer image1 = ScalarImageType::New();
  CreateImage(image1);

  ScalarImageType::Pointer image2 = ScalarImageType::New();
  CreateImage(image2);

  using ImageToVectorImageFilterType = itk::ComposeImageFilter<ScalarImageType>;
  ImageToVectorImageFilterType::Pointer imageToVectorImageFilter =␣
↪ImageToVectorImageFilterType::New();
  imageToVectorImageFilter->SetInput(0, image0);
  imageToVectorImageFilter->SetInput(1, image1);
  imageToVectorImageFilter->SetInput(2, image2);
  imageToVectorImageFilter->Update();

  VectorImageType::Pointer vectorImage = imageToVectorImageFilter->GetOutput();

  return EXIT_SUCCESS;
}

void
CreateImage(ScalarImageType::Pointer image)
{
  ScalarImageType::IndexType start;
  start.Fill(0);

  ScalarImageType::SizeType size;
  size.Fill(100);

  ScalarImageType::RegionType region(start, size);

  image->SetRegions(region);
```

(continues on next page)

(continued from previous page)

```
  image->Allocate();
  image->FillBuffer(0);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *VectorImage*<**typename** *TInputImage*::PixelType, *TInputImage*::
**class ComposeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

ComposeImageFilter combine several scalar images into a multicomponent image.

ComposeImageFilter combine several scalar images into an itk::Image of vector pixel (itk::Vector, itk::RGBPixel, . . . ), of std::complex pixel, or in an itk::VectorImage.

**Inputs and Usage**

```
filter->SetInput( 0, image0 );
filter->SetInput( 1, image1 );
...
filter->Update();
itk::VectorImage< PixelType, dimension >::Pointer = filter->GetOutput();
```

All input images are expected to have the same template parameters and have the same size and origin.

**See** VectorImage

**See** VectorIndexSelectionCastImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Vector Image From Scalar Images
- Compose Vector From Three Scalar Images
- Convert Real And Imaginary Images To Complex Image

See itk::ComposeImageFilter for additional documentation.

### Join Images

### Synopsis

Join images, stacking their components

### Results

Output:

```
0
10
```

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkJoinImageFilter.h"
#include "itkVectorImageToImageAdaptor.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image, unsigned char value);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage(image1, 0);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage(image2, 10);

  using JoinImageFilterType = itk::JoinImageFilter<ImageType, ImageType>;

  JoinImageFilterType::Pointer joinFilter = JoinImageFilterType::New();
  joinFilter->SetInput1(image1);
  joinFilter->SetInput2(image2);
  joinFilter->Update();

  itk::Index<2> index;
  index[0] = 0;
  index[1] = 0;

  std::cout << static_cast<int>(joinFilter->GetOutput()->GetPixel(index)[0]) << std::
↪endl;
  std::cout << static_cast<int>(joinFilter->GetOutput()->GetPixel(index)[1]) << std::
↪endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image, unsigned char value)
{
  // Create an image
  ImageType::IndexType start;
```

(continues on next page)

(continued from previous page)

```
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(value);
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**>
**class JoinImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, Functor::MakeJoin<*TInputI*

Join two images, resulting in an image where each pixel has the components of the first image followed by the components of the second image.

JoinImageFilter combines two images by appending the components of one image to the components of another image. The output image type is always a itk::Vector image and the vector value type will the smallest type that can represent the dynamic range of both the input value types. Hence, joining an image of char and unsigned char results in an image of shorts since that is the smallest datatype with a large enough dynamic range. To define a consistent behavior across different architectures, the join of an int and an unsigned int is float. On a 64 bit architecture, this join could be represented in a long. But on 32 bit architectures, the only safe join value type is a float. For this and similar ambiguous cases, the join value type is promoted to a float.

Note that this filter is not templated over its output image type. Rather the filter determines what its output image type is based on the input data types. To determine the output type, use JoinImageFilter<Image1, Image2>::OutputImageType

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Join Images

See itk::JoinImageFilter for additional documentation.

## 3.4.12 ImageFeature

### Add Gaussian Noise to an Image

### Synopsis

Adds Gaussian Noise to a particular image

### Results

Fig. 141: Input Image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Additive Gaussian Noise Image Filter.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("mean", type=float)
parser.add_argument("standard_deviation", type=float)
args = parser.parse_args()

# Use unsigned char to save to PNG format
InputPixelType = itk.UC
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
```

(continues on next page)

Fig. 142: Output Image

(continued from previous page)

```
reader.SetFileName(args.input_image)

FilterType = itk.AdditiveGaussianNoiseImageFilter[InputImageType, InputImageType]
AdditiveFilter = FilterType.New()
AdditiveFilter.SetInput(reader.GetOutput())
AdditiveFilter.SetMean(args.mean)
AdditiveFilter.SetStandardDeviation(args.standard_deviation)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(AdditiveFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkAdditiveGaussianNoiseImageFilter.h"

int
main(int argc, char * argv[])
{
  // Check for proper arguments; if not, explain usage.
  if (argc != 5)
```

(continues on next page)

(continued from previous page)

```
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> [Mean] [Standard Deviation]";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  // Initialize and assign user provided variables
  const char * inputImage = argv[1];
  const char * outputImage = argv[2];

  // Get floating point numbers for the mean and std dev to perform the algorithm
  const double mean = std::stod(argv[3]);
  const double deviation = std::stod(argv[4]);

  constexpr unsigned int Dimension = 2;
  // Use unsigned char to save to PNG format
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  // Read the file to be converted
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using ImageType = itk::Image<PixelType, Dimension>;

  // Create the filter and apply the algorithm to the image
  using FilterType = itk::AdditiveGaussianNoiseImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetMean(mean);                    // Set the mean
  filter->SetStandardDeviation(deviation); // Set the standard deviation

  // Set the writer to save file
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputImage);
  writer->SetInput(filter->GetOutput());

  // Write the output image
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<class **TInputImage**, class **TOutputImage** = *TInputImage*>
**class AdditiveGaussianNoiseImageFilter** : **public** itk::NoiseBaseImageFilter<*TInputImage*, *TOutputImage*>

Alter an image with additive Gaussian white noise.

Additive Gaussian white noise can be modeled as:

The noise is independent of the pixel intensities.

$I = I_0 + N$

where $I$ is the observed image, $I_0$ is the noise-free image and $N$ is a normally distributed random variable of mean $\mu$ and variance $\sigma^2$:

$N \sim \mathcal{N}(\mu, \sigma^2)$

This code was contributed in the Insight Journal paper "Noise

Simulation". https://www.insight-journal.org/browse/publication/721

**Author** Gaetan Lehmann

See itk::AdditiveGaussianNoiseImageFilter for additional documentation.

### Apply a Filter Only to a Specified Image Region

### Synopsis

To apply a filter to only operate on a specified ImageRegion.

### Results

Output:

```
Created random image.
Computed derivative.
```

### Code

### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkDerivativeImageFilter.h"

int
main(int, char *[])
{
  using ImageType = itk::Image<float, 2>;

  itk::Size<2> smallSize;
  smallSize.Fill(10);

  itk::Index<2> index;
```

(continues on next page)

(continued from previous page)

```
  index.Fill(0);

  itk::ImageRegion<2> region(index, smallSize);

  itk::Size<2> bigSize;
  bigSize.Fill(10000);

  itk::RandomImageSource<ImageType>::Pointer randomImageSource = itk::
↪RandomImageSource<ImageType>::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce non-random results
  randomImageSource->SetSize(bigSize);
  randomImageSource->GetOutput()->SetRequestedRegion(smallSize);
  randomImageSource->Update();

  std::cout << "Created random image." << std::endl;

  using DerivativeImageFilterType = itk::DerivativeImageFilter<ImageType, ImageType>;
  DerivativeImageFilterType::Pointer derivativeFilter = DerivativeImageFilterType::
↪New();
  derivativeFilter->SetInput(randomImageSource->GetOutput());
  derivativeFilter->SetDirection(0); // "x" axis
  derivativeFilter->GetOutput()->SetRequestedRegion(smallSize);
  derivativeFilter->Update();

  std::cout << "Computed derivative." << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class DerivativeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Computes the directional derivative of an image. The directional derivative at each pixel location is computed by convolution with a derivative operator of user-specified order.

SetOrder specifies the order of the derivative.

SetDirection specifies the direction of the derivative with respect to the coordinate axes of the image.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply A Filter Only To A Specified Region Of An Image
- Apply A Filter Only To A Specified Region Of An Image

See itk::DerivativeImageFilter for additional documentation.

### Apply a Filter to a Specified Region of an Image

### Synopsis

Computes the derivative of an image in a particular direction.

### Results

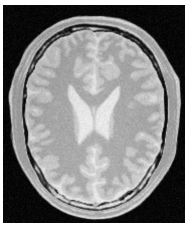


Fig. 143: Input image

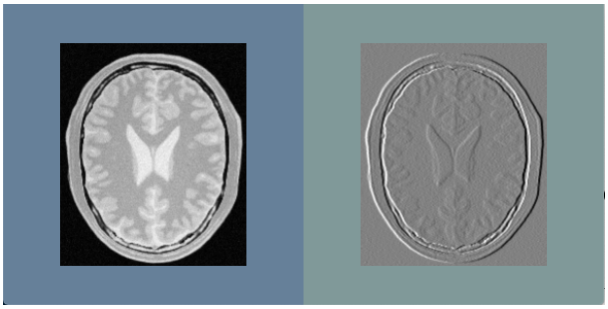


Fig. 144: Output in QuickView

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkDerivativeImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile" << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using FloatImageType = itk::Image<float, 2>;
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;

  using readerType = itk::ImageFileReader<UnsignedCharImageType>;

  using filterType = itk::DerivativeImageFilter<UnsignedCharImageType, FloatImageType>
↪;

  // Create and setup a reader
  readerType::Pointer reader = readerType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a derivative filter
  filterType::Pointer derivativeFilter = filterType::New();
  derivativeFilter->SetInput(reader->GetOutput());
  derivativeFilter->SetDirection(0); // "x" axis

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;

  viewer.AddImage<UnsignedCharImageType>(reader->GetOutput());
  viewer.AddImage<FloatImageType>(derivativeFilter->GetOutput());
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class DerivativeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Computes the directional derivative of an image. The directional derivative at each pixel location is computed by convolution with a derivative operator of user-specified order.

SetOrder specifies the order of the derivative.

SetDirection specifies the direction of the derivative with respect to the coordinate axes of the image.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Apply A Filter Only To A Specified Region Of An Image
- Apply A Filter Only To A Specified Region Of An Image

See itk::DerivativeImageFilter for additional documentation.

### Bilateral Filter an Image

### Synopsis

Bilateral filter an image.

### Results

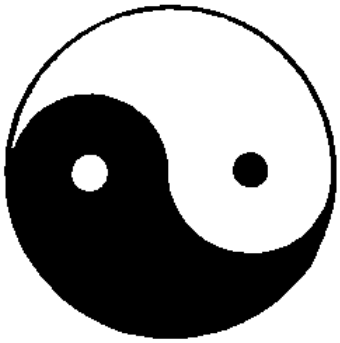


Fig. 145: Input image.


Fig. 146: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkBilateralImageFilter.h"
#include "itkSubtractImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile [domainSigma] [rangeSigma]" << std::endl;
    return EXIT_FAILURE;
  }
  double rangeSigma = 2.0;
  double domainSigma = 2.0;
  if (argc > 2)
  {
    domainSigma = std::stod(argv[2]);
  }
  if (argc > 3)
  {
    rangeSigma = std::stod(argv[3]);
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using ImageType = itk::Image<float, 2>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  using FilterType = itk::BilateralImageFilter<ImageType, ImageType>;
  using SubtractImageFilterType = itk::SubtractImageFilter<ImageType>;

  // Create and setup a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a derivative filter
  FilterType::Pointer bilateralFilter = FilterType::New();
  bilateralFilter->SetInput(reader->GetOutput());
  bilateralFilter->SetDomainSigma(domainSigma);
  bilateralFilter->SetRangeSigma(rangeSigma);

  SubtractImageFilterType::Pointer diff = SubtractImageFilterType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(bilateralFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  std::stringstream desc;
  desc << "Bilateral\ndomainSigma = " << domainSigma << " rangeSigma = " <<␣
↪rangeSigma;
  viewer.AddImage(bilateralFilter->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - Bilateral";
  viewer.AddImage(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BilateralImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Blurs an image while preserving edges.

This filter uses bilateral filtering to blur an image using both domain and range "neighborhoods". Pixels that are close to a pixel in the image domain and similar to a pixel in the image range are used to calculate the filtered value. Two gaussian kernels (one in the image domain and one in the image range) are used to smooth the image. The result is an image that is smoothed in homogeneous regions yet has edges preserved. The result is similar to anisotropic diffusion but the implementation in non-iterative. Another benefit to bilateral filtering is that any distance metric can be used for kernel smoothing the image range. Hence, color images can be smoothed as vector images, using the CIE distances between intensity values as the similarity metric (the Gaussian kernel for the image domain is evaluated using CIE distances). A separate version of this filter will be designed for color and vector images.

Bilateral filtering is capable of reducing the noise in an image by an order of magnitude while maintaining edges.

The bilateral operator used here was described by Tomasi and Manduchi (Bilateral Filtering for Gray and ColorImages. IEEE ICCV. 1998.)

**See** GaussianOperator

**See** RecursiveGaussianImageFilter

**See** DiscreteGaussianImageFilter

**See** AnisotropicDiffusionImageFilter

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Bilateral Filter An Image

See itk::BilateralImageFilter for additional documentation.

### Compute Laplacian

### Synopsis

This filter computes the Laplacian of a scalar-valued image.

### Results

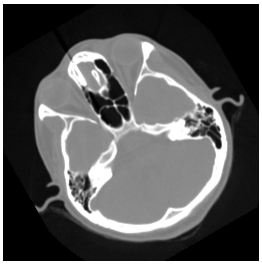


Fig. 147: Input image

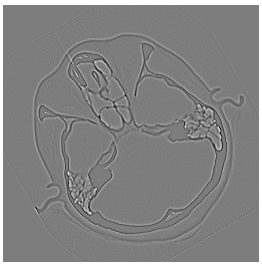


Fig. 148: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Laplacian.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

InputPixelType = itk.F
```

(continues on next page)

(continued from previous page)

```
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.LaplacianImageFilter[InputImageType, InputImageType]
laplacianFilter = FilterType.New()
laplacianFilter.SetInput(reader.GetOutput())

RescaleFilterType = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType]
rescaler = RescaleFilterType.New()
rescaler.SetInput(laplacianFilter.GetOutput())

outputPixelTypeMinimum = itk.NumericTraits[OutputPixelType].min()
outputPixelTypeMaximum = itk.NumericTraits[OutputPixelType].max()

rescaler.SetOutputMinimum(outputPixelTypeMinimum)
rescaler.SetOutputMaximum(outputPixelTypeMaximum)

WriterType = itk.ImageFileWriter[OutputImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLaplacianImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  using InputPixelType = float;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::LaplacianImageFilter<InputImageType, InputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(rescaler->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LaplacianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

This filter computes the Laplacian of a scalar-valued image.

The Laplacian is an isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. Often, the Laplacian is applied to an image that has first been smoothed with a Gaussian filter in order to reduce its sensitivity to noise.

The Laplacian at each pixel location is computed by convolution with the itk::LaplacianOperator.

**Inputs and Outputs** The input to this filter is a scalar-valued itk::Image of arbitrary dimension. The output is a scalar-valued itk::Image.

**Warning** The pixel type of the input and output images must be of real type (float or double). ConceptChecking

is used here to enforce the input pixel type. You will get a compilation error if the pixel type of the input and output images is not float or double.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**See** LaplacianOperator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Laplacian

See itk::LaplacianImageFilter for additional documentation.

### Detect Edges With Canny Edge Detection Filter

### Synopsis

Apply CannyEdgeDetectionImageFilter to an image

### Results

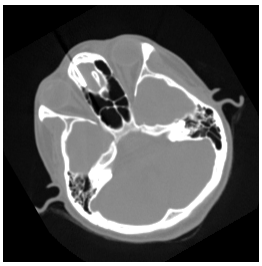


Fig. 149: Input image

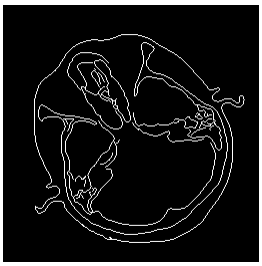


Fig. 150: Rescaled Output image (values are in [0, 255])

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Detect Edges With Canny Edge Detection Filter."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("variance", type=float)
parser.add_argument("lower_threshold", type=float)
parser.add_argument("upper_threshold", type=float)
args = parser.parse_args()

InputPixelType = itk.F
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

reader = itk.ImageFileReader[InputImageType].New()
reader.SetFileName(args.input_image)

cannyFilter = itk.CannyEdgeDetectionImageFilter[InputImageType, InputImageType].New()
cannyFilter.SetInput(reader.GetOutput())
cannyFilter.SetVariance(args.variance)
cannyFilter.SetLowerThreshold(args.lower_threshold)
cannyFilter.SetUpperThreshold(args.upper_threshold)

rescaler = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType].New()
rescaler.SetInput(cannyFilter.GetOutput())
rescaler.SetOutputMinimum(0)
rescaler.SetOutputMaximum(255)

writer = itk.ImageFileWriter[OutputImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCannyEdgeDetectionImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputImage> <OutputImage> ";
    std::cerr << "<Variance> <LowerThreshold> <UpperThreshold>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using InputPixelType = float;
  using OutputPixelType = unsigned char;

  const char *          inputImage = argv[1];
  const char *          outputImage = argv[2];
  const InputPixelType variance = std::stod(argv[3]);
  const InputPixelType lowerThreshold = std::stod(argv[4]);
  const InputPixelType upperThreshold = std::stod(argv[5]);

  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using FilterType = itk::CannyEdgeDetectionImageFilter<InputImageType,␣
↪InputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetVariance(variance);
  filter->SetLowerThreshold(lowerThreshold);
  filter->SetUpperThreshold(upperThreshold);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(255);

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputImage);
  writer->SetInput(rescaler->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CannyEdgeDetectionImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
This filter is an implementation of a Canny edge detector for scalar-valued images.

Based on John Canny's paper "A Computational Approach

to Edge Detection"(IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No.6, November 1986), there are four major steps used in the edge-detection scheme: (1) Smooth the input image with Gaussian filter. (2) Calculate the second directional derivatives of the smoothed image. (3) Non-Maximum Suppression: the zero-crossings of 2nd derivative are found, and the sign of third derivative is used to find the correct extrema. (4) The hysteresis thresholding is applied to the gradient magnitude (multiplied with zero-crossings) of the smoothed image to find and link edges.

**Inputs and Outputs** The input to this filter should be a scalar, real-valued Itk image of arbitrary dimension. The output should also be a scalar, real-value Itk image of the same dimensionality.

**Parameters** There are four parameters for this filter that control the sub-filters used by the algorithm.

Variance and Maximum error are used in the Gaussian smoothing of the input image. See itkDiscreteGaussianImageFilter for information on these parameters.

Threshold is the lowest allowed value in the output image. Its data type is the same as the data type of the output image. Any values below the Threshold level will be replaced with the OutsideValue parameter value, whose default is zero.

**See** DiscreteGaussianImageFilter

**See** ZeroCrossingImageFilter

**See** ThresholdImageFilter

See itk::CannyEdgeDetectionImageFilter for additional documentation.

### Extract Contours From Image

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Extract contours from an image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
/*
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkSimpleContourExtractorImageFilter.h"
#include "itkImageRegionIterator.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void CreateImage(UnsignedCharImageType::Pointer image);

int main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  using SimpleContourExtractorImageFilterType = itk::
↪SimpleContourExtractorImageFilter <UnsignedCharImageType, UnsignedCharImageType>;
  SimpleContourExtractorImageFilterType::Pointer contourFilter
          = SimpleContourExtractorImageFilterType::New();
  contourFilter->SetInput(image);
  contourFilter->Update();

  using WriterType = itk::ImageFileWriter< UnsignedCharImageType  >;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(contourFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
void CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start,size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a line pixels
  for(unsigned int i = 40; i < 60; ++i)
    {
    itk::Index<2> pixel;
    pixel.Fill(i);
    image->SetPixel(pixel, 255);
    }

  // Create another line of pixels
  for(unsigned int i = 10; i < 20; ++i)
    {
    itk::Index<2> pixel;
    pixel[0] = 10;
    pixel[1] = i;
    image->SetPixel(pixel, 255);
    }

  using WriterType = itk::ImageFileWriter< UnsignedCharImageType  >;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SimpleContourExtractorImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>
Computes an image of contours which will be the contour of the first image.

A pixel of the source image is considered to belong to the contour if its pixel value is equal to the input foreground value and it has in its neighborhood at least one pixel which its pixel value is equal to the input background value. The output image will have pixels which will be set to the output foreground value if they belong to the contour, otherwise they will be set to the output background value.

The neighborhood "radius" is set thanks to the radius params.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Extract Contours From Image

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

See itk::SimpleContourExtractorImageFilter for additional documentation.

### Find Zero Crossings in Signed Image

### Synopsis

Find zero crossings in a signed image.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkZeroCrossingImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using ImageType = itk::Image<float, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  if (argc < 2)
  {
    CreateImage(image);
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
```

(continues on next page)

(continued from previous page)

```
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  using FilterType = itk::ZeroCrossingImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(image);
  filter->SetBackgroundValue(0);
  filter->SetForegroundValue(255);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "Zero Crossing";
  viewer.AddImage(filter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(-1);

  // Make half of the image negative
  for (unsigned int i = 0; i < 100; ++i)
  {
    for (unsigned int j = 0; j < 50; ++j)
    {
      itk::Index<2> index;
      index[0] = i;
      index[1] = j;
      image->SetPixel(index, 1);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ZeroCrossingImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
This filter finds the closest pixel to the zero-crossings (sign changes) in a signed itk::Image.

Pixels closest to zero-crossings are labeled with a foreground value. All other pixels are marked with a background value. The algorithm works by detecting differences in sign among neighbors using city-block style connectivity (4-neighbors in 2d, 6-neighbors in 3d, etc.).

**Inputs and Outputs** The input to this filter is an itk::Image of arbitrary dimension. The algorithm assumes a signed data type (zero-crossings are not defined for unsigned data types), and requires that operator>, operator<, operator==, and operator!= are defined.

The output of the filter is a binary, labeled image of user-specified type. By default, zero-crossing pixels are labeled with a default "foreground" value of itk::NumericTraits<OutputDataType>::OneValue(), where OutputDataType is the data type of the output image. All other pixels are labeled with a default "background" value of itk::NumericTraits<OutputDataType>::ZeroValue().

**Parameters** There are two parameters for this filter. ForegroundValue is the value that marks zero-crossing pixels. The BackgroundValue is the value given to all other pixels.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Find Zero Crossings In Signed Image

See itk::ZeroCrossingImageFilter for additional documentation.

### Laplacian Recursive Gaussian Image Filter

### Synopsis

Compute the Laplacian of an image.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse
```

(continues on next page)

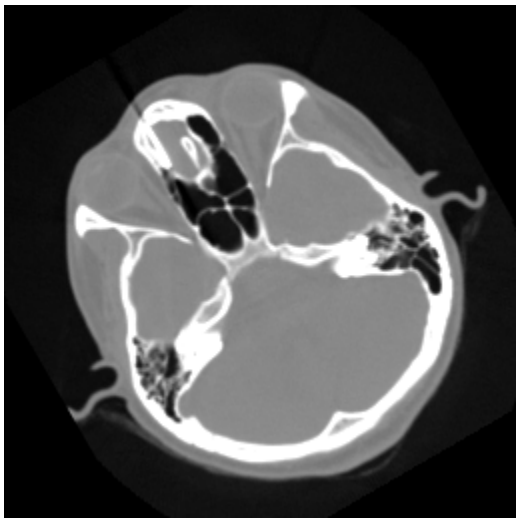


Fig. 151: Input image

Fig. 152: Rescaled Output image [0, 255]

(continued from previous page)

```
parser = argparse.ArgumentParser(
    description="Laplacian Recursive Gaussian Image Filter."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

input_image = itk.imread(args.input_image, pixel_type=itk.F)

output_image = itk.laplacian_recursive_gaussian_image_filter(input_image)

itk.imwrite(output_image, args.output_image)
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLaplacianRecursiveGaussianImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  using OutputPixelType = float;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::LaplacianRecursiveGaussianImageFilter<InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->Update();

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class LaplacianRecursiveGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Computes the Laplacian of Gaussian (LoG) of an image.

Computes the Laplacian of Gaussian (LoG) of an image by convolution with the second derivative of a Gaussian. This filter is implemented using the recursive gaussian filters.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Laplacian

See itk::LaplacianRecursiveGaussianImageFilter for additional documentation.

### Segment Blood Vessels

### Synopsis

The example takes an image (say MRA image), computes the vesselness measure of the image using the HessianRecursiveGaussianImageFilter and the Hessian3DToVesselnessMeasureImageFilter. The goal is to detect bright, tubular structures in the image.

Note that since the algorithm is based on the Hessian, it will also identify black tubular structures.

An important parameter is the *Sigma* that determines the amount of smoothing applied during Hessian estimation. A larger *Sigma* will decrease the identification of noise or small structures as vessels. In this example, the *Sigma* is large enough only vessels comprising the Circle of Willis and other large vessels are segmented.

### Results

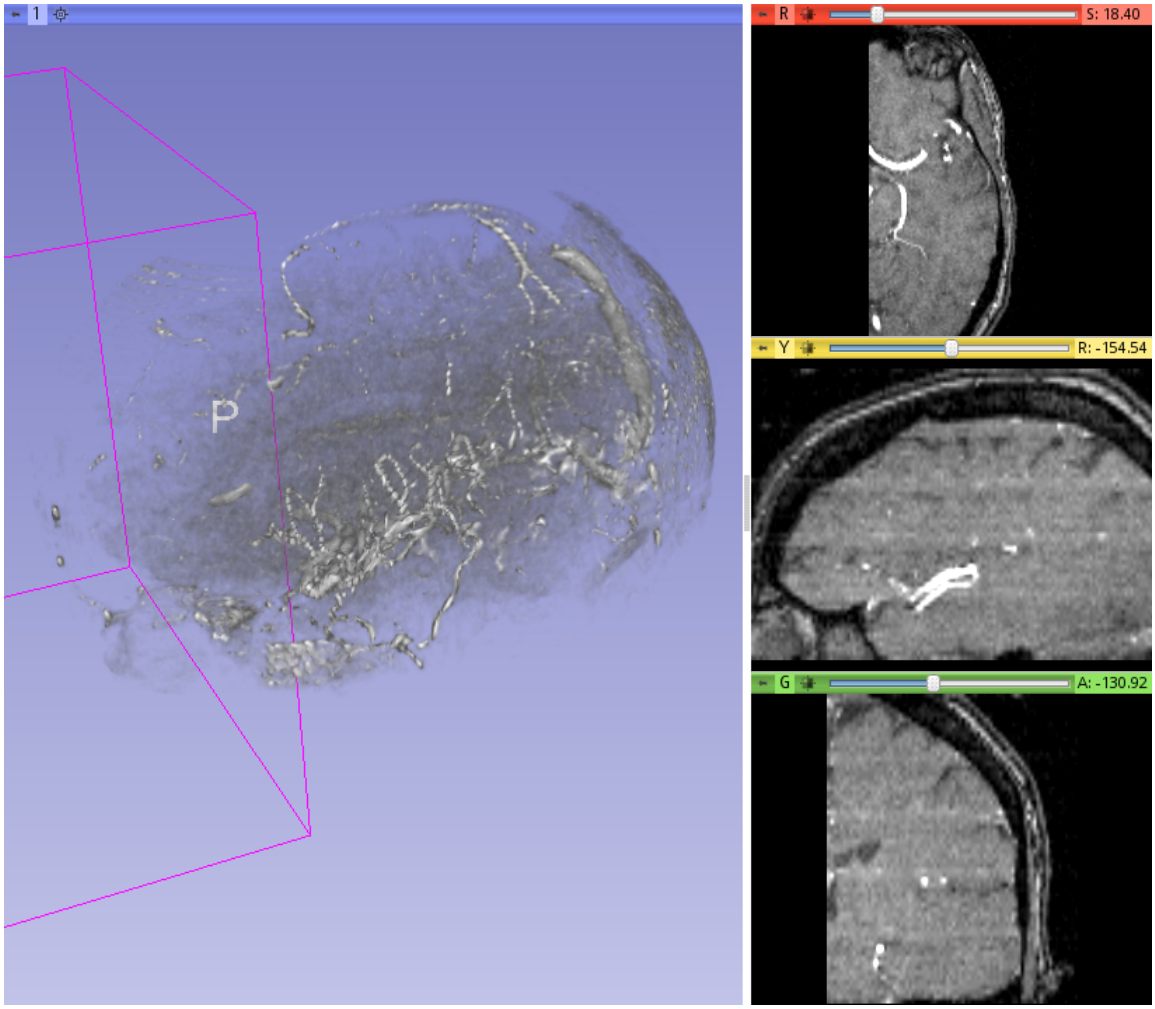



Fig. 153: Input head MRA.

### Code

### Python

```
#!/usr/bin/env python

import argparse

import itk
from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("5.0.0"):
    print("ITK 5.0.0 or newer is required.")
    sys.exit(1)
```

(continues on next page)

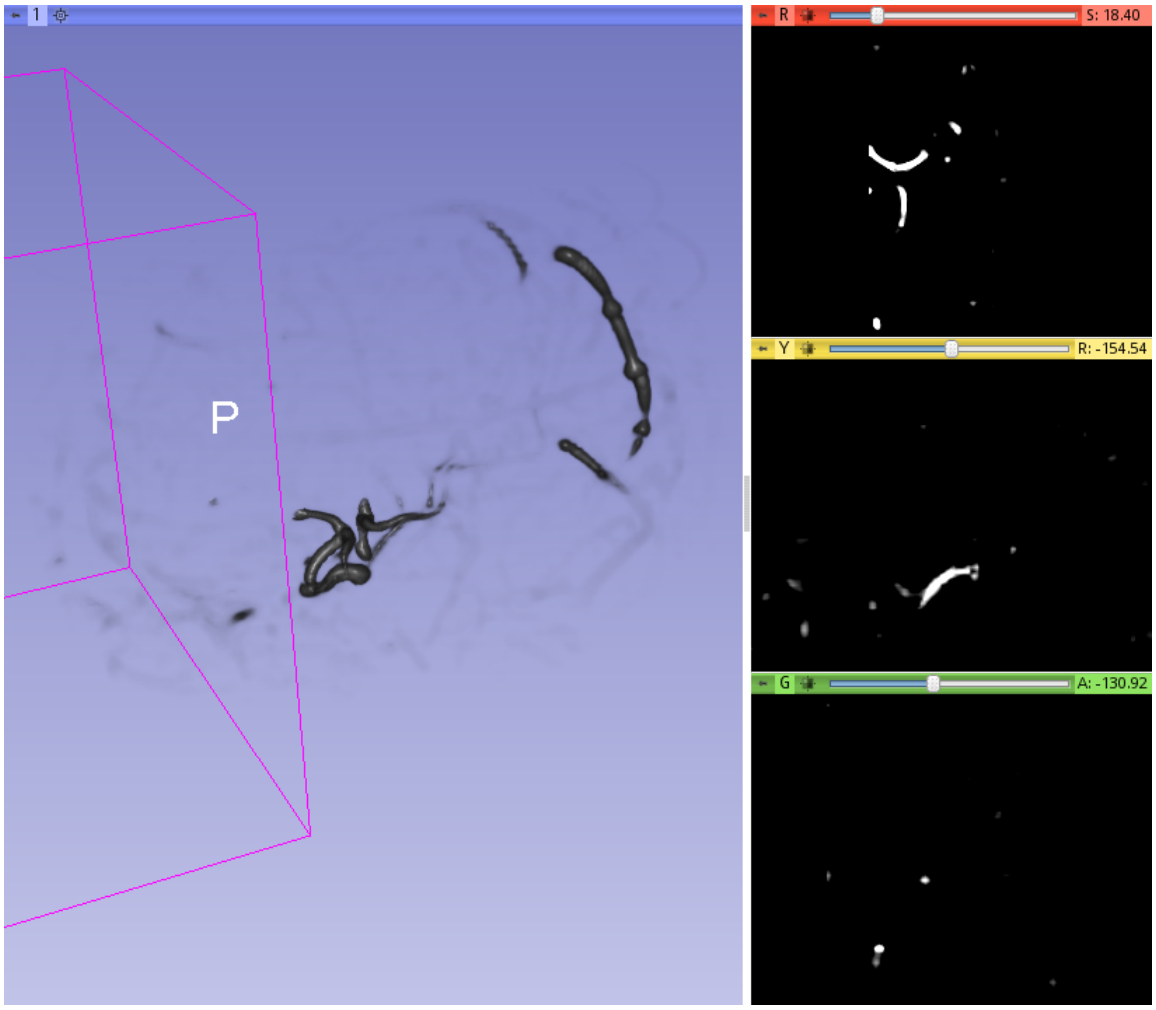



Fig. 154: Output vessel segmentation.

(continued from previous page)

```
parser = argparse.ArgumentParser(description="Segment blood vessels.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("--sigma", type=float, default=1.0)
parser.add_argument("--alpha1", type=float, default=0.5)
parser.add_argument("--alpha2", type=float, default=2.0)
args = parser.parse_args()

input_image = itk.imread(args.input_image, itk.ctype("float"))

hessian_image = itk.hessian_recursive_gaussian_image_filter(
    input_image, sigma=args.sigma
)

vesselness_filter = itk.Hessian3DToVesselnessMeasureImageFilter[
    itk.ctype("float")
].New()
vesselness_filter.SetInput(hessian_image)
vesselness_filter.SetAlpha1(args.alpha1)
vesselness_filter.SetAlpha2(args.alpha2)

itk.imwrite(vesselness_filter, args.output_image)
```

### C++

```
#include "itkHessian3DToVesselnessMeasureImageFilter.h"
#include "itkHessianRecursiveGaussianImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: <InputImage> <OutputImage> [Sigma] [Alpha1] [Alpha2]" << std:
↪:endl;
    return EXIT_FAILURE;
  }
  const char * inputImage = argv[1];
  const char * outputImage = argv[2];
  double       sigma = 1.0;
  if (argc > 3)
  {
    sigma = std::atof(argv[3]);
  }
  double alpha1 = 0.5;
  if (argc > 4)
  {
    alpha1 = std::atof(argv[4]);
  }
  double alpha2 = 2.0;
  if (argc > 5)
  {
```

(continues on next page)

(continued from previous page)

```
    alpha2 = std::atof(argv[5]);
  }

  constexpr unsigned int Dimension = 3;
  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using HessianFilterType = itk::HessianRecursiveGaussianImageFilter<ImageType>;
  HessianFilterType::Pointer hessianFilter = HessianFilterType::New();
  hessianFilter->SetInput(reader->GetOutput());
  hessianFilter->SetSigma(sigma);

  using VesselnessMeasureFilterType = itk::Hessian3DToVesselnessMeasureImageFilter
↪<PixelType>;
  VesselnessMeasureFilterType::Pointer vesselnessFilter = VesselnessMeasureFilterType:
↪:New();
  vesselnessFilter->SetInput(hessianFilter->GetOutput());
  vesselnessFilter->SetAlpha1(alpha1);
  vesselnessFilter->SetAlpha2(alpha2);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(vesselnessFilter->GetOutput());
  writer->SetFileName(outputImage);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**>
**class Hessian3DToVesselnessMeasureImageFilter** : **public** itk::*ImageToImageFilter*<*Image*<SymmetricSecondRanl
Line filter to provide a vesselness measure for tubular objects from the hessian matrix.

The filter takes as input an image of hessian pixels (SymmetricSecondRankTensor pixels) and preserves pixels that have eigen values $\lambda_3$ close to 0 and $\lambda_2$ and $\lambda_1$ as large negative values (for bright tubular structures).

$$|\lambda_1| < |\lambda_2| < |\lambda_3|$$

- Bright tubular structures will have low $\lambda_1$ and large negative values of $\lambda_2$ and $\lambda_3$.

- Conversely dark tubular structures will have a low value of $\lambda_1$ and large positive values of $\lambda_2$ and $\lambda_3$.
- Bright plate like structures have low values of $\lambda_1$ and $\lambda_2$ and large negative values of $\lambda_3$
- Dark plate like structures have low values of $\lambda_1$ and $\lambda_2$ and large positive values of $\lambda_3$
- Bright spherical (blob) like structures have all three eigen values as large negative numbers
- Dark spherical (blob) like structures have all three eigen values as large positive numbers

This filter is used to discriminate the Bright tubular structures.

**Notes:** The filter takes into account that the eigen values play a crucial role in discriminating shape and orientation of structures.

http://www.image.med.osaka-u.ac.jp/member/yoshi/paper/linefilter.pdf

**References:**

"3D Multi-scale line filter for segmentation and visualization of

curvilinear structures in medical images", Yoshinobu Sato, Shin Nakajima, Hideki Atsumi, Thomas Koller, Guido Gerig, Shigeyuki Yoshida, Ron Kikinis.

**See** HessianRecursiveGaussianImageFilter

**See** SymmetricEigenAnalysisImageFilter

**See** SymmetricSecondRankTensor

See itk::Hessian3DToVesselnessMeasureImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage** = *Image*<SymmetricSecondRankTensor<**typename** *NumericTraits*
**class HessianRecursiveGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Computes the Hessian matrix of an image by convolution with the Second and Cross derivatives of a Gaussian.

This filter is implemented using the recursive gaussian filters

See itk::HessianRecursiveGaussianImageFilter for additional documentation.

### Sharpen Image

### Synopsis

Sharpen an image.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkLaplacianSharpeningImageFilter.h"
#include "itkSubtractImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using FloatImageType = itk::Image<float, 2>;

  using readerType = itk::ImageFileReader<FloatImageType>;
  readerType::Pointer reader = readerType::New();
  reader->SetFileName(inputFilename);

  using LaplacianSharpeningImageFilterType = itk::LaplacianSharpeningImageFilter
↪<FloatImageType, FloatImageType>;
  LaplacianSharpeningImageFilterType::Pointer laplacianSharpeningImageFilter =
    LaplacianSharpeningImageFilterType::New();
  laplacianSharpeningImageFilter->SetInput(reader->GetOutput());

  using SubtractType = itk::SubtractImageFilter<FloatImageType>;
  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(laplacianSharpeningImageFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  std::stringstream desc;
  desc << "LaplacianSharpeningImageFilter";
  viewer.AddImage(laplacianSharpeningImageFilter->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - LaplacianSharpening";
  viewer.AddImage(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LaplacianSharpeningImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
This filter sharpens an image using a Laplacian. LaplacianSharpening highlights regions of rapid intensity change and therefore highlights or enhances the edges. The result is an image that appears more in focus.

**The LaplacianSharpening at each pixel location is computed by** convolution with the itk::LaplacianOperator.

**Inputs and Outputs** The input to this filter is a scalar-valued itk::Image of arbitrary dimension. The output is a scalar-valued itk::Image.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**See** LaplacianOperator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Sharpen Image

See itk::LaplacianSharpeningImageFilter for additional documentation.

### Sobel Edge Detection Image Filter

### Synopsis

Apply SobelEdgeDetectionImageFilter to an image

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Sobel Edge Detection Image Filter.")
```

(continues on next page)

Fig. 155: Input image

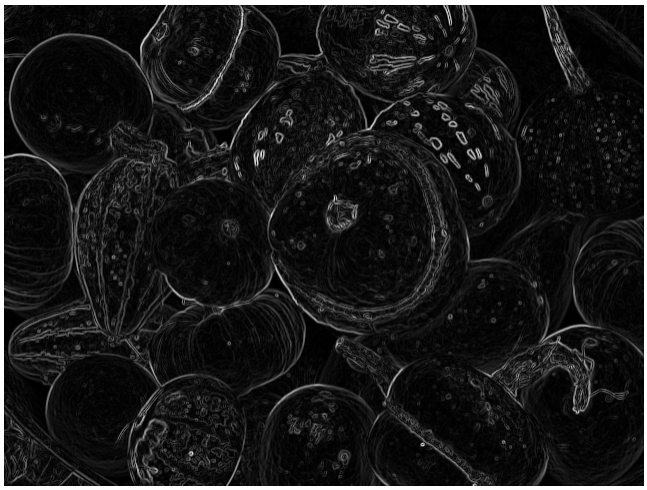


Fig. 156: Rescaled Output image (values are in [0, 255])

(continued from previous page)

```
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

input_image = itk.imread(args.input_image, pixel_type=itk.F)

output_image = itk.sobel_edge_detection_image_filter(input_image)

itk.imwrite(output_image, args.output_image)
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSobelEdgeDetectionImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using OutputPixelType = float;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::SobelEdgeDetectionImageFilter<InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
```

(continues on next page)

(continued from previous page)

```
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SobelEdgeDetectionImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
A 2D or 3D edge detection using the Sobel operator.

This filter uses the Sobel operator to calculate the image gradient and then finds the magnitude of this gradient vector. The Sobel gradient magnitude (square-root sum of squares) is an indication of edge strength.

**See** ImageToImageFilter

**See** SobelOperator

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Sobel Edge Detection Image Filter

See itk::SobelEdgeDetectionImageFilter for additional documentation.

### Zero-crossing Based Edge Decor

### Synopsis

A zero-crossing based edge detecor

### Results

### Code

### C++

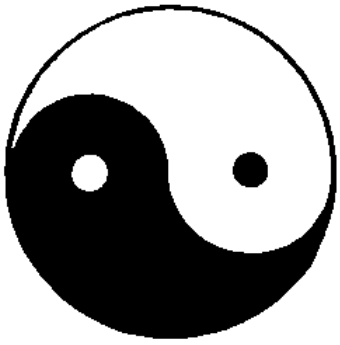


Fig. 157: Input image.

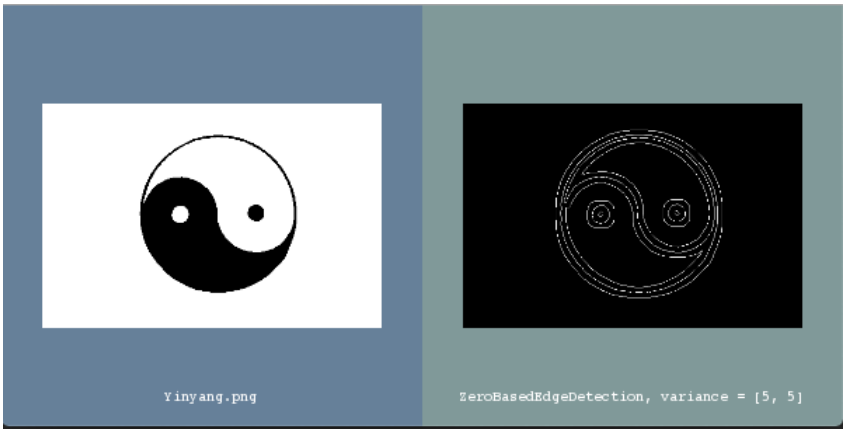



Fig. 158: Output In VTK Window

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkZeroCrossingBasedEdgeDetectionImageFilter.h"

#include "itksys/SystemTools.hxx"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  double var = 5.0;
  if (argc > 2)
  {
    var = std::stod(argv[2]);
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using FloatImageType = itk::Image<float, 2>;
  using ReaderType = itk::ImageFileReader<FloatImageType>;

  using FilterType = itk::ZeroCrossingBasedEdgeDetectionImageFilter<FloatImageType,␣
↪FloatImageType>;

  // Create and setup a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a derivative filter
  FilterType::Pointer edgeDetector = FilterType::New();
  edgeDetector->SetInput(reader->GetOutput());
  FilterType::ArrayType variance;
  variance.Fill(var);
  edgeDetector->SetVariance(variance);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFilename));

  std::stringstream desc;
  desc << "ZeroBasedEdgeDetection, variance = " << edgeDetector->GetVariance();
  viewer.AddImage(edgeDetector->GetOutput(), true, desc.str());
```

(continues on next page)

(continued from previous page)

```
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ZeroCrossingBasedEdgeDetectionImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImag*

This filter implements a zero-crossing based edge detecor.

The zero-crossing based edge detector looks for pixels in the Laplacian of an image where the value of the Laplacian passes through zero points where the Laplacian changes sign. Such points often occur at "edges" in images i.e. points where the intensity of the image changes rapidly, but they also occur at places that are not as easy to associate with edges. It is best to think of the zero crossing detector as some sort of feature detector rather than as a specific edge detector.

Zero crossings always lie on closed contours and so the output from the zero crossing detector is usually a binary image with single pixel thickness lines showing the positions of the zero crossing points.

In this implementation, the input image is first smoothed with a Gaussian filter, then the LaplacianImageFilter is applied to smoothed image. Finally the zero-crossing of the Laplacian of the smoothed image is detected. The output is a binary image.

**Inputs and Outputs** The input to the filter should be a scalar, itk::Image of arbitrary dimension. The output image is a binary, labeled image. See itkZeroCrossingImageFilter for more information on requirements of the data type of the output.

To use this filter, first set the parameters (variance and maximum error) needed by the embedded DiscreteGaussianImageFilter, i.e. See DiscreteGaussianImageFilter for information about these parameters. Optionally, you may also set foreground and background values for the zero-crossing filter. The default label values are Zero for the background and One for the foreground, as defined in NumericTraits for the data type of the output image.

**See** DiscreteGaussianImageFilter

**See** LaplacianImageFilter

**See** ZeroCrossingImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Zero-crossing Based Edge Decor

See itk::ZeroCrossingBasedEdgeDetectionImageFilter for additional documentation.

### 3.4.13 ImageFilterBase

#### Apply Kernel to Every Pixel

#### Synopsis

Apply a kernel to every pixel in an image.

#### Results

Fig. 159: input.png

Fig. 160: output.png

#### Code

#### Python

```
#!/usr/bin/env python

import sys
import itk

dimension = 2
float_image_type = itk.Image[itk.F, dimension]
unsigned_char_image_type = itk.Image[itk.UC, dimension]

start = itk.Index[dimension]()
start.Fill(0)

size = itk.Size[dimension]()
size.Fill(100)

region = itk.ImageRegion[dimension]()
region.SetIndex(start)
region.SetSize(size)

image = float_image_type.New(Regions=region)
image.Allocate()
image.FillBuffer(0)
```

(continues on next page)

(continued from previous page)

```
image[20:80, 20:80] = 15

input_image = itk.rescale_intensity_image_filter(
    image,
    output_minimum=0,
    output_maximum=255,
    ttype=(float_image_type, unsigned_char_image_type),
)

itk.imwrite(input_image, "inputPython.png")

sobel_operator = itk.SobelOperator[itk.F, dimension]()
radius = itk.Size[dimension]()
radius.Fill(1)  # a radius of 1x1 creates a 3x3 operator
sobel_operator.SetDirection(0)  # Create the operator for the X axis derivative
sobel_operator.CreateToRadius(radius)

image = itk.neighborhood_operator_image_filter(image, operator=sobel_operator)

output_image = itk.rescale_intensity_image_filter(
    image,
    output_minimum=0,
    output_maximum=255,
    ttype=(float_image_type, unsigned_char_image_type),
)

itk.imwrite(output_image, "outputPython.png")
```

### C++

```
#include "itkImage.h"
#include "itkNeighborhoodOperatorImageFilter.h"
#include "itkSobelOperator.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"

using FloatImageType = itk::Image<float, 2>;

void
CreateImage(FloatImageType::Pointer image);
void
CastRescaleAndWrite(FloatImageType::Pointer image, const std::string & filename);

int
main(int, char *[])
{
  FloatImageType::Pointer image = FloatImageType::New();
  CreateImage(image);
  CastRescaleAndWrite(image, "input.png");

  using SobelOperatorType = itk::SobelOperator<float, 2>;
  SobelOperatorType sobelOperator;
  itk::Size<2>      radius;
```

(continues on next page)

(continued from previous page)

```
  radius.Fill(1);                // a radius of 1x1 creates a 3x3 operator
  sobelOperator.SetDirection(0); // Create the operator for the X axis derivative
  sobelOperator.CreateToRadius(radius);

  using NeighborhoodOperatorImageFilterType = itk::NeighborhoodOperatorImageFilter
↪<FloatImageType, FloatImageType>;
  NeighborhoodOperatorImageFilterType::Pointer filter =␣
↪NeighborhoodOperatorImageFilterType::New();
  filter->SetOperator(sobelOperator);
  filter->SetInput(image);
  filter->Update();

  CastRescaleAndWrite(filter->GetOutput(), "output.png");

  return EXIT_SUCCESS;
}

void
CreateImage(FloatImageType::Pointer image)
{
  FloatImageType::IndexType start;
  start.Fill(0);

  FloatImageType::SizeType size;
  size.Fill(100);

  FloatImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 80; c++)
    {
      FloatImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}

void
CastRescaleAndWrite(FloatImageType::Pointer image, const std::string & filename)
{
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;
  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(image);
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(filename);
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TOperatorValueType** = **typename** *TOutputImage*: **class NeighborhoodOperatorImageFilter** : **public** itk::*ImageToImageFilter<TInputImage*, *TOutputImage*>

Applies a single NeighborhoodOperator to an image region.

This filter calculates successive inner products between a single NeighborhoodOperator and a NeighborhoodIterator, which is swept across every pixel in an image region. For operators that are symmetric across their axes, the result is a fast convolution with the image region. Apply the mirror()'d operator for non-symmetric NeighborhoodOperators.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Kernel To Every Pixel

Subclassed by itk::MaskNeighborhoodOperatorImageFilter< TInputImage, TMaskImage, TOutputImage, TOperatorValueType >, itk::NormalizedCorrelationImageFilter< TInputImage, TMaskImage, TOutputImage, TOperatorValueType >

See itk::NeighborhoodOperatorImageFilter for additional documentation.

### Apply Kernel to Every Pixel in Non-Zero Image

### Synopsis

Apply a kernel to every pixel in an image that is non-zero in a mask.

### Results

Fig. 161: input.png

Fig. 162: mask.png

Fig. 163: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkMaskImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkMaskNeighborhoodOperatorImageFilter.h"
#include "itkSobelOperator.h"

namespace
{
using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;
} // namespace

static void
CreateImage(UnsignedCharImageType::Pointer image);
static void
CreateHalfMask(UnsignedCharImageType::Pointer image, UnsignedCharImageType::Pointer␣
↪mask);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
```

(continues on next page)

(continued from previous page)

```
  CreateImage(image);

  UnsignedCharImageType::Pointer mask = UnsignedCharImageType::New();
  CreateHalfMask(image, mask);

  using SobelOperatorType = itk::SobelOperator<float, 2>;
  SobelOperatorType sobelOperator;
  itk::Size<2>      radius;
  radius.Fill(1);                // a radius of 1x1 creates a 3x3 operator
  sobelOperator.SetDirection(0); // Create the operator for the X axis derivative
  sobelOperator.CreateToRadius(radius);

  // Visualize mask image
  using MaskNeighborhoodOperatorImageFilterType =
    itk::MaskNeighborhoodOperatorImageFilter<UnsignedCharImageType,␣
↪UnsignedCharImageType, FloatImageType, float>;
  MaskNeighborhoodOperatorImageFilterType::Pointer␣
↪maskNeighborhoodOperatorImageFilter =
    MaskNeighborhoodOperatorImageFilterType::New();
  maskNeighborhoodOperatorImageFilter->SetInput(image);
  maskNeighborhoodOperatorImageFilter->SetMaskImage(mask);
  maskNeighborhoodOperatorImageFilter->SetOperator(sobelOperator);
  maskNeighborhoodOperatorImageFilter->Update();

  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(maskNeighborhoodOperatorImageFilter->GetOutput());
  rescaleFilter->Update();

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}


void
CreateHalfMask(UnsignedCharImageType::Pointer image, UnsignedCharImageType::Pointer␣
↪mask)
{
  itk::ImageRegion<2> region = image->GetLargestPossibleRegion();

  mask->SetRegions(region);
  mask->Allocate();
  mask->FillBuffer(0);

  itk::Size<2> regionSize = region.GetSize();

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(mask, region);

  // Make the left half of the mask white and the right half black
  while (!imageIterator.IsAtEnd())
  {
```

(continues on next page)

(continued from previous page)

```
    if (static_cast<unsigned int>(imageIterator.GetIndex()[0]) > regionSize[0] / 2)
    {
      imageIterator.Set(0);
    }
    else
    {
      imageIterator.Set(1);
    }

    ++imageIterator;
  }

  using RescaleFilterType = itk::RescaleIntensityImageFilter<UnsignedCharImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(mask);
  rescaleFilter->Update();

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("mask.png");
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 30; r < 70; r++)
  {
    for (unsigned int c = 30; c < 70; c++)
    {
      FloatImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage**, typename **TOperatorValueType** = **t**
**class MaskNeighborhoodOperatorImageFilter** : **public** itk::*NeighborhoodOperatorImageFilter<TInputImage*, *TOutpu*

Applies a single NeighborhoodOperator to an image, processing only those pixels that are under a mask.

This filter calculates successive inner products between a single NeighborhoodOperator and a NeighborhoodIterator, which is swept across every pixel that is set in the input mask. If no mask is given, this filter is equivalent to its superclass. Output pixels that are outside of the mask will be set to DefaultValue if UseDefaultValue is true (default). Otherwise, they will be set to the value of the input pixel.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodOperatorImageFilter

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Kernel To Every Pixel In Non-Zero Image

See itk::MaskNeighborhoodOperatorImageFilter for additional documentation.

### Cast an Image to Another Type

### Synopsis

Cast an Image to another type

### Results

Fig. 164: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Cast An Image To Another Type.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

Dimension = 2
InputPixelType = itk.F
OutputPixelType = itk.UC

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

reader = itk.ImageFileReader[InputImageType].New()
reader.SetFileName(args.input_image)

rescaler = itk.RescaleIntensityImageFilter[InputImageType, InputImageType].New()
rescaler.SetInput(reader.GetOutput())
rescaler.SetOutputMinimum(0)
outputPixelTypeMaximum = itk.NumericTraits[OutputPixelType].max()
rescaler.SetOutputMaximum(outputPixelTypeMaximum)

castImageFilter = itk.CastImageFilter[InputImageType, OutputImageType].New()
castImageFilter.SetInput(rescaler.GetOutput())

writer = itk.ImageFileWriter[OutputImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(castImageFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkCastImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  const char * inputImage = argv[1];
  const char * outputImage = argv[2];

  using InputPixelType = float;
  using OutputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using RescaleType = itk::RescaleIntensityImageFilter<InputImageType, InputImageType>
↪;
  RescaleType::Pointer rescale = RescaleType::New();
  rescale->SetInput(reader->GetOutput());
  rescale->SetOutputMinimum(0);
  rescale->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  using FilterType = itk::CastImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(rescale->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputImage);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CastImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Casts input pixels to output pixel type.

This filter is templated over the input image type and the output image type.

A typical use is to cast a

```
itk::Image<type1, dim>
```

to a

```
itk::Image<type2, dim>
```

This filter can also be used to cast a

```
itk::VectorImage<type1, dim>
```

to a

```
itk::VectorImage<type2, dim>
```

If you need to perform a dimensionaly reduction, you may want to use the ExtractImageFilter instead of the CastImageFilter.

**See** UnaryFunctorImageFilter

**See** ExtractImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cast An Image To Another Type

Subclassed by itk::GPUImageToImageFilter< TInputImage, TOutputImage, CastImageFilter< TInputImage, TOutputImage > >

See itk::CastImageFilter for additional documentation.

### Compute Local Noise in Image

### Synopsis

Compute the local noise in an image.

### Results

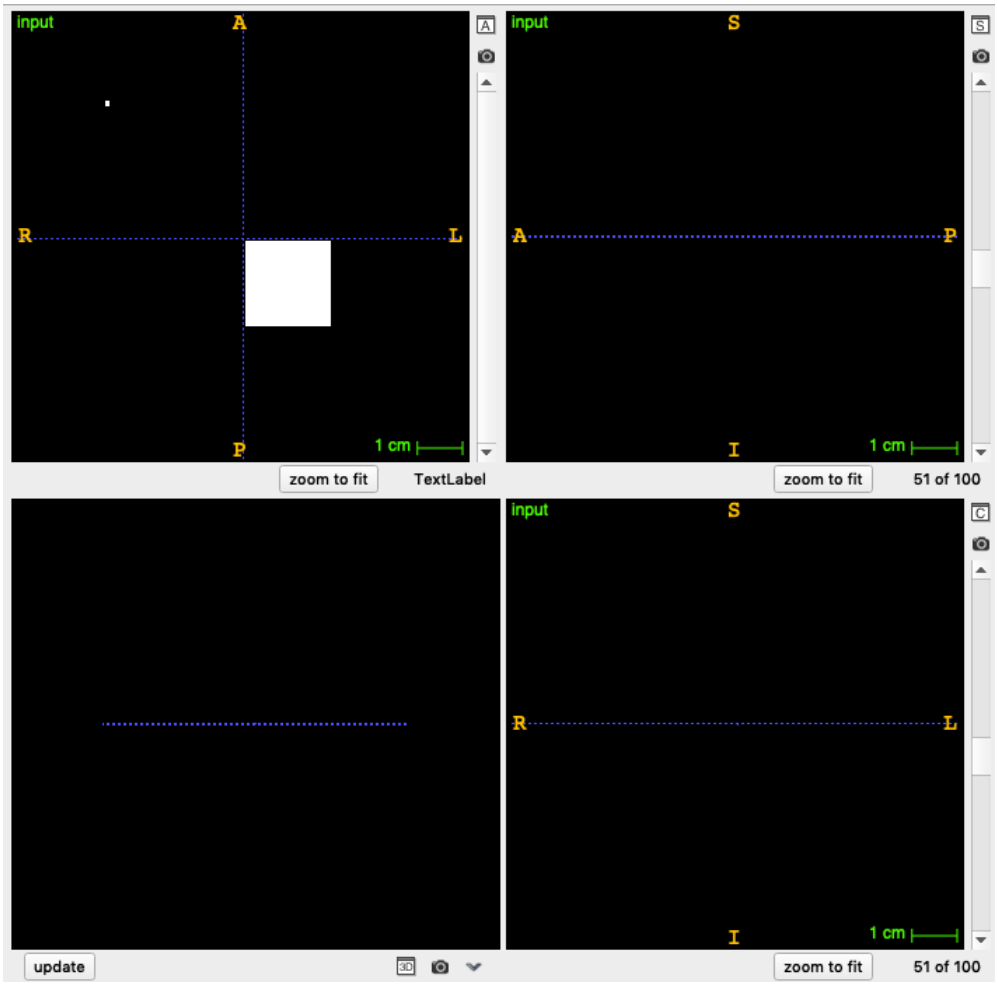



Fig. 165: input.mhd

### Code

### C++

```
#include "itkImage.h"
#include "itkNoiseImageFilter.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<float, 2>;

void
CreateImage(ImageType::Pointer image);

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
```

(continues on next page)

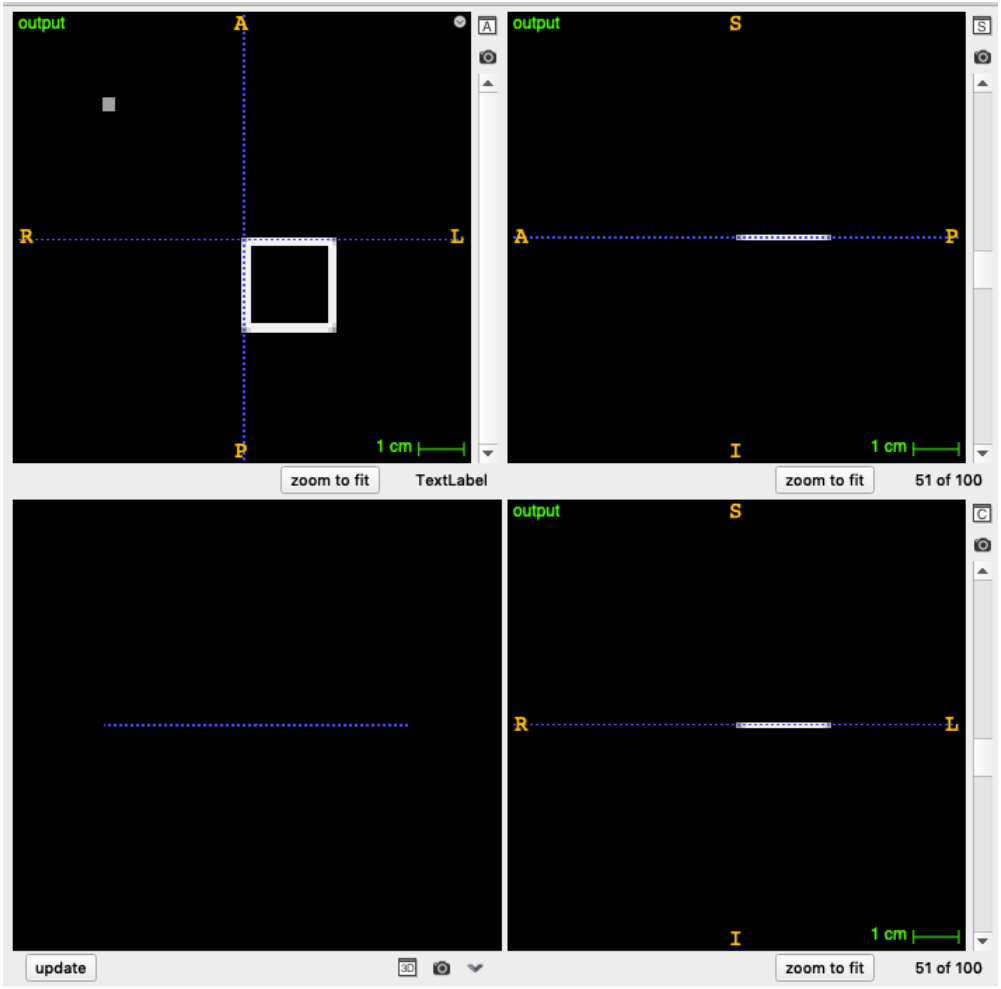



Fig. 166: output.mhd

(continued from previous page)

```
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using NoiseImageFilterType = itk::NoiseImageFilter<ImageType, ImageType>;
  NoiseImageFilterType::Pointer noiseImageFilter = NoiseImageFilterType::New();
  noiseImageFilter->SetInput(image);
  noiseImageFilter->SetRadius(1);
  noiseImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.mhd");
  writer->SetInput(noiseImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image that is mostly constant but has some different kinds of objects.
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);

  // Create a black image
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a white square
  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70 &&␣
↪imageIterator.GetIndex()[1] > 50 &&
        imageIterator.GetIndex()[1] < 70)
    {
      imageIterator.Set(255);
    }
    ++imageIterator;
  }

  // Create a rogue white pixel
  ImageType::IndexType pixel;
  pixel.Fill(20);
  image->SetPixel(pixel, 255);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.mhd");
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class NoiseImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>
Calculate the local noise in an image.

Computes an image where a given pixel is the standard deviation of the pixels in a neighborhood about the corresponding input pixel. This serves as an estimate of the local noise (or texture) in an image. Currently, this noise estimate assume a piecewise constant image. This filter should be extended to fitting a (hyper) plane to the neighborhood and calculating the standard deviation of the residuals to this (hyper) plane.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Local Noise In Image

See itk::NoiseImageFilter for additional documentation.

### Custom Operation to Corresponding Pixels in Two Images

### Synopsis

Apply a predefined operation to corresponding pixels in two images.

### Results

Output:

```
pixel1 was = 2
pixel2 was = 5
output is = 9
```

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkVector.h"
#include "itkVariableLengthVector.h"
#include "itkRigid2DTransform.h"
#include "itkBinaryFunctorImageFilter.h"

using ImageType = itk::Image<float, 2>;
static void
CreateImage(ImageType::Pointer image);

namespace Functor
{
template <class TPixel>
class MySquaredDifference
{
public:
  MySquaredDifference() = default;
  ~MySquaredDifference() = default;
  bool
  operator!=(const MySquaredDifference &) const
  {
    return false;
  }

  bool
  operator==(const MySquaredDifference & other) const
  {
    return !(*this != other);
  }

  inline TPixel
  operator()(const TPixel & A, const TPixel & B) const
  {
    const auto   dA = static_cast<double>(A);
    const auto   dB = static_cast<double>(B);
    const double diff = dA - dB;

    return static_cast<TPixel>(diff * diff);
  }
};
} // namespace Functor

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage(image1);
  image1->FillBuffer(2);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage(image2);
  image2->FillBuffer(5);
```

(continues on next page)

(continued from previous page)

```
  using FilterType =
    itk::BinaryFunctorImageFilter<ImageType, ImageType, ImageType, Functor::
↪MySquaredDifference<ImageType::PixelType>>;

  FilterType::Pointer filter = FilterType::New();
  filter->SetInput1(image1);
  filter->SetInput2(image2);
  filter->Update();

  itk::Index<2> pixelIndex;
  pixelIndex.Fill(0);

  ImageType::PixelType input1PixelValue = image1->GetPixel(pixelIndex);
  ImageType::PixelType input2PixelValue = image2->GetPixel(pixelIndex);
  ImageType::PixelType outputPixelValue = filter->GetOutput()->GetPixel(pixelIndex);

  std::cout << "pixel1 was = " << input1PixelValue << std::endl;
  std::cout << "pixel2 was = " << input2PixelValue << std::endl;
  std::cout << "output is = " << outputPixelValue << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
}
```

## Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**, typename **TOutputImage**, typename **TFunction**>
**class BinaryFunctorImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage1*, *TOutputImage*>

Implements pixel-wise generic operation of two images, or of an image and a constant.

This class is parameterized over the types of the two input images and the type of the output image. It is also parameterized by the operation to be applied. A Functor style is used.

The constant must be of the same type than the pixel type of the corresponding image. It is wrapped in a SimpleDataObjectDecorator so it can be updated through the pipeline. The SetConstant() and GetConstant() methods are provided as shortcuts to set or get the constant value without manipulating the decorator.

**See** BinaryGeneratorImagFilter

**See** UnaryFunctorImageFilter TernaryFunctorImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Predefined Operation To Corresponding Pixels In Two Images
- Custom Operation To Corresponding Pixels In Two Images

See itk::BinaryFunctorImageFilter for additional documentation.

### Predefined Operation to Corresponding Pixels in Two Images

### Synopsis

Apply a predefined operation to corresponding pixels in two images.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkBinaryFunctorImageFilter.h"
#include "itkAndImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage(image1);
  image1->FillBuffer(2);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage(image2);
  image2->FillBuffer(5);

  using FilterType =
    itk::BinaryFunctorImageFilter<ImageType, ImageType, ImageType, itk::Functor::AND
↪<ImageType::PixelType>>;

  FilterType::Pointer filter = FilterType::New();
  filter->SetInput1(image1);
  filter->SetInput2(image2);
  filter->Update();

  itk::Index<2> pixelIndex;
  pixelIndex.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  ImageType::PixelType input1PixelValue = image1->GetPixel(pixelIndex);
  ImageType::PixelType input2PixelValue = image2->GetPixel(pixelIndex);
  ImageType::PixelType outputPixelValue = filter->GetOutput()->GetPixel(pixelIndex);

  std::cout << "pixel1 was = " << input1PixelValue << std::endl;
  std::cout << "pixel2 was = " << input2PixelValue << std::endl;
  std::cout << "output is = " << outputPixelValue << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(10);

  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**, typename **TOutputImage**, typename **TFunction**>
**class BinaryFunctorImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage1*, *TOutputImage*>

Implements pixel-wise generic operation of two images, or of an image and a constant.

This class is parameterized over the types of the two input images and the type of the output image. It is also parameterized by the operation to be applied. A Functor style is used.

The constant must be of the same type than the pixel type of the corresponding image. It is wrapped in a SimpleDataObjectDecorator so it can be updated through the pipeline. The SetConstant() and GetConstant() methods are provided as shortcuts to set or get the constant value without manipulating the decorator.

**See** BinaryGeneratorImagFilter

**See** UnaryFunctorImageFilter TernaryFunctorImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Predefined Operation To Corresponding Pixels In Two Images
- Custom Operation To Corresponding Pixels In Two Images

See itk::BinaryFunctorImageFilter for additional documentation.

### 3.4.14 ImageFusion

#### Color Boundaries of Labeled Regions

#### Synopsis

Color the boundaries of labeled regions in an image.

#### Results

Fig. 167: image.png

Fig. 168: output.png

#### Code

#### C++

```
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelMapContourOverlayImageFilter.h"
#include "itkRGBPixel.h"

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
}

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);
```

(continues on next page)

(continued from previous page)

```
  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType>;

  using LabelMapContourOverlayImageFilterType =
    itk::LabelMapContourOverlayImageFilter<BinaryImageToLabelMapFilterType::
↪OutputImageType, ImageType, RGBImageType>;
  LabelMapContourOverlayImageFilterType::Pointer labelMapContourOverlayImageFilter =
    LabelMapContourOverlayImageFilterType::New();
  labelMapContourOverlayImageFilter->SetInput(binaryImageToLabelMapFilter->
↪GetOutput());
  labelMapContourOverlayImageFilter->SetFeatureImage(image);
  labelMapContourOverlayImageFilter->SetOpacity(.5);
  labelMapContourOverlayImageFilter->Update();

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(labelMapContourOverlayImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make two squares
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 20) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 20))
    {
      imageIterator.Set(255);
    }
```

(continues on next page)

(continued from previous page)

```
    if ((imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70) &&
        (imageIterator.GetIndex()[1] > 50 && imageIterator.GetIndex()[1] < 70))
    {
      imageIterator.Set(255);
    }
    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TLabelMap**, typename **TFeatureImage**, typename **TOutputImage** = *Image*<*RGBPixel*<**typename** *TFeatu*
**class LabelMapContourOverlayImageFilter** : **public** itk::LabelMapFilter<*TLabelMap*, *TOutputImage*>

Apply a colormap to the contours (outlines) of each object in a label map and superimpose it on top of the feature image.

The feature image is typically the image from which the labeling was produced. Use the SetInput function to set the LabelMap, and the SetFeatureImage function to set the feature image.

Apply a colormap to a label map and put it on top of the input image. The set of colors is a good selection of distinct colors. The opacity of the label map can be defined by the user. A background label produce a gray pixel with the same intensity than the input one.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelMapOverlayImageFilter, LabelOverlayImageFilter, LabelOverlayFunctor

**See** LabelMapToBinaryImageFilter, LabelMapToLabelImageFilter,

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Color Boundaries Of Labeled Regions

See itk::LabelMapContourOverlayImageFilter for additional documentation.

### Color Labeled Regions in Image

### Synopsis

Color labeled regions in an image.

### Results

Fig. 169: image.png

Fig. 170: output.png

### Code

### C++

```
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelMapOverlayImageFilter.h"
#include "itkRGBPixel.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType>;

  using LabelMapOverlayImageFilterType =
    itk::LabelMapOverlayImageFilter<BinaryImageToLabelMapFilterType::OutputImageType,␣
↪ImageType, RGBImageType>;
  LabelMapOverlayImageFilterType::Pointer labelMapOverlayImageFilter =␣
↪LabelMapOverlayImageFilterType::New();
  labelMapOverlayImageFilter->SetInput(binaryImageToLabelMapFilter->GetOutput());
  labelMapOverlayImageFilter->SetFeatureImage(image);
  labelMapOverlayImageFilter->SetOpacity(.5);
  labelMapOverlayImageFilter->Update();

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(labelMapOverlayImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make two squares
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 20) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 20))
    {
      imageIterator.Set(255);
    }

    if ((imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70) &&
        (imageIterator.GetIndex()[1] > 50 && imageIterator.GetIndex()[1] < 70))
    {
      imageIterator.Set(255);
    }
    ++imageIterator;
```

(continues on next page)

(continued from previous page)

```
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TLabelMap**, typename **TFeatureImage**, typename **TOutputImage** = *Image*<*RGBPixel*<**typename** *TFeatu*
**class LabelMapOverlayImageFilter** : **public** itk::LabelMapFilter<*TLabelMap*, *TOutputImage*>

Apply a colormap to a label map and superimpose it on an image.

Apply a colormap to a label map and put it on top of the feature image. The feature image is typically the image from which the labeling was produced. Use the SetInput function to set the LabelMap, and the SetFeatureImage function to set the feature image.

The set of colors is a good selection of distinct colors. The opacity of the label map can be defined by the user. A background label produce a gray pixel with the same intensity than the input one.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelOverlayImageFilter, LabelOverlayFunctor

**See** LabelMapToRGBImageFilter, LabelMapToBinaryImageFilter, LabelMapToLabelImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Color Labeled Regions In Image

See itk::LabelMapOverlayImageFilter for additional documentation.

### Overlay Label Map on Image

### Synopsis

Overlay a LabelMap on an image.

### Results

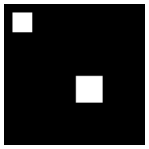


Fig. 171: image.png

Fig. 172: output.png

### Code

### C++

```
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelOverlayImageFilter.h"
#include "itkRGBPixel.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  using LabelMapToLabelImageFilterType =
    itk::LabelMapToLabelImageFilter<BinaryImageToLabelMapFilterType::OutputImageType,␣
↪ImageType>;
  LabelMapToLabelImageFilterType::Pointer labelMapToLabelImageFilter =␣
↪LabelMapToLabelImageFilterType::New();
  labelMapToLabelImageFilter->SetInput(binaryImageToLabelMapFilter->GetOutput());
  labelMapToLabelImageFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType>;

  using LabelOverlayImageFilterType = itk::LabelOverlayImageFilter<ImageType,␣
↪ImageType, RGBImageType>;
  LabelOverlayImageFilterType::Pointer labelOverlayImageFilter =␣
↪LabelOverlayImageFilterType::New();
  labelOverlayImageFilter->SetInput(image);
  labelOverlayImageFilter->SetLabelImage(labelMapToLabelImageFilter->GetOutput());
  labelOverlayImageFilter->SetOpacity(.5);
  labelOverlayImageFilter->Update();

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(labelOverlayImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make two squares
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 20) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 20))
    {
      imageIterator.Set(255);
    }

    if ((imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70) &&
        (imageIterator.GetIndex()[1] > 50 && imageIterator.GetIndex()[1] < 70))
    {
      imageIterator.Set(255);
    }
    ++imageIterator;
  }
```

(continues on next page)

(continued from previous page)

```
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TLabelImage**, typename **TOutputImage**>
**class LabelOverlayImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage*, *TLabelImage*, *TOutputImage*>

Apply a colormap to a label image and put it on top of the input image.

Apply a colormap to a label image and put it on top of the input image. The set of colors is a good selection of distinct colors. The opacity of the label image can be defined by the user. The user can also choose if the want to use a background and which label value is the background. A background label produce a gray pixel with the same intensity than the input one.

This class was contributed to the Insight Journal https://www.insight-journal.org/browse/publication/79

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelToRGBImageFilter

**See** LabelMapOverlayImageFilter, LabelOverlayFunctor

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Overlay Label Map On Image

See itk::LabelOverlayImageFilter for additional documentation.

### Overlay Label Map on Top of an Image

### Synopsis

Apply a colormap to a label map and superimpose it on an image

### Results

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
```

(continues on next page)

Fig. 173: Input image

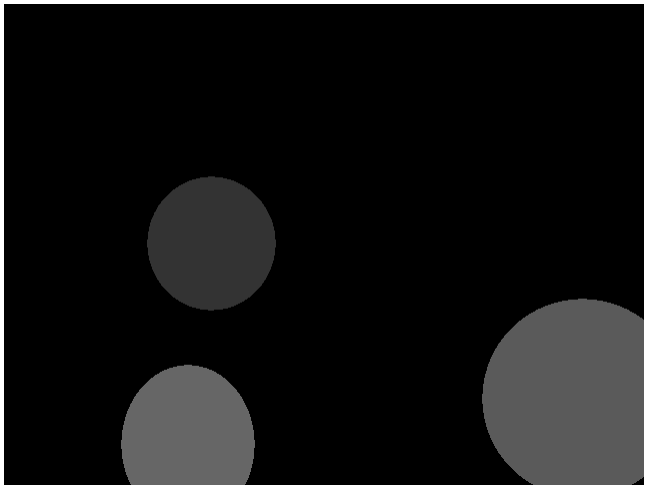


Fig. 174: Label image

Fig. 175: Output image

(continued from previous page)

```
import argparse

parser = argparse.ArgumentParser(description="Overlay Label Map On Top Of An Image.")
parser.add_argument("input_image")
parser.add_argument("label_map")
parser.add_argument("output_image")
args = parser.parse_args()

PixelType = itk.ctype("unsigned char")
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]


reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

labelReader = itk.ImageFileReader[ImageType].New()
labelReader.SetFileName(args.label_map)


LabelType = itk.ctype("unsigned long")
LabelObjectType = itk.StatisticsLabelObject[LabelType, Dimension]
LabelMapType = itk.LabelMap[LabelObjectType]

converter = itk.LabelImageToLabelMapFilter[ImageType, LabelMapType].New()
converter.SetInput(labelReader)

RGBImageType = itk.Image[itk.RGBPixel[PixelType], Dimension]
overlayFilter = itk.LabelMapOverlayImageFilter[
    LabelMapType, ImageType, RGBImageType
].New()
overlayFilter.SetInput(converter.GetOutput())
overlayFilter.SetFeatureImage(reader.GetOutput())
overlayFilter.SetOpacity(0.5)


itk.imwrite(overlayFilter.GetOutput(), args.output_image)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelObject.h"
#include "itkLabelMap.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapOverlayImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
```

(continues on next page)

(continued from previous page)

```
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <LabelMap> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * labelFileName = argv[2];
  const char * outputFileName = argv[3];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;


  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  ReaderType::Pointer labelReader = ReaderType::New();
  labelReader->SetFileName(labelFileName);


  using LabelType = PixelType;
  using LabelObjectType = itk::LabelObject<LabelType, Dimension>;
  using LabelMapType = itk::LabelMap<LabelObjectType>;

  using ConverterType = itk::LabelImageToLabelMapFilter<ImageType, LabelMapType>;
  ConverterType::Pointer converter = ConverterType::New();
  converter->SetInput(labelReader->GetOutput());

  using FilterType = itk::LabelMapOverlayImageFilter<LabelMapType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(converter->GetOutput());
  filter->SetFeatureImage(reader->GetOutput());
  filter->SetOpacity(0.5);


  using WriterType = itk::ImageFileWriter<FilterType::OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TLabelMap**, typename **TFeatureImage**, typename **TOutputImage** = *Image*<*RGBPixel*<**typename** *TFeatu*
**class LabelMapOverlayImageFilter** : **public** itk::LabelMapFilter<*TLabelMap*, *TOutputImage*>

Apply a colormap to a label map and superimpose it on an image.

Apply a colormap to a label map and put it on top of the feature image. The feature image is typically the image from which the labeling was produced. Use the SetInput function to set the LabelMap, and the SetFeatureImage function to set the feature image.

The set of colors is a good selection of distinct colors. The opacity of the label map can be defined by the user. A background label produce a gray pixel with the same intensity than the input one.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelOverlayImageFilter, LabelOverlayFunctor

**See** LabelMapToRGBImageFilter, LabelMapToBinaryImageFilter, LabelMapToLabelImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Color Labeled Regions In Image

See itk::LabelMapOverlayImageFilter for additional documentation.

### 3.4.15 ImageGradient

#### Apply GradientRecursiveGaussianImageFilter

#### Synopsis

Computes the gradient of an image by convolution with the first derivative of a Gaussian.

The output of the GradientRecursiveGaussianImageFilter is composed of two imagescomponents (gradients along the X and Y directions). In this example, we store these components using an image with CovariantVector pixel type. Covariant vectors are the natural representation for gradients since they are the equivalent of normals to iso-values manifolds.

This example shows also how to extract the X and Y gradients as images, and how to compute the magnitude of the CovariantVector image.

Note that the covariant vector types were only added to the VectorIndexSelectionCastImageFilter Python wrapping in ITK 4.7.

### Results

Fig. 176: Input image

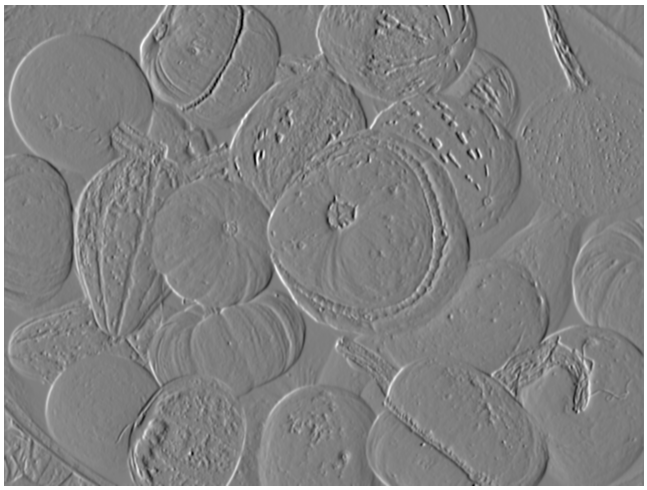


Fig. 177: Gradient along X direction

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.7.0"):
```

(continues on next page)

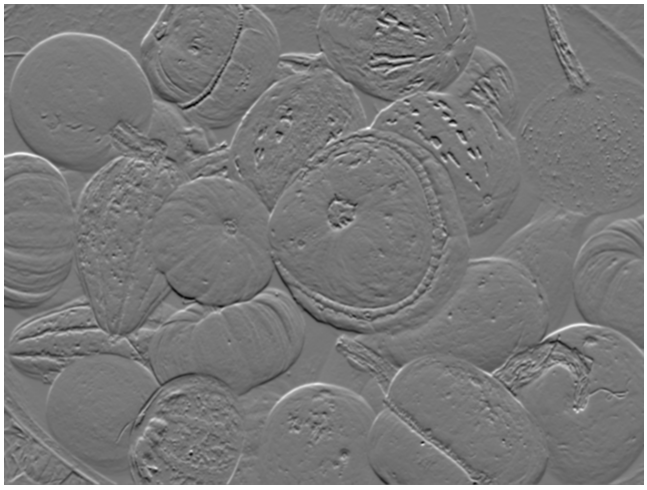


Fig. 178: Gradient along Y direction

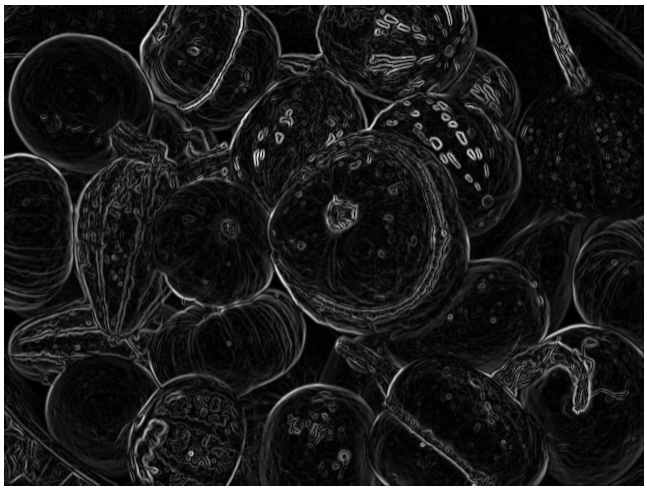


Fig. 179: The image of the magnitude of the vector containing the X and Y values.

(continued from previous page)

```
    print("ITK 4.7.0 is required (see example documentation).")
    sys.exit(1)

parser = argparse.ArgumentParser(description="Apply Gradient Recursive Gaussian.")
parser.add_argument("input_image")
parser.add_argument("output_image_x")
parser.add_argument("output_image_y")
parser.add_argument("output_image_magnitude")
args = parser.parse_args()

Dimension = 2

filenames = []
filenames.append(args.output_image_x)
filenames.append(args.output_image_y)

# Input and output are png files, use unsigned char
PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]
# Float type for GradientRecursiveGaussianImageFilter
FloatPixelType = itk.F
FloatImageType = itk.Image[FloatPixelType, Dimension]
# The output of GradientRecursiveGaussianImageFilter
# are images of the gradient along X and Y, so the type of
# the output is a covariant vector of dimension 2 (X, Y)
CovPixelType = itk.CovariantVector[FloatPixelType, Dimension]
CovImageType = itk.Image[CovPixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.GradientRecursiveGaussianImageFilter[ImageType, CovImageType]
gradientFilter = FilterType.New()
gradientFilter.SetInput(reader.GetOutput())

# Allows to select the X or Y output images
IndexSelectionType = itk.VectorIndexSelectionCastImageFilter[
    CovImageType, FloatImageType
]
indexSelectionFilter = IndexSelectionType.New()
indexSelectionFilter.SetInput(gradientFilter.GetOutput())

# Rescale for png output
RescalerType = itk.RescaleIntensityImageFilter[FloatImageType, ImageType]
rescaler = RescalerType.New()
rescaler.SetOutputMinimum(itk.NumericTraits[PixelType].min())
rescaler.SetOutputMaximum(itk.NumericTraits[PixelType].max())
rescaler.SetInput(indexSelectionFilter.GetOutput())

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetInput(rescaler.GetOutput())

# Write the X and Y images
for i in range(2):
    writer.SetFileName(filenames[i])
```

(continues on next page)

(continued from previous page)

```
    indexSelectionFilter.SetIndex(i)
    writer.Update()

# Compute the magnitude of the vector and output the image
MagnitudeType = itk.VectorMagnitudeImageFilter[CovImageType, FloatImageType]
magnitudeFilter = MagnitudeType.New()
magnitudeFilter.SetInput(gradientFilter.GetOutput())

# Rescale for png output
rescaler.SetInput(magnitudeFilter.GetOutput())

writer.SetFileName(args.output_image_magnitude)
writer.SetInput(rescaler.GetOutput())
writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkGradientRecursiveGaussianImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkVectorIndexSelectionCastImageFilter.h"
#include "itkVectorMagnitudeImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileNameX> <OutputFileNameY>
↪<OutputFileNameMagnitude>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileNameX = argv[2];
  const char * outputFileNameY = argv[3];
  const char * outputFileNameMagnitude = argv[4];

  const char * filenames[2];
  filenames[0] = outputFileNameX;
  filenames[1] = outputFileNameY;

  constexpr unsigned int Dimension = 2;

  // Input and output are png files, use unsigned char
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
  // Double type for GradientRecursiveGaussianImageFilter
  using DoublePixelType = double;
```

(continues on next page)

(continued from previous page)

```
  using DoubleImageType = itk::Image<DoublePixelType, Dimension>;
  // The output of GradientRecursiveGaussianImageFilter
  // are images of the gradient along X and Y, so the type of
  // the output is a covariant vector of dimension 2 (X, Y)
  using CovPixelType = itk::CovariantVector<DoublePixelType, Dimension>;
  using CovImageType = itk::Image<CovPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::GradientRecursiveGaussianImageFilter<ImageType,␣
↪CovImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  // Allows to select the X or Y output images
  using IndexSelectionType = itk::VectorIndexSelectionCastImageFilter<CovImageType,␣
↪DoubleImageType>;
  IndexSelectionType::Pointer indexSelectionFilter = IndexSelectionType::New();
  indexSelectionFilter->SetInput(filter->GetOutput());

  // Rescale for png output
  using RescalerType = itk::RescaleIntensityImageFilter<DoubleImageType, ImageType>;
  RescalerType::Pointer rescaler = RescalerType::New();
  rescaler->SetOutputMinimum(itk::NumericTraits<PixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<PixelType>::max());
  rescaler->SetInput(indexSelectionFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(rescaler->GetOutput());

  // Write the X and Y images
  for (int i = 0; i < 2; ++i)
  {

    writer->SetFileName(filenames[i]);
    indexSelectionFilter->SetIndex(i);

    try
    {
      writer->Update();
    }
    catch (itk::ExceptionObject & error)
    {
      std::cerr << "Error: " << error << std::endl;
      return EXIT_FAILURE;
    }
  }

  // Compute the magnitude of the vector and output the image
  using MagnitudeType = itk::VectorMagnitudeImageFilter<CovImageType, DoubleImageType>
↪;
  MagnitudeType::Pointer magnitudeFilter = MagnitudeType::New();
  magnitudeFilter->SetInput(filter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  // Rescale for png output
  rescaler->SetInput(magnitudeFilter->GetOutput());

  writer->SetFileName(outputFileNameMagnitude);
  writer->SetInput(rescaler->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<*CovariantVector*<**typename** *NumericTraits*<**typename**
**class GradientRecursiveGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Computes the gradient of an image by convolution with the first derivative of a Gaussian.

This filter is implemented using the recursive gaussian filters.

This filter supports both scalar and vector pixel types within the input image, including VectorImage type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply GradientRecursiveGaussianImageFilter on Image with Vector type
- Implementation Of Snakes

See itk::GradientRecursiveGaussianImageFilter for additional documentation.

### Apply GradientRecursiveGaussianImageFilter on Image With Vector type

### Synopsis

Computes the gradient of an image by convolution with the first derivative of a Gaussian.

The output of the GradientRecursiveGaussianImageFilter is composed of two images (gradient along the X and Y directions). In this example, the input is an image having two values per pixel (vector type). One value is coming from the inputed image (first image); the second is the inverse of the inputed image (second image).

The output of GradientRecursiveGaussianImageFilter will then have four values per pixel. These are in the following order: gradient along X axis (first image), gradient along Y axis (first image), gradient along X axis (second image), gradient along Y axis (second image).

Note that the Python example will only work with ITK 4.8.0 or greater. The ComoposeImagefilter could be used to cast from unsigned char to double type without explicitely using the CastImageFilter; but in Python this possibilty is

not allowed. This reduces the number of combinations that need to be wrapped, which helps reducing the size of the ITK binaries.

### Results

Fig. 180: Input image (first image)

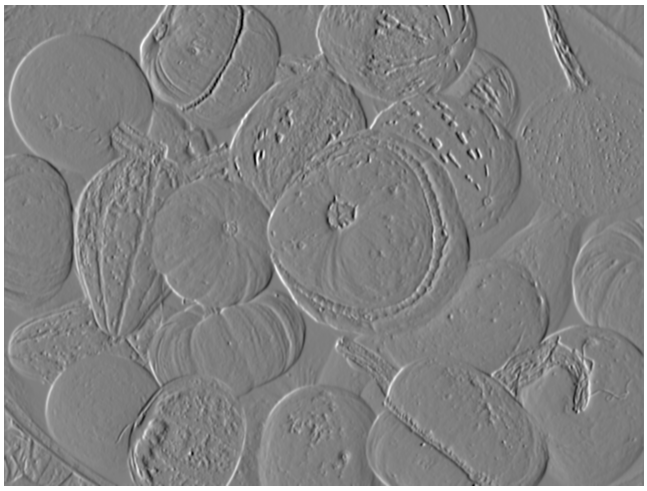


Fig. 181: Gradient along X direction (first image)

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
```

(continues on next page)

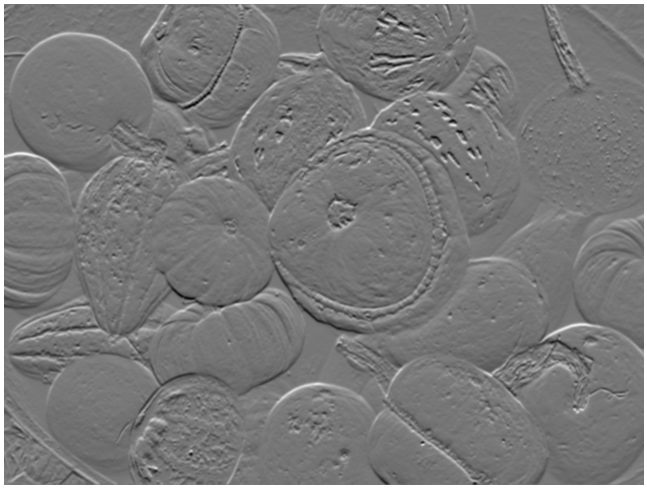


Fig. 182: Gradient along Y direction (first image)

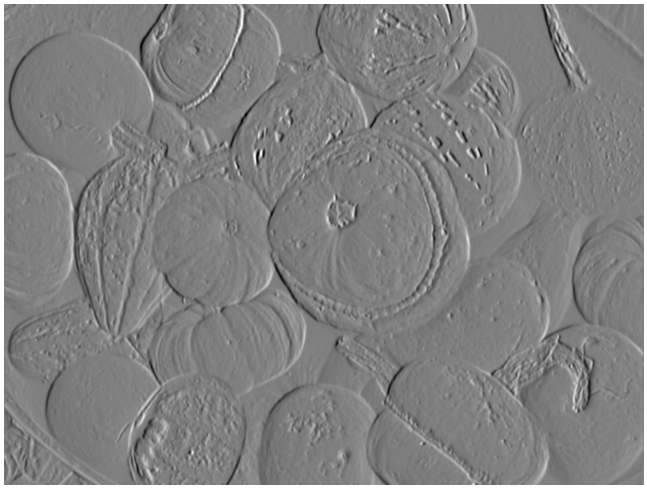


Fig. 183: Gradient along X direction (second image)

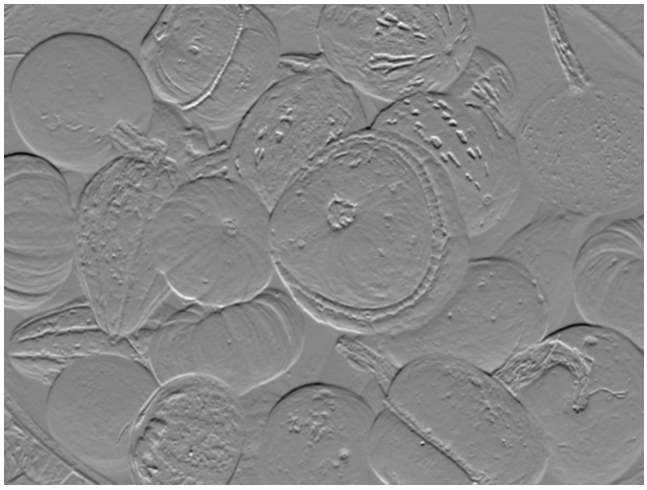


Fig. 184: Gradient along Y direction (second image)

(continued from previous page)

```
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.8.0"):
    print("ITK 4.8.0 is required (see example documentation).")
    sys.exit(1)

parser = argparse.ArgumentParser(
    description="Apply Gradient Recursive Gaussian With Vector Input."
)
parser.add_argument("input_image")
parser.add_argument("output_image_1x")
parser.add_argument("output_image_1y")
parser.add_argument("output_image_2x")
parser.add_argument("output_image_2y")
args = parser.parse_args()

filenames = [
    args.output_image_1x,
    args.output_image_1y,
    args.output_image_2x,
    args.output_image_2y,
]

ImageDimension = 2
VectorDimension = 2
CovDimension = 4

PixelType = itk.UC
ImageType = itk.Image[PixelType, ImageDimension]
FloatPixelType = itk.F
FloatImageType = itk.Image[FloatPixelType, ImageDimension]
VecPixelType = itk.Vector[FloatPixelType, VectorDimension]
VecImageType = itk.Image[VecPixelType, ImageDimension]
CovPixelType = itk.CovariantVector[FloatPixelType, CovDimension]
CovImageType = itk.Image[CovPixelType, ImageDimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

# Invert the input image
InvertType = itk.InvertIntensityImageFilter[ImageType, ImageType]
inverter = InvertType.New()
inverter.SetInput(reader.GetOutput())

# Cast the image to double type.
CasterType = itk.CastImageFilter[ImageType, FloatImageType]
caster = CasterType.New()
caster2 = CasterType.New()

# Create an image, were each pixel has 2 values: first value is the value
# coming from the input image, second value is coming from the inverted
# image
ComposeType = itk.ComposeImageFilter[FloatImageType, VecImageType]
composer = ComposeType.New()
```

(continues on next page)

(continued from previous page)

```
caster.SetInput(reader.GetOutput())
composer.SetInput(0, caster.GetOutput())
caster2.SetInput(inverter.GetOutput())
composer.SetInput(1, caster2.GetOutput())

# Apply the gradient filter on the two images, this will return and image
# with 4 values per pixel: two X and Y gradients
FilterType = itk.GradientRecursiveGaussianImageFilter[VecImageType, CovImageType]
gradientfilter = FilterType.New()
gradientfilter.SetInput(composer.GetOutput())

# Set up the filter to select each gradient
IndexSelectionType = itk.VectorIndexSelectionCastImageFilter[
    CovImageType, FloatImageType
]
indexSelectionFilter = IndexSelectionType.New()
indexSelectionFilter.SetInput(gradientfilter.GetOutput())

# Rescale for png output
RescalerType = itk.RescaleIntensityImageFilter[FloatImageType, ImageType]
rescaler = RescalerType.New()
rescaler.SetOutputMinimum(itk.NumericTraits[PixelType].min())
rescaler.SetOutputMaximum(itk.NumericTraits[PixelType].max())
rescaler.SetInput(indexSelectionFilter.GetOutput())

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetInput(rescaler.GetOutput())

for i in range(4):
    indexSelectionFilter.SetIndex(i)
    writer.SetFileName(filenames[i])
    writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkGradientRecursiveGaussianImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkVectorIndexSelectionCastImageFilter.h"
#include "itkVectorMagnitudeImageFilter.h"
#include "itkInvertIntensityImageFilter.h"
#include "itkComposeImageFilter.h"
#include "itkCastImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " <InputFileName> <OutputFileName1X> <OutputFileName1Y>
↪<OutputFileName2X> <OutputFileName2Y>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName1X = argv[2];
  const char * outputFileName1Y = argv[3];
  const char * outputFileName2X = argv[4];
  const char * outputFileName2Y = argv[5];

  const char * filenames[4];
  filenames[0] = outputFileName1X;
  filenames[1] = outputFileName1Y;
  filenames[2] = outputFileName2X;
  filenames[3] = outputFileName2Y;

  constexpr unsigned int ImageDimension = 2;
  constexpr unsigned int VectorDimension = 2;
  constexpr unsigned int CovDimension = 4;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, ImageDimension>;
  using DoublePixelType = double;
  using DoubleImageType = itk::Image<DoublePixelType, ImageDimension>;
  using VecPixelType = itk::Vector<DoublePixelType, VectorDimension>;
  using VecImageType = itk::Image<VecPixelType, ImageDimension>;
  using CovPixelType = itk::CovariantVector<DoublePixelType, CovDimension>;
  using CovImageType = itk::Image<CovPixelType, ImageDimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // Invert the input image
  using InvertType = itk::InvertIntensityImageFilter<ImageType, ImageType>;
  InvertType::Pointer inverter = InvertType::New();
  inverter->SetInput(reader->GetOutput());

  // Cast the image to double type.
  using CasterType = itk::CastImageFilter<ImageType, DoubleImageType>;
  CasterType::Pointer caster = CasterType::New();
  CasterType::Pointer caster2 = CasterType::New();

  // Create an image, were each pixel has 2 values: first value is the value
  // coming from the input image, second value is coming from the inverted
  // image
  using ComposeType = itk::ComposeImageFilter<DoubleImageType, VecImageType>;
  ComposeType::Pointer composer = ComposeType::New();
  caster->SetInput(reader->GetOutput());
  composer->SetInput(0, caster->GetOutput());
  caster2->SetInput(inverter->GetOutput());
  composer->SetInput(1, caster2->GetOutput());

  // Apply the gradient filter on the two images, this will return and image
  // with 4 values per pixel: two X and Y gradients
```

(continues on next page)

(continued from previous page)

```
  using FilterType = itk::GradientRecursiveGaussianImageFilter<VecImageType,␣
↪CovImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(composer->GetOutput());

  // Set up the filter to select each gradient
  using IndexSelectionType = itk::VectorIndexSelectionCastImageFilter<CovImageType,␣
↪DoubleImageType>;
  IndexSelectionType::Pointer indexSelectionFilter = IndexSelectionType::New();
  indexSelectionFilter->SetInput(filter->GetOutput());

  // Rescale for png output
  using RescalerType = itk::RescaleIntensityImageFilter<DoubleImageType, ImageType>;
  RescalerType::Pointer rescaler = RescalerType::New();
  rescaler->SetOutputMinimum(itk::NumericTraits<PixelType>::min());
  rescaler->SetOutputMaximum(itk::NumericTraits<PixelType>::max());
  rescaler->SetInput(indexSelectionFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(rescaler->GetOutput());

  // Write the X and Y images
  for (int i = 0; i < 4; ++i)
  {
    indexSelectionFilter->SetIndex(i);
    writer->SetFileName(filenames[i]);

    try
    {
      writer->Update();
    }
    catch (itk::ExceptionObject & error)
    {
      std::cerr << "Error: " << error << std::endl;
      return EXIT_FAILURE;
    }
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<*CovariantVector*<**typename** *NumericTraits*<**typename**
**class GradientRecursiveGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Computes the gradient of an image by convolution with the first derivative of a Gaussian.

This filter is implemented using the recursive gaussian filters.

This filter supports both scalar and vector pixel types within the input image, including VectorImage type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Apply GradientRecursiveGaussianImageFilter on Image with Vector type
- Implementation Of Snakes

See itk::GradientRecursiveGaussianImageFilter for additional documentation.

### Compute and Display Gradient of Image

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Compute and display the gradient of an image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkCovariantVector.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGradientImageFilter.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkPointData.h"
#include "vtkFloatArray.h"
#include "vtkImageData.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkRenderWindow.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"
#include "vtkGlyph3DMapper.h"
#include "vtkArrowSource.h"

static void
VectorImageToVTKImage(itk::Image<itk::CovariantVector<float, 2>, 2>::Pointer␣
↪vectorImage, vtkImageData * VTKImage);
```

(continues on next page)

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "inputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using FloatImageType = itk::Image<float, 2>;
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;

  // Create and setup a reader
  using ReaderType = itk::ImageFileReader<UnsignedCharImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a gradient filter
  using GradientFilterType = itk::GradientImageFilter<UnsignedCharImageType, float>;
  GradientFilterType::Pointer gradientFilter = GradientFilterType::New();
  gradientFilter->SetInput(reader->GetOutput());
  gradientFilter->Update();

  // Visualize original image
  using ConnectorType = itk::ImageToVTKImageFilter<UnsignedCharImageType>;
  ConnectorType::Pointer originalConnector = ConnectorType::New();
  originalConnector->SetInput(reader->GetOutput());

  vtkSmartPointer<vtkImageActor> originalActor = vtkSmartPointer<vtkImageActor>::
↪New();
  originalActor->SetInput(originalConnector->GetOutput());

  // Visualize gradient
  vtkSmartPointer<vtkImageData> gradientImage = vtkSmartPointer<vtkImageData>::New();
  VectorImageToVTKImage(gradientFilter->GetOutput(), gradientImage);

  vtkSmartPointer<vtkArrowSource> arrowSource = vtkSmartPointer<vtkArrowSource>::
↪New();

  vtkSmartPointer<vtkGlyph3DMapper> gradientMapper = vtkSmartPointer<vtkGlyph3DMapper>
↪::New();
  gradientMapper->ScalingOn();
  gradientMapper->SetScaleFactor(.05);
  gradientMapper->SetSourceConnection(arrowSource->GetOutputPort());
  gradientMapper->SetInputConnection(gradientImage->GetProducerPort());
  gradientMapper->Update();

  vtkSmartPointer<vtkActor> gradientActor = vtkSmartPointer<vtkActor>::New();
  gradientActor->SetMapper(gradientMapper);
```

(continues on next page)

(continued from previous page)

```
  // Visualize
  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.5, 1.0 };
  double rightViewport[4] = { 0.5, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(600, 300);

  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(1, 0, 0);

  leftRenderer->AddActor(originalActor);
  rightRenderer->AddActor(gradientActor);

  vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  renderWindowInteractor->SetInteractorStyle(style);

  renderWindowInteractor->SetRenderWindow(renderWindow);
  renderWindowInteractor->Initialize();

  renderWindowInteractor->Start();

  return EXIT_SUCCESS;
}

void
VectorImageToVTKImage(itk::Image<itk::CovariantVector<float, 2>, 2>::Pointer␣
↪vectorImage, vtkImageData * VTKImage)
{
  itk::Image<itk::CovariantVector<float, 2>, 2>::RegionType region = vectorImage->
↪GetLargestPossibleRegion();
  itk::Image<itk::CovariantVector<float, 2>, 2>::SizeType   imageSize = region.
↪GetSize();
  VTKImage->SetExtent(0, imageSize[0] - 1, 0, imageSize[1] - 1, 0, 0);

  vtkSmartPointer<vtkFloatArray> vectors = vtkSmartPointer<vtkFloatArray>::New();
  vectors->SetNumberOfComponents(3);
  vectors->SetNumberOfTuples(imageSize[0] * imageSize[1]);
  vectors->SetName("GradientVectors");

  int counter = 0;
  for (unsigned int j = 0; j < imageSize[1]; j++)
  {
    for (unsigned int i = 0; i < imageSize[0]; i++)
```

(continues on next page)

(continued from previous page)

```
    {
      itk::Image<itk::CovariantVector<float, 2>, 2>::IndexType index;
      index[0] = i;
      index[1] = j;

      itk::Image<itk::CovariantVector<float, 2>, 2>::PixelType pixel = vectorImage->
↪GetPixel(index);

      float v[2];
      v[0] = pixel[0];
      v[1] = pixel[1];
      v[2] = 0;
      vectors->InsertTupleValue(counter, v);
      counter++;
    }
  }
  // std::cout << region << std::endl;

  VTKImage->GetPointData()->SetVectors(vectors);
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOperatorValueType** = float, typename **TOutputValueType** = float, typename
**class GradientImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImageType*>

Computes the gradient of an image using directional derivatives.

Computes the gradient of an image using directional derivatives. The directional derivative at each pixel location is computed by convolution with a first-order derivative operator.

The second template parameter defines the value type used in the derivative operator (defaults to float). The third template parameter defines the value type used for output image (defaults to float). The output image is defined as a covariant vector image whose value type is specified as this third template parameter.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Gradient Of Vector Image
- Compute And Display Gradient Of Image

See itk::GradientImageFilter for additional documentation.

### Compute Gradient Magnitude of Grayscale Image

### Synopsis

This example demonstrates how to compute the magnitude of the gradient of an image.

### Results

Fig. 185: Input image

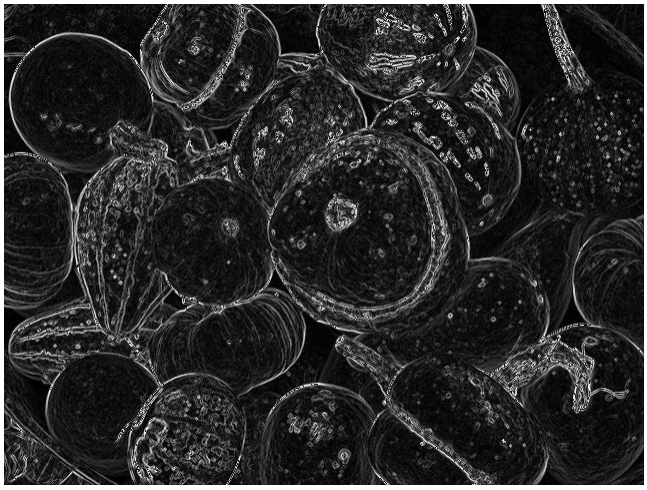


Fig. 186: Output image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGradientMagnitudeImageFilter.h"

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputImage> <OutputImage" << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using OutputPixelType = float;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  // Create and setup a reader
  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  // Create and setup a gradient filter
  using FilterType = itk::GradientMagnitudeImageFilter<InputImageType,␣
↪OutputImageType>;

  FilterType::Pointer gradientFilter = FilterType::New();
  gradientFilter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(gradientFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class GradientMagnitudeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Computes the gradient magnitude of an image region at each pixel.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Gradient Magnitude Of Grayscale Image

See itk::GradientMagnitudeImageFilter for additional documentation.

### Compute Gradient Magnitude Recursive Gaussian of Grayscale Image

### Synopsis

Compute the gradient magnitude of the image after first smoothing with a Gaussian kernel.

### Results

Fig. 187: Input image

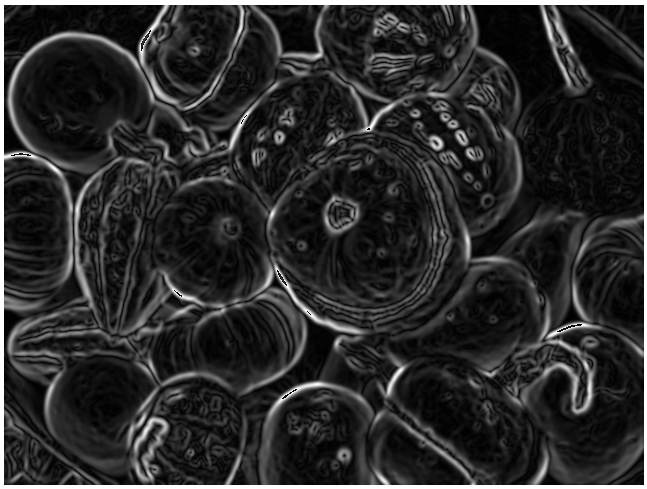


Fig. 188: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Compute Gradient Magnitude Recursive Gaussian."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("sigma")
args = parser.parse_args()

PixelType = itk.F
Dimension = 2
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

gradientMagnitudeImageFilter = itk.GradientMagnitudeRecursiveGaussianImageFilter.New(
    reader
)
gradientMagnitudeImageFilter.SetInput(reader.GetOutput())
gradientMagnitudeImageFilter.SetSigma(args.sigma)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(gradientMagnitudeImageFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName> <Sigma>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char * inputImage = argv[1];
  const char * outputImage = argv[2];
  const double sigma = std::stod(argv[3]);

  constexpr unsigned int Dimension = 2;
  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using ImageType = itk::Image<PixelType, Dimension>;

  using FilterType = itk::GradientMagnitudeRecursiveGaussianImageFilter<ImageType,␣
↪ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetSigma(sigma);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputImage);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class GradientMagnitudeRecursiveGaussianImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutput*
Computes the Magnitude of the Gradient of an image by convolution with the first derivative of a Gaussian.

This filter is implemented using the recursive gaussian filters

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Gradient Magnitude Of Grayscale Image

See itk::GradientMagnitudeRecursiveGaussianImageFilter for additional documentation.

### Gradient of Vector Image

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Compute the gradient of a vector image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkCovariantVector.h"
#include "itkGradientImageFilter.h"
#include "itkImageRegionIterator.h"

int
main(int argc, char * argv[])
{
  // Setup types
  using VectorType = itk::CovariantVector<float, 2>;
  using VectorImageType = itk::Image<VectorType, 2>;
  VectorImageType::Pointer image = VectorImageType::New();
```

(continues on next page)

(continued from previous page)

```
  itk::Size<2> size;
  size[0] = 5;
  size[1] = 5;

  itk::Index<2> index;
  index[0] = 0;
  index[1] = 0;

  VectorImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(index);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<VectorImageType> iterator(image, region);

  while (!iterator.IsAtEnd())
  {
  }

  // Create and setup a gradient filter
  using GradientFilterType = itk::GradientImageFilter<VectorImageType, VectorType>;
  GradientFilterType::Pointer gradientFilter = GradientFilterType::New();
  gradientFilter->SetInput(image);
  gradientFilter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOperatorValueType** = float, typename **TOutputValueType** = float, typename
**class GradientImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImageType*>

Computes the gradient of an image using directional derivatives.

Computes the gradient of an image using directional derivatives. The directional derivative at each pixel location is computed by convolution with a first-order derivative operator.

The second template parameter defines the value type used in the derivative operator (defaults to float). The third template parameter defines the value type used for output image (defaults to float). The output image is defined as a covariant vector image whose value type is specified as this third template parameter.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Gradient Of Vector Image

- Compute And Display Gradient Of Image

See itk::GradientImageFilter for additional documentation.

### Implementation of Snakes

### Synopsis

Simple implementation of Snakes.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkGradientRecursiveGaussianImageFilter.h"
#include "itkGradientMagnitudeImageFilter.h"
#include "itkImageFileReader.h"

#include "itkMath.h"
#include <vnl/vnl_matrix.h>
#include "vnl/algo/vnl_determinant.h"
#include "vnl/algo/vnl_matrix_inverse.h"
#include <vnl/vnl_vector.h>

#include <iostream>

namespace
{
using ImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;
using IndexType = ImageType::IndexType;
using OutputPixelType = itk::CovariantVector<float, 2>;
using OutputImageType = itk::Image<OutputPixelType, 2>;
using FilterType = itk::GradientRecursiveGaussianImageFilter<FloatImageType,␣
↪OutputImageType>;
using GradMagfilterType = itk::GradientMagnitudeImageFilter<ImageType, FloatImageType>
↪;
using ReaderType = itk::ImageFileReader<ImageType>;
} // namespace

vnl_vector<double>
generateCircle(double cx, double cy, double rx, double ry, int n);
```

(continues on next page)

(continued from previous page)

```
void
createImage(ImageType::Pointer image, int w, int h, double cx, double cy, double rx,␣
↪double ry);
vnl_matrix<double>
computeP(double alpha, double beta, double gamma, double N) throw();
vnl_vector<double>
sampleImage(vnl_vector<double> x, vnl_vector<double> y, OutputImageType::Pointer␣
↪gradient, int position);

int
main(int argc, char * argv[])
{
  // Image dimensions
  int                w = 300;
  int                h = 300;
  ImageType::Pointer image;
  if (argc < 7)
  {
    std::cout << "Usage " << argv[0] << " points alpha beta gamma sigma iterations␣
↪[image]" << std::endl;
    return EXIT_FAILURE;
  }
  else if (argc < 8)
  {
    // Synthesize the image
    image = ImageType::New();
    createImage(image, w, h, 150, 150, 50, 50);
  }
  else if (argc == 8)
  {
    // Open the image
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[7]);
    try
    {
      reader->Update();
      image = reader->GetOutput();
      w = image->GetLargestPossibleRegion().GetSize()[0];
      h = image->GetLargestPossibleRegion().GetSize()[1];
    }
    catch (itk::ExceptionObject & err)
    {
      std::cerr << "Caught unexpected exception " << err;
      return EXIT_FAILURE;
    }
  }

  // Snake parameters
  double alpha = 0.001;
  double beta = 0.4;
  double gamma = 100;
  double iterations = 1;
  int    nPoints = 20;
  double sigma;

  nPoints = std::stoi(argv[1]);
  alpha = std::stod(argv[2]);
```

(continues on next page)

(continued from previous page)

```
  beta = std::stod(argv[3]);
  gamma = std::stod(argv[4]);
  sigma = std::stod(argv[5]);
  iterations = std::stoi(argv[6]);

  // Temporal variables
  vnl_matrix<double> P;
  vnl_vector<double> v;
  double             N;

  // Generate initial snake circle
  v = generateCircle(130, 130, 50, 50, nPoints);

  // Compute P matrix.
  N = v.size() / 2;
  try
  {
    P = computeP(alpha, beta, gamma, N);
  }
  catch (...)
  {
    return EXIT_FAILURE;
  }

  // Compute the magnitude gradient
  GradMagfilterType::Pointer gradientMagnitudeFilter = GradMagfilterType::New();
  gradientMagnitudeFilter->SetInput(image);
  gradientMagnitudeFilter->Update();

  // Compute the gradient of the gradient magnitude
  FilterType::Pointer gradientFilter = FilterType::New();
  gradientFilter->SetInput(gradientMagnitudeFilter->GetOutput());
  gradientFilter->SetSigma(sigma);
  gradientFilter->Update();

  // Loop
  vnl_vector<double> x(N);
  vnl_vector<double> y(N);

  std::cout << "Initial snake" << std::endl;
  for (int i = 0; i < N; ++i)
  {
    x[i] = v[2 * i];
    y[i] = v[2 * i + 1];
    std::cout << "(" << x[i] << ", " << y[i] << ")" << std::endl;
  }

  for (int i = 0; i < iterations; ++i)
  {
    vnl_vector<double> fex;
    vnl_vector<double> fey;
    fex = sampleImage(x, y, gradientFilter->GetOutput(), 0);
    fey = sampleImage(x, y, gradientFilter->GetOutput(), 1);

    x = (x + gamma * fex).post_multiply(P);
    y = (y + gamma * fey).post_multiply(P);
  }
```

(continues on next page)

(continued from previous page)

```
  // Display the answer
  std::cout << "Final snake after " << iterations << " iterations" << std::endl;
  vnl_vector<double> v2(2 * N);
  for (int i = 0; i < N; ++i)
  {
    v2[2 * i] = x[i];
    v2[2 * i + 1] = y[i];
    std::cout << "(" << x[i] << ", " << y[i] << ")" << std::endl;
  }

  return EXIT_SUCCESS;
}

vnl_vector<double>
generateCircle(double cx, double cy, double rx, double ry, int n)
{
  vnl_vector<double> v(2 * (n + 1));

  for (int i = 0; i < n; ++i)
  {
    v[2 * i] = cx + rx * cos(2 * itk::Math::pi * i / n);
    v[2 * i + 1] = cy + ry * sin(2 * itk::Math::pi * i / n);
  }
  v[2 * n] = v[0];
  v[2 * n + 1] = v[1];
  return v;
}

void
createImage(ImageType::Pointer image, int w, int h, double cx, double cy, double rx,␣
↪double ry)
{
  itk::Size<2> size;
  size[0] = w;
  size[1] = h;

  itk::RandomImageSource<ImageType>::Pointer randomImageSource = itk::
↪RandomImageSource<ImageType>::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce non-random results
  randomImageSource->SetSize(size);
  randomImageSource->SetMin(200);
  randomImageSource->SetMax(255);
  randomImageSource->Update();

  image->SetRegions(randomImageSource->GetOutput()->GetLargestPossibleRegion());
  image->Allocate();

  IndexType index;

  // Draw oval
  for (int i = 0; i < w; ++i)
  {
    for (int j = 0; j < h; ++j)
    {
      index[0] = i;
      index[1] = j;
```

(continues on next page)

(continued from previous page)

```
      if (((i - cx) * (i - cx) / (rx * rx) + (j - cy) * (j - cy) / (ry * ry)) < 1)
      {
        image->SetPixel(index, randomImageSource->GetOutput()->GetPixel(index) - 100);
      }
      else
      {
        image->SetPixel(index, randomImageSource->GetOutput()->GetPixel(index));
      }
    }
  }
}

vnl_matrix<double>
computeP(double alpha, double beta, double gamma, double N) throw()
{

  double a = gamma * (2 * alpha + 6 * beta) + 1;
  double b = gamma * (-alpha - 4 * beta);
  double c = gamma * beta;

  vnl_matrix<double> P(N, N);

  P.fill(0);

  // Fill diagonal
  P.fill_diagonal(a);

  // Fill next two diagonals
  for (int i = 0; i < N - 1; ++i)
  {
    P(i + 1, i) = b;
    P(i, i + 1) = b;
  }
  // Moreover
  P(0, N - 1) = b;
  P(N - 1, 0) = b;

  // Fill next two diagonals
  for (int i = 0; i < N - 2; ++i)
  {
    P(i + 2, i) = c;
    P(i, i + 2) = c;
  }
  // Moreover
  P(0, N - 2) = c;
  P(1, N - 1) = c;
  P(N - 2, 0) = c;
  P(N - 1, 1) = c;

  if (vnl_determinant(P) == 0.0)
  {
    std::cerr << "Singular matrix. Determinant is 0." << std::endl;
    throw;
  }

  // Compute the inverse of the matrix P
  vnl_matrix<double> Pinv;
```

(continues on next page)

(continued from previous page)

```
  Pinv = vnl_matrix_inverse<double>(P);

  return Pinv.transpose();
}

vnl_vector<double>
sampleImage(vnl_vector<double> x, vnl_vector<double> y, OutputImageType::Pointer␣
↪gradient, int position)
{
  int size;
  size = x.size();
  vnl_vector<double> ans(size);

  IndexType index;
  for (int i = 0; i < size; ++i)
  {
    index[0] = x[i];
    index[1] = y[i];
    ans[i] = gradient->GetPixel(index)[position];
  }
  return ans;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<*CovariantVector*<**typename** *NumericTraits*<**typename**
**class GradientRecursiveGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Computes the gradient of an image by convolution with the first derivative of a Gaussian.

This filter is implemented using the recursive gaussian filters.

This filter supports both scalar and vector pixel types within the input image, including VectorImage type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply GradientRecursiveGaussianImageFilter on Image with Vector type
- Implementation Of Snakes

See itk::GradientRecursiveGaussianImageFilter for additional documentation.

### 3.4.16 ImageGrid

#### Append Two 3D Volumes

#### Synopsis

Append two 3D volumes in the Z-direction.

### Results

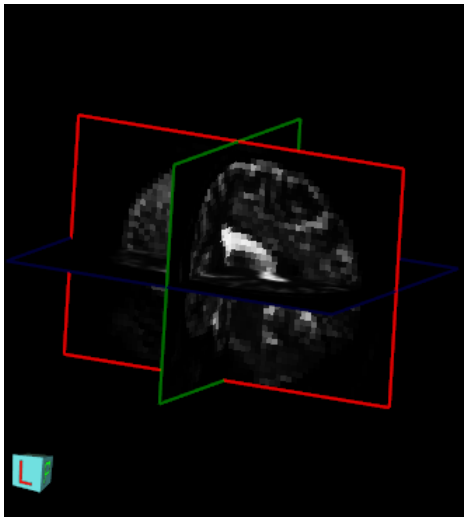


Fig. 189: Input image

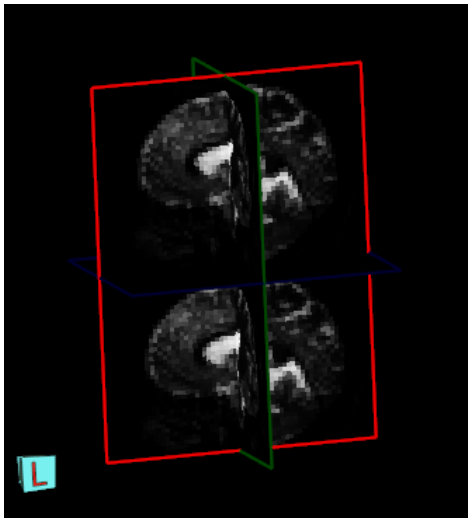


Fig. 190: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkTileImageFilter.h"

int
main(int argc, char * argv[])
```

(continues on next page)

(continued from previous page)

```
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName1> <InputFileName2> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char * inputFileName1 = argv[1];
  const char * inputFileName2 = argv[2];
  const char * outputFileName = argv[3];

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName(inputFileName1);

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName(inputFileName2);

  using TileFilterType = itk::TileImageFilter<ImageType, ImageType>;
  TileFilterType::Pointer tileFilter = TileFilterType::New();
  tileFilter->SetInput(0, reader1->GetOutput());
  tileFilter->SetInput(1, reader2->GetOutput());
  TileFilterType::LayoutArrayType layout;
  layout[0] = 1;
  layout[1] = 1;
  layout[2] = 2;
  tileFilter->SetLayout(layout);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(tileFilter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class TileImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Tile multiple input images into a single output image.

This filter will tile multiple images using a user-specified layout. The tile sizes will be large enough to accommodate the largest image for each tile. The layout is specified with the SetLayout method. The layout has the same dimension as the output image. If all entries of the layout are positive, the tiled output will contain the exact number of tiles. If the layout contains a 0 in the last dimension, the filter will compute a size that will accommodate all of the images. Empty tiles are filled with the value specified with the SetDefault value method. The input images must have a dimension less than or equal to the output image. The output image have a larger dimension than the input images. This filter can be used to create a volume from a series of inputs by specifying a layout of 1,1,0.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Stack 2D Images Into 3D Image
- Tile Images Side By Side

See itk::TileImageFilter for additional documentation.

### Change Image Origin Spacing or Direction

### Synopsis

Change an Image's Origin, Spacing, or Direction.

The *ChangeInformationImageFilter* is commonly used to modify image metadata such as *Origin*, *Spacing*, and *Orientation*. This filter leaves intact the pixel data of the image. This filter should be used with extreme caution, since it can easily change information that is critical for the safety of many medical image analysis tasks, such as measurement the volume of a tumor, or providing guidance for surgery.

### Results

```
Original image: Image (0x3131ce0)
  RTTI typeinfo:   itk::Image<unsigned char, 3u>
  Reference Count: 3
  Modified Time: 195
  Debug: Off
  Object Name:
  Observers:
    none
  Source: (0x312d620)
  Source output name: Primary
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 47
  UpdateMTime: 0
```

(continues on next page)

(continued from previous page)

```
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [48, 62, 42]
  BufferedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  RequestedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  Spacing: [4, 4, 4]
  Origin: [0, 0, 0]
  Direction:
1 0 0
0 1 0
0 0 1

  IndexToPointMatrix:
4 0 0
0 4 0
0 0 4

  PointToIndexMatrix:
0.25 0 0
0 0.25 0
0 0 0.25

  Inverse Direction:
1 0 0
0 1 0
0 0 1

  PixelContainer:
    ImportImageContainer (0x3131fa0)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned char>
      Reference Count: 1
      Modified Time: 34
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 0
      Container manages memory: true
      Size: 0
      Capacity: 0

************************************
Changed image: Image (0x3137100)
  RTTI typeinfo:   itk::Image<unsigned char, 3u>
  Reference Count: 3
  Modified Time: 222
  Debug: Off
  Object Name:
  Observers:
```

(continues on next page)

(continued from previous page)

```
    none
  Source: (0x3132240)
  Source output name: Primary
  Release Data: Off
  Data Released: False
  Global Release Data: Off
  PipelineMTime: 215
  UpdateMTime: 0
  RealTimeStamp: 0 seconds
  LargestPossibleRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [48, 62, 42]
  BufferedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  RequestedRegion:
    Dimension: 3
    Index: [0, 0, 0]
    Size: [0, 0, 0]
  Spacing: [8.4, 8.4, 8.4]
  Origin: [8.4, 2.9, 4.4]
  Direction:
0.866025 -0.5 0
0.5 0.866025 0
0 0 1

  IndexToPointMatrix:
7.27461 -4.2 0
4.2 7.27461 0
0 0 8.4

  PointToIndexMatrix:
0.103098 0.0595238 0
-0.0595238 0.103098 0
0 0 0.119048

  Inverse Direction:
0.866025 0.5 0
-0.5 0.866025 0
0 0 1

  PixelContainer:
    ImportImageContainer (0x3137520)
      RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned char>
      Reference Count: 1
      Modified Time: 200
      Debug: Off
      Object Name:
      Observers:
        none
      Pointer: 0
      Container manages memory: true
      Size: 0
      Capacity: 0
```

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkVersor.h"
#include "itkChangeInformationImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0] << " <inputFileName>"
              << " [scalingFactor]"
              << " [translationX translationY translationZ]"
              << " [rotationZinDegrees]" << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  double       scalingFactor = 1.0;
  if (argc > 3)
  {
    scalingFactor = std::stod(argv[3]);
  }
  double translationX = 0.0;
  if (argc > 3)
  {
    translationX = std::stod(argv[3]);
  }
  double translationY = 0.0;
  if (argc > 4)
  {
    translationY = std::stod(argv[4]);
  }
  double translationZ = 0.0;
  if (argc > 5)
  {
    translationZ = std::stod(argv[5]);
  }
  double rotationZ = 0.0;
  if (argc > 6)
  {
    rotationZ = std::stod(argv[6]);
  }

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  try
```

(continues on next page)

(continued from previous page)

```
  {
    reader->UpdateOutputInformation();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
  ImageType::ConstPointer inputImage = reader->GetOutput();
  std::cout << "Original image: " << inputImage << std::endl;

  using FilterType = itk::ChangeInformationImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  const ImageType::SpacingType spacing(scalingFactor);
  filter->SetOutputSpacing(spacing);
  filter->ChangeSpacingOn();

  ImageType::PointType::VectorType translation;
  translation[0] = translationX;
  translation[1] = translationY;
  translation[2] = translationZ;
  ImageType::PointType origin = inputImage->GetOrigin();
  origin += translation;
  filter->SetOutputOrigin(origin);
  filter->ChangeOriginOn();

  itk::Versor<double> rotation;
  const double        angleInRadians = rotationZ * itk::Math::pi / 180.0;
  rotation.SetRotationAroundZ(angleInRadians);
  const ImageType::DirectionType direction = inputImage->GetDirection();
  const ImageType::DirectionType newDirection = direction * rotation.GetMatrix();
  filter->SetOutputDirection(newDirection);
  filter->ChangeDirectionOn();

  try
  {
    filter->UpdateOutputInformation();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
  std::cout << "************************************" << std::endl;
  ImageType::ConstPointer output = filter->GetOutput();
  std::cout << "Changed image: " << output << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ChangeInformationImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TInputImage*>
Change the origin, spacing and/or region of an Image.

Change the origin, spacing, direction and/or buffered region of an itkImage. This "Information" along with an Image's container comprise the itkImage. By default, the output's information is set to the input's information. The methods ChangeSpacingOn/Off, ChangeOriginOn/Off, ChangeDirectionOn/Off and ChangeRegionOn/Off control whether the default origin, spacing, direction or buffered region should be changed. If On, the associated information will be replaced with either the ReferenceImage information (if UseReferenceImage is true) or the ivars OutputSpacing, OutputOrigin, OutputDirection, OutputOffset.

In addition, the method CenterImageOn will recompute the output image origin (using the selected output spacing) the align the center of the image with the coordinate 0.

See itk::ChangeInformationImageFilter for additional documentation.

### Create 3D Volume

### Synopsis

This example reads in a series of 2D images and stacks them to create a 3D image.

### Results

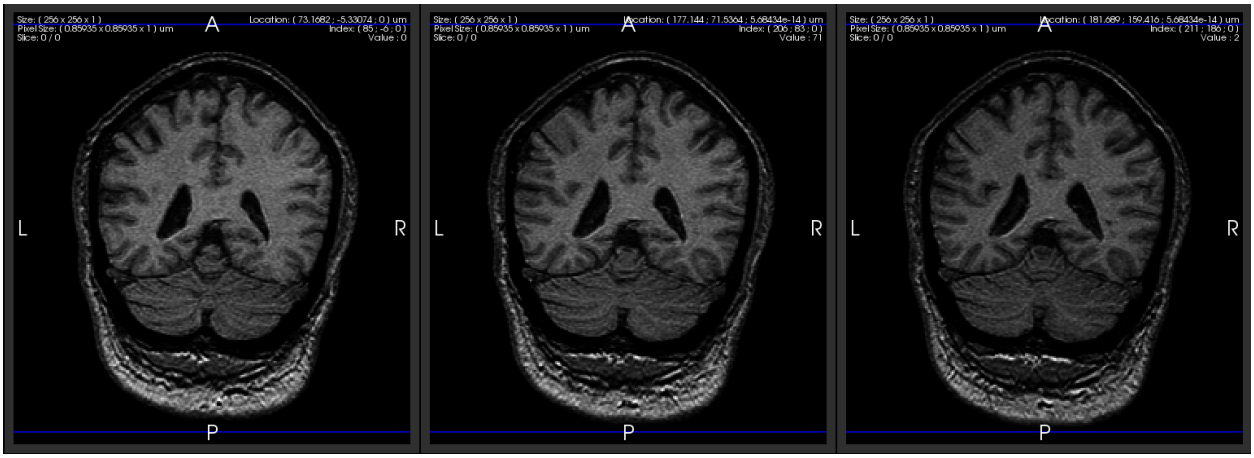



Fig. 191: Input 2D images

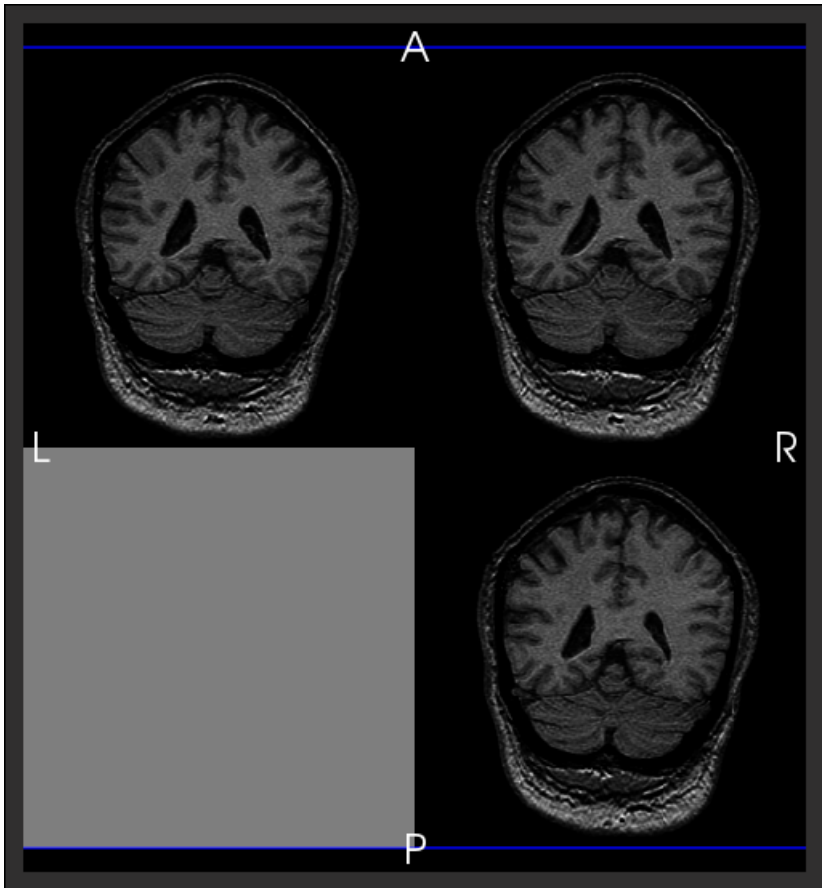



Fig. 192: Single output 3D image

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk

if len(sys.argv) < 3:
    print("Usage: " + sys.argv[0] + " <input1> <input2> <input3> ... <output>")
    sys.exit(1)

InputDimension = 2
OutputDimension = 3

PixelType = itk.UC

InputImageType = itk.Image[PixelType, InputDimension]
OutputImageType = itk.Image[PixelType, OutputDimension]

reader = itk.ImageFileReader[InputImageType].New()

tileFilter = itk.TileImageFilter[InputImageType, OutputImageType].New()

layout = [2, 2, 0]
tileFilter.SetLayout(layout)

for ii in range(1, len(sys.argv) - 1):
    reader.SetFileName(sys.argv[ii])
    reader.Update()

    inputImage = reader.GetOutput()
    inputImage.DisconnectPipeline()

    tileFilter.SetInput(ii - 1, inputImage)

defaultValue = 128
tileFilter.SetDefaultPixelValue(defaultValue)
tileFilter.Update()

writer = itk.ImageFileWriter[OutputImageType].New()
writer.SetFileName(sys.argv[-1])
writer.SetInput(tileFilter.GetOutput())
writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkTileImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<input1> <input2> <input3> ... <output>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int InputDimension = 2;
  constexpr unsigned int OutputDimension = 3;

  using PixelType = unsigned char;
  using InputImageType = itk::Image<PixelType, InputDimension>;
  using OutputImageType = itk::Image<PixelType, OutputDimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();

  using FilterType = itk::TileImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();

  itk::FixedArray<unsigned int, OutputDimension> layout;
  layout[0] = 2;
  layout[1] = 2;
  layout[2] = 0;

  filter->SetLayout(layout);

  for (int ii = 1; ii < argc - 1; ++ii)
  {
    reader->SetFileName(argv[ii]);

    try
    {
      reader->Update();
    }
    catch (itk::ExceptionObject & e)
    {
      std::cerr << e << std::endl;
      return EXIT_FAILURE;
    }

    InputImageType::Pointer input = reader->GetOutput();
    input->DisconnectPipeline();

    filter->SetInput(ii - 1, input);
```

(continues on next page)

(continued from previous page)

```
  }

  constexpr PixelType defaultValue = 128;

  filter->SetDefaultPixelValue(defaultValue);

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[argc - 1]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class TileImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Tile multiple input images into a single output image.

This filter will tile multiple images using a user-specified layout. The tile sizes will be large enough to accommodate the largest image for each tile. The layout is specified with the SetLayout method. The layout has the same dimension as the output image. If all entries of the layout are positive, the tiled output will contain the exact number of tiles. If the layout contains a 0 in the last dimension, the filter will compute a size that will accommodate all of the images. Empty tiles are filled with the value specified with the SetDefault value method. The input images must have a dimension less than or equal to the output image. The output image have a larger dimension than the input images. This filter can be used to create a volume from a series of inputs by specifying a layout of 1,1,0.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Stack 2D Images Into 3D Image
- Tile Images Side By Side

See itk::TileImageFilter for additional documentation.

### Crop Image by Specifying Region

### Synopsis

Crop an image by specifying the region to throw away.

### Results

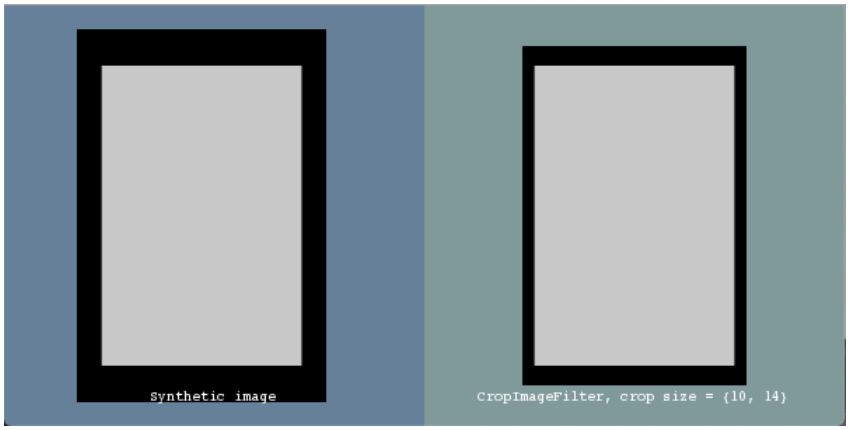



Fig. 193: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageFileReader.h"
#include "itkCropImageFilter.h"

#include "itksys/SystemTools.hxx"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using PixelType = itk::RGBPixel<unsigned char>;
using ImageType = itk::Image<PixelType, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer  image = ImageType::New();
  ImageType::SizeType cropSize;
  std::stringstream   desc;
```

(continues on next page)

(continued from previous page)

```
  if (argc > 1)
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    if (argc > 2)
    {
      cropSize[0] = std::stoi(argv[2]);
      cropSize[1] = std::stoi(argv[3]);
    }
    reader->Update();
    image = reader->GetOutput();
    desc << itksys::SystemTools::GetFilenameName(argv[1]);
  }
  else
  {
    CreateImage(image);
    cropSize[0] = 10;
    cropSize[1] = 14;
    desc << "Synthetic image";
  }

  using CropImageFilterType = itk::CropImageFilter<ImageType, ImageType>;

  CropImageFilterType::Pointer cropFilter = CropImageFilterType::New();
  cropFilter->SetInput(image);
  // The SetBoundaryCropSize( cropSize ) method specifies the size of
  // the boundary to be cropped at both the uppper & lower ends of the
  // image eg. cropSize pixels will be removed at both upper & lower
  // extents

  cropFilter->SetBoundaryCropSize(cropSize);

  // The below two lines are equivalent to the above line:
  // cropFilter->SetUpperBoundaryCropSize(cropSize);
  // cropFilter->SetLowerBoundaryCropSize(cropSize);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(image.GetPointer(), true, desc.str());

  std::stringstream desc2;
  desc2 << "CropImageFilter, crop size = {" << cropSize[0] << ", " << cropSize[1] <<
↪"}";
  viewer.AddRGBImage(cropFilter->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
```

(continues on next page)

(continued from previous page)

```
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(itk::NumericTraits<ImageType::PixelType>::ZeroValue());

  // Make a rectangle, centered at (100,150) with sides 160 & 240
  // This provides a 20 x 30 border around the square for the crop filter to remove
  for (unsigned int r = 20; r < 180; r++)
  {
    for (unsigned int c = 30; c < 270; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 200);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CropImageFilter** : **public** itk::*ExtractImageFilter<TInputImage, TOutputImage>*

Decrease the image size by cropping the image by an itk::Size at both the upper and lower bounds of the largest possible region.

CropImageFilter changes the image boundary of an image by removing pixels outside the target region. The target region is not specified in advance, but calculated in BeforeThreadedGenerateData().

This filter uses ExtractImageFilter to perform the cropping.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Crop Image By Specifying Region

See itk::CropImageFilter for additional documentation.

### Extract Region of Interest in One Image

### Synopsis

Extract a given Region Of Interest (ROI) in a given image

### Results

Fig. 194: Input image

Fig. 195: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRegionOfInterestImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 7)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " <start x> <end x> <start y> <end y>";
    std::cerr << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  const auto startx = static_cast<itk::IndexValueType>(std::stoi(argv[3]));
  const auto endx = static_cast<itk::IndexValueType>(std::stoi(argv[4]));

  const auto starty = static_cast<itk::IndexValueType>(std::stoi(argv[5]));
  const auto endy = static_cast<itk::IndexValueType>(std::stoi(argv[6]));

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  ImageType::IndexType start;
  start[0] = startx;
  start[1] = starty;

  ImageType::IndexType end;
  end[0] = endx;
  end[1] = endy;

  ImageType::RegionType region;
  region.SetIndex(start);
  region.SetUpperIndex(end);

  using FilterType = itk::RegionOfInterestImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetRegionOfInterest(region);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RegionOfInterestImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Extract a region of interest from the input image.

This filter produces an output image of the same dimension as the input image. The user specifies the region of the input image that will be contained in the output image. The origin coordinates of the output images will be computed in such a way that if mapped to physical space, the output image will overlay the input image with perfect registration. In other words, a registration process between the output image and the input image will return an identity transform.

If you are interested in changing the dimension of the image, you may want to consider the ExtractImageFilter. For example for extracting a 2D image from a slice of a 3D image.

The region to extract is set using the method SetRegionOfInterest.

**See** ExtractImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Region Of Interest In One Image

See itk::RegionOfInterestImageFilter for additional documentation.

### Fit Spline Into Point Set

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Fit a spline to a point set.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkBSplineScatteredDataPointSetToImageFilter.h"
#include "itkPointSet.h"
#include "itkImage.h"
#include "itkVectorImage.h"
#include "itkImageFileWriter.h"

int
main()
{
  constexpr unsigned int ParametricDimension = 1;
  constexpr unsigned int DataDimension = 2;

  using DataType = itk::Vector<float, DataDimension>;

  using PointSetType = itk::PointSet<DataType, ParametricDimension>;

  PointSetType::Pointer pointSet = PointSetType::New();

  PointSetType::PointType param0, param1, param2;

  param0[0] = 0.0;
  DataType p0;
  p0[0] = 10.0;
  p0[1] = 10.0;

  pointSet->SetPoint(0, param0);
  pointSet->SetPointData(0, p0);

  param1[0] = 1.0;
  DataType p1;
  p1[0] = 80.0;
  p1[1] = 50.0;
  pointSet->SetPoint(1, param1);
  pointSet->SetPointData(1, p1);

  param2[0] = 2.0;
  DataType p2;
  p2[0] = 180.0;
  p2[1] = 180.0;
  pointSet->SetPoint(2, param2);
  pointSet->SetPointData(2, p2);

  using ImageType = itk::Image<DataType, ParametricDimension>;
  using SplineFilterType = itk::BSplineScatteredDataPointSetToImageFilter
↪<PointSetType, ImageType>;
  SplineFilterType::Pointer splineFilter = SplineFilterType::New();

  int splineorder = 2; // complexity of the spline

  SplineFilterType::ArrayType ncontrol;
  ncontrol[0] = splineorder + 1;
  SplineFilterType::ArrayType closedim;
  closedim[0] = 0;
```

(continues on next page)

(continued from previous page)

```
  ImageType::PointType parametricDomainOrigin;
  parametricDomainOrigin[0] = 0.0;

  ImageType::SpacingType parametricDomainSpacing;
  parametricDomainSpacing[0] = 0.0001; // this determines the sampling of the␣
↪continuous B-spline object.

  ImageType::SizeType parametricDomainSize;
  parametricDomainSize[0] = 2.0 / parametricDomainSpacing[0] + 1;
  splineFilter->SetGenerateOutputImage(true); // the only reason to turn this off is␣
↪if one only wants to use the
                                              // control point lattice for further␣
↪processing
  splineFilter->SetInput(pointSet);
  splineFilter->SetSplineOrder(splineorder);
  splineFilter->SetNumberOfControlPoints(ncontrol);
  splineFilter->SetNumberOfLevels(3);
  splineFilter->SetCloseDimension(closedim);
  splineFilter->SetSize(parametricDomainSize);
  splineFilter->SetSpacing(parametricDomainSpacing);
  splineFilter->SetOrigin(parametricDomainOrigin);
  splineFilter->Update();

  // The output will consist of a 1-D image where each voxel contains the
  // (x,y,z) locations of the points
  using OutputImageType = itk::Image<unsigned char, 2>;
  OutputImageType::Pointer  outputImage = OutputImageType::New();
  OutputImageType::SizeType size;
  size.Fill(200);

  OutputImageType::IndexType start;
  start.Fill(0);

  OutputImageType::RegionType region(start, size);
  outputImage->SetRegions(region);
  outputImage->Allocate();
  outputImage->FillBuffer(0);

  for (unsigned int i = 0; i < splineFilter->GetOutput()->GetLargestPossibleRegion().
↪GetSize()[0]; ++i)
  {
    ImageType::IndexType splineIndex;
    splineIndex[0] = i;

    DataType outputPixel = splineFilter->GetOutput()->GetPixel(splineIndex);

    OutputImageType::IndexType index;
    index[0] = outputPixel[0];
    index[1] = outputPixel[1];

    outputImage->SetPixel(index, 255);
  }

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("spline.png");
```

(continues on next page)

(continued from previous page)

```
  writer->SetInput(outputImage);
  writer->Update();

  return EXIT_SUCCESS;
};
```

### Classes demonstrated

template<typename **TInputPointSet**, typename **TOutputImage**>
**class BSplineScatteredDataPointSetToImageFilter** : **public** itk::PointSetToImageFilter<*TInputPointSet*, *TOutputI*
Image filter which provides a B-spline output approximation.

Given an n-D image with scattered data, this filter finds a fast approximation to that irregularly spaced data using uniform B-splines. The traditional method of inverting the observation matrix to find a least-squares fit is made obsolete. Therefore, memory issues are not a concern and inverting large matrices is not applicable. In addition, this allows fitting to be multi-threaded. This class generalizes from Lee's original paper to encompass n-D data in m-D parametric space and any *feasible* B-spline order as well as the option of specifying a confidence value for each point.

In addition to specifying the input point set, one must specify the number of control points. The specified number of control points must be greater than m_SplineOrder. If one wishes to use the multilevel component of this algorithm, one must also specify the number of levels in the hierarchy. If this is desired, the number of control points becomes the number of control points for the coarsest level. The algorithm then increases the number of control points at each level so that the B-spline n-D grid is refined to twice the previous level.

There are two parts to fitting scattered data: the parameterization assignment problem and the fitting problem given a parameterization. This filter only addresses the second problem in that the user must provide a parametric value for each scattered datum. Different parametric assignment schemes result in different B-spline object outputs.

This filter is general in that it accepts n-D scattered data in m-D parametric dimensions. Input to this filter is an m-D point set with a Vector data type of n dimensions. This means that the parametric values are stored in the points container of the point set whereas the scattered data are stored in the points data container of the point set.

Typical B-spline objects include curves, which have a parametric dimension of 1 and a data dimension of 2 or 3 (depending on the space in which the curve resides) and deformation fields which commonly have parametric and data dimensions of 2 or 3 (again depending on the space of the field). As an example, a curve through a set of 2D points has data dimension 2 and parametric dimension 1. The univariate curve could be represented as: <x(u),y(u)> Another example is a 3D deformation of 3D points, which has parametric dimension 3 and data dimension 3 and can be represented as: <dx(u,v,w), dy(u,v,w), dz(u,v,w)>. However, as mentioned before, the code is general such that, if the user wanted, she could model a time varying 3-D displacement field which resides in 4-D space as <dx(u, v, w, t), dy(u, v, w, t), dz(u, v, w, t)>.

The output is an image defining the sampled B-spline parametric domain where each pixel houses the sampled B-spline object value. For a curve fit to 3-D points, the output is a 1-D image where each voxel contains a vector with the approximated (x,y,z) location. The continuous, finite, rectilinear domain (as well as the sampling rate) is specified via the combination of the SetSpacing() and SetSize() functions. For a 2-D deformation on 2-D points, the output is a 2-D image where each voxel contains the approximated (dx, dy) vector.

The parameterization must be specified using SetPoint, where the actual coordinates of the point are set via SetPointData. For example, to compute a spline through the (ordered) 2D points (5,6) and (7,8), you should use:

```
using DataType = itk::Vector< float, 2 >;
PointSetType::PointType param0;
```

(continues on next page)

(continued from previous page)

```
param0[0] = 0.0;
DataType p0;
p0[0] =  10.0; p0[1]= 10.0;
pointSet->SetPoint(0, param0);
pointSet->SetPointData( 0, p0 );

PointSetType::PointType param1;
param1[0] = 1.0;
DataType p1;
p1[0] =  80.0; p1[1]= 50.0;
pointSet->SetPoint(1, param1);
pointSet->SetPointData( 1, p1 );
```

This code was contributed in the Insight Journal paper: "N-D C^k B-Spline Scattered Data Approximation" by Nicholas J. Tustison, James C. Gee https://www.insight-journal.org/browse/publication/57

**Author** Nicholas J. Tustison

**REFERENCE**

S. Lee, G. Wolberg, and S. Y. Shin, "Scattered Data Interpolation

with Multilevel B-Splines", IEEE Transactions on Visualization and Computer Graphics, 3(3):228-244, 1997.

**REFERENCE**

N.J. Tustison and J.C. Gee, "Generalized n-D C^k Scattered Data Approximation

with Confidence Values", Proceedings of the MIAR conference, August 2006.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
-

See itk::BSplineScatteredDataPointSetToImageFilter for additional documentation.

### Flip an Image Over Specified Axes

### Synopsis

Flip an image over the specified axes. Pixels are swapped over the given axis.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
```

(continues on next page)

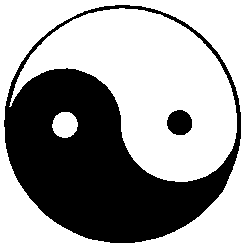


Fig. 196: Input image

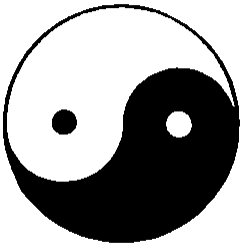


Fig. 197: Output image

(continued from previous page)

```
import argparse

parser = argparse.ArgumentParser(description="Flip An Image Over Specified Axis.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("axis_to_flip", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

flipFilter = itk.FlipImageFilter[ImageType].New()
flipFilter.SetInput(reader.GetOutput())

if args.axis_to_flip == 0:
    flipAxes = (True, False)
else:
    flipAxes = (False, True)

flipFilter.SetFlipAxes(flipAxes)
```

(continues on next page)

(continued from previous page)

```
writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(flipFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkFlipImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <AxisToFlip>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using FlipImageFilterType = itk::FlipImageFilter<ImageType>;

  FlipImageFilterType::Pointer flipFilter = FlipImageFilterType::New();
  flipFilter->SetInput(reader->GetOutput());

  FlipImageFilterType::FlipAxesArrayType flipAxes;
  if (std::stoi(argv[3]) == 0)
  {
    flipAxes[0] = true;
    flipAxes[1] = false;
  }
  else
  {
    flipAxes[0] = false;
    flipAxes[1] = true;
  }

  flipFilter->SetFlipAxes(flipAxes);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
```

(continues on next page)

(continued from previous page)

```
  writer->SetFileName(argv[2]);
  writer->SetInput(flipFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class FlipImageFilter** : **public** itk::*ImageToImageFilter*<*TImage*, *TImage*>
Flips an image across user specified axes.

FlipImageFilter flips an image across user specified axes. The flip axes are set via method SetFlipAxes( array ) where the input is a FixedArray<bool,ImageDimension>. The image is flipped across axes for which array[i] is true.

In terms of grid coordinates the image is flipped within the LargestPossibleRegion of the input image. As such, the LargestPossibleRegion of the output image is the same as the input.

In terms of geometric coordinates, the output origin is such that the image is flipped with respect to the coordinate axes.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Flip An Image Over Specified Axes

See itk::FlipImageFilter for additional documentation.

### Pad an Image by Mirroring

### Synopsis

Pad an image using mirroring over the boundaries

### Results

Fig. 198: Input image

Fig. 199: Output image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMirrorPadImageFilter.h"

int
```

(continues on next page)

(continued from previous page)

```
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  ImageType::SizeType lowerBound;
  lowerBound[0] = 20;
  lowerBound[1] = 30;

  ImageType::SizeType upperBound;
  upperBound[0] = 50;
  upperBound[1] = 40;

  using FilterType = itk::MirrorPadImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetPadLowerBound(lowerBound);
  filter->SetPadUpperBound(upperBound);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Pad an Image With a Constant

### Synopsis

Pad an itk::Image with a given constant

### Results

Fig. 200: Input image

Fig. 201: Output image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkConstantPadImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName> <LowerExtendSize> ";
    std::cerr << "<UpperExtendSize> <ConstantValue>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  ImageType::SizeType lowerExtendRegion;
  lowerExtendRegion.Fill(std::stoi(argv[3]));

  ImageType::SizeType upperExtendRegion;
  upperExtendRegion.Fill(std::stoi(argv[4]));

  using FilterType = itk::ConstantPadImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetPadLowerBound(lowerExtendRegion);
  filter->SetPadUpperBound(upperExtendRegion);
  filter->SetConstant(std::stoi(argv[5]));

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }

  return EXIT_SUCCESS;
}
```

### Pad Image by Wrapping

### Synopsis

Pad an image by wrapping.

### Results

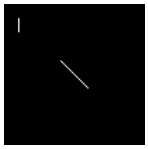


Fig. 202: image.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkWrapPadImageFilter.h"
#include "itkImageRegionIterator.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using WrapPadImageFilterType = itk::WrapPadImageFilter<ImageType, ImageType>;

  ImageType::SizeType lowerBound;
  lowerBound[0] = 20;
  lowerBound[1] = 30;

  ImageType::SizeType upperBound;
  upperBound[0] = 50;
```

(continues on next page)

(continued from previous page)

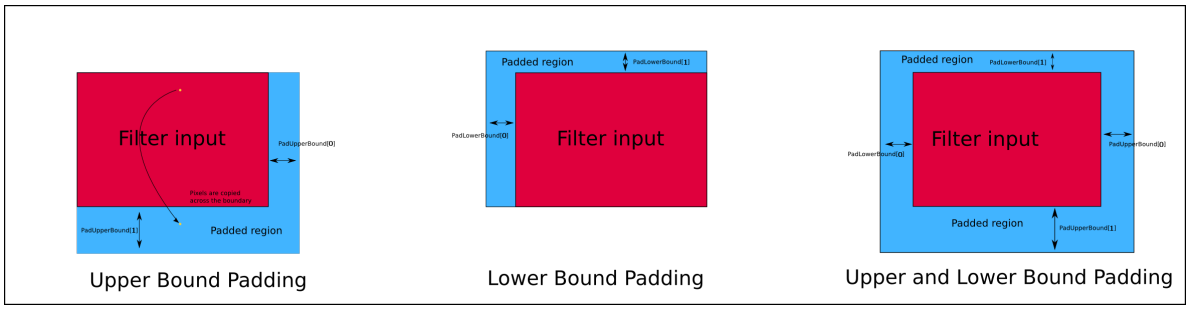
```
  upperBound[1] = 40;

  WrapPadImageFilterType::Pointer padFilter = WrapPadImageFilterType::New();
  padFilter->SetInput(image);
  padFilter->SetPadLowerBound(lowerBound);
  padFilter->SetPadUpperBound(upperBound);
  padFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 100;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();
  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(imageIterator.GetIndex()[0]);
    ++imageIterator;
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class WrapPadImageFilter** : **public** itk::PadImageFilter<*TInputImage*, *TOutputImage*>

Increase the image size by padding with replicants of the input image value.

WrapPadImageFilter changes the image bounds of an image. Added pixels are filled in with a wrapped replica of the input image. For instance, if the output image needs a pixel that is **two pixels to the left of the LargestPossibleRegion** of the input image, the value assigned will be from the pixel **two pixels inside the right boundary of the LargestPossibleRegion**. The image bounds of the output must be specified.




This filter is implemented as a multithreaded filter. It provides a ThreadedGenerateData() method for its implementation.

**See** MirrorPadImageFilter, ConstantPadImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Pad Image By Wrapping

See itk::WrapPadImageFilter for additional documentation.

### Paste Image Into Another One

### Synopsis

Paste one itk::Image into another one

### Results

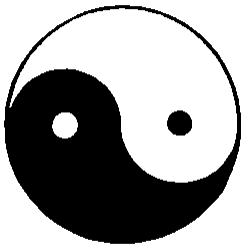


Fig. 203: Source image

Fig. 204: Destination image

Fig. 205: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkPasteImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <SourceFileName> <DestinationFileName> <OutputFileName> <start x>
↪<start y>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * sourceFileName = argv[1];
  const char * destinationFileName = argv[2];
  const char * outputFileName = argv[3];

  int startX = std::stoi(argv[4]);
  int startY = std::stoi(argv[5]);

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::IndexType index;
  index[0] = startX;
  index[1] = startY;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer sourceReader = ReaderType::New();
  sourceReader->SetFileName(sourceFileName);
  sourceReader->Update();

  ReaderType::Pointer destinationReader = ReaderType::New();
  destinationReader->SetFileName(destinationFileName);

  using FilterType = itk::PasteImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetSourceImage(sourceReader->GetOutput());
  filter->SetSourceRegion(sourceReader->GetOutput()->GetLargestPossibleRegion());
  filter->SetDestinationImage(destinationReader->GetOutput());
  filter->SetDestinationIndex(index);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
```

(continues on next page)

(continued from previous page)

```
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TSourceImage** = *TInputImage*, typename **TOutputImage** = *TInputImage*>
**class PasteImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Paste an image (or a constant value) into another image.

PasteImageFilter allows a region in a destination image to be filled with a source image or a constant pixel value. The SetDestinationIndex() method prescribes where in the destination input to start pasting data from the source input. The SetSourceRegion method prescribes the section of the second image to paste into the first. When a constant pixel value is set, the SourceRegion describes the size of the region filled. If the output requested region does not include the SourceRegion after it has been repositioned to DestinationIndex, then the output will just be a copy of the input.

This filter supports running "InPlace" to efficiently reuse the destination image buffer for the output, removing the need to copy the destination pixels to the output.

When the source image has a lower dimension than the destination image then the DestinationSkipAxes parameter specifies which axes in the destination image are set to 1 when copying the region or filling with a constant.

The two inputs and output image will have the same pixel type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Paste Image Into Another One
- Run Image Filter On Region Of Image

See itk::PasteImageFilter for additional documentation.

### Permute Axes of an Image

### Synopsis

switch the axes of an image

### Results

Fig. 206: Input image

Fig. 207: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkPermuteAxesImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::PermuteAxesImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  FilterType::PermuteOrderArrayType order;
  order[0] = 1;
  order[1] = 0;

  filter->SetOrder(order);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

template<typename **TImage**>
**class PermuteAxesImageFilter** : **public** itk::*ImageToImageFilter*<*TImage*, *TImage*>
Permutes the image axes according to a user specified order.

PermuateAxesImageFilter permutes the image axes according to a user specified order. The permutation order is set via method SetOrder( order ) where the input is an array of ImageDimension number of unsigned int. The elements of the array must be a rearrangment of the numbers from 0 to ImageDimension - 1.

The i-th axis of the output image corresponds with the order[i]-th axis of the input image.

The output meta image information (LargestPossibleRegion, spacing, origin) is computed by permuting the corresponding input meta information.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Permute Axes Of An Image

See itk::PermuteAxesImageFilter for additional documentation.

### Process a 2D Slice of a 3D Image

### Synopsis

This example illustrates the common task of extracting a 2D slice from a 3D volume. Perform some processing on that slice and then paste it on an output volume of the same size as the volume from the input.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Process A 2D Slice Of A 3D Image.")
parser.add_argument("input_3D_image")
parser.add_argument("output_3D_image")
parser.add_argument("slice_number", type=int)
args = parser.parse_args()

Dimension = 3
PixelType = itk.ctype("short")
```

(continues on next page)

(continued from previous page)

```
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_3D_image)
reader.Update()
inputImage = reader.GetOutput()

extractFilter = itk.ExtractImageFilter.New(inputImage)
extractFilter.SetDirectionCollapseToSubmatrix()

# set up the extraction region [one slice]
inputRegion = inputImage.GetBufferedRegion()
size = inputRegion.GetSize()
size[2] = 1  # we extract along z direction
start = inputRegion.GetIndex()
sliceNumber = args.slice_number
start[2] = sliceNumber
desiredRegion = inputRegion
desiredRegion.SetSize(size)
desiredRegion.SetIndex(start)

extractFilter.SetExtractionRegion(desiredRegion)
pasteFilter = itk.PasteImageFilter.New(inputImage)
medianFilter = itk.MedianImageFilter.New(extractFilter)
pasteFilter.SetSourceImage(medianFilter.GetOutput())
pasteFilter.SetDestinationImage(inputImage)
pasteFilter.SetDestinationIndex(start)

indexRadius = size
indexRadius[0] = 1  # radius along x
indexRadius[1] = 1  # radius along y
indexRadius[2] = 0  # radius along z
medianFilter.SetRadius(indexRadius)
medianFilter.UpdateLargestPossibleRegion()
medianImage = medianFilter.GetOutput()
pasteFilter.SetSourceRegion(medianImage.GetBufferedRegion())

itk.imwrite(pasteFilter.GetOutput(), args.output_3D_image)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkExtractImageFilter.h"
#include "itkPasteImageFilter.h"
#include "itkMedianImageFilter.h"

int
main(int argc, char ** argv)
{
  // Verify the number of parameters in the command line
  if (argc <= 3)
  {
    std::cerr << "Usage: " << std::endl;
```

(continues on next page)

(continued from previous page)

```
  std::cerr << argv[0] << " input3DImageFile  output3DImageFile " << std::endl;
  std::cerr << " sliceNumber " << std::endl;
  return EXIT_FAILURE;
}

using PixelType = short;

using ImageType = itk::Image<PixelType, 3>;

using ReaderType = itk::ImageFileReader<ImageType>;
using WriterType = itk::ImageFileWriter<ImageType>;

// Here we recover the file names from the command line arguments
const char * inputFilename = argv[1];
const char * outputFilename = argv[2];

ReaderType::Pointer reader = ReaderType::New();
reader->SetFileName(inputFilename);
try
{
  reader->Update();
}
catch (itk::ExceptionObject & err)
{
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
}

WriterType::Pointer writer = WriterType::New();
writer->SetFileName(outputFilename);

using ExtractFilterType = itk::ExtractImageFilter<ImageType, ImageType>;
ExtractFilterType::Pointer extractFilter = ExtractFilterType::New();
extractFilter->SetDirectionCollapseToSubmatrix();

// set up the extraction region [one slice]
const ImageType *     inputImage = reader->GetOutput();
ImageType::RegionType inputRegion = inputImage->GetBufferedRegion();
ImageType::SizeType   size = inputRegion.GetSize();
size[2] = 1; // we extract along z direction
ImageType::IndexType start = inputRegion.GetIndex();
const unsigned int   sliceNumber = std::stoi(argv[3]);
start[2] = sliceNumber;
ImageType::RegionType desiredRegion;
desiredRegion.SetSize(size);
desiredRegion.SetIndex(start);

extractFilter->SetExtractionRegion(desiredRegion);
using PasteFilterType = itk::PasteImageFilter<ImageType>;
PasteFilterType::Pointer pasteFilter = PasteFilterType::New();
using MedianFilterType = itk::MedianImageFilter<ImageType, ImageType>;
MedianFilterType::Pointer medianFilter = MedianFilterType::New();
extractFilter->SetInput(inputImage);
medianFilter->SetInput(extractFilter->GetOutput());
pasteFilter->SetSourceImage(medianFilter->GetOutput());
pasteFilter->SetDestinationImage(inputImage);
```

(continues on next page)

(continued from previous page)

```
  pasteFilter->SetDestinationIndex(start);

  ImageType::SizeType indexRadius;
  indexRadius[0] = 1; // radius along x
  indexRadius[1] = 1; // radius along y
  indexRadius[2] = 0; // radius along z
  medianFilter->SetRadius(indexRadius);
  medianFilter->UpdateLargestPossibleRegion();
  const ImageType * medianImage = medianFilter->GetOutput();
  pasteFilter->SetSourceRegion(medianImage->GetBufferedRegion());
  writer->SetInput(pasteFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ExtractImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Decrease the image size by cropping the image to the selected region bounds.

ExtractImageFilter changes the image boundary of an image by removing pixels outside the target region. The target region must be specified.

ExtractImageFilter also collapses dimensions so that the input image may have more dimensions than the output image (i.e. 4-D input image to a 3-D output image). To specify what dimensions to collapse, the ExtractionRegion must be specified. For any dimension dim where ExtractionRegion.Size[dim] = 0, that dimension is collapsed. The index to collapse on is specified by ExtractionRegion.Index[dim]. For example, we have a image 4D = a 4x4x4x4 image, and we want to get a 3D image, 3D = a 4x4x4 image, specified as [x,y,z,2] from 4D (i.e. the 3rd "time" slice from 4D). The ExtractionRegion.Size = [4,4,4,0] and ExtractionRegion.Index = [0,0,0,2].

The number of dimension in ExtractionRegion.Size and Index must = InputImageDimension. The number of non-zero dimensions in ExtractionRegion.Size must = OutputImageDimension.

The output image produced by this filter will have the same origin as the input image, while the ImageRegion of the output image will start at the starting index value provided in the ExtractRegion parameter. If you are looking for a filter that will re-compute the origin of the output image, and provide an output image region whose index is set to zeros, then you may want to use the RegionOfInterestImageFilter. The output spacing is is simply the collapsed version of the input spacing.

Determining the direction of the collapsed output image from an larger dimensional input space is an ill defined problem in general. It is required that the application developer select the desired transformation strategy for collapsing direction cosines. It is REQUIRED that a strategy be explicitly requested (i.e. there is no working default). Direction Collapsing Strategies: 1) DirectionCollapseToUnknown(); This is the default and the filter can not run when this is set. The reason is to explicitly force the application developer to define their desired

behavior. 1) DirectionCollapseToIdentity(); Output has identity direction no matter what 2) DirectionCollapseToSubmatrix(); Output direction is the sub-matrix if it is positive definite, else throw an exception.

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Note** This filter is derived from InPlaceImageFilter. When the input to this filter matched the output requested region, like with streaming filter for input, then setting this filter to run in-place will result in no copying of the bulk pixel data.

**See** CropImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Crop Image By Specifying Region

Subclassed by itk::CropImageFilter< TInputImage, TOutputImage >

See itk::ExtractImageFilter for additional documentation.

template<typename **TInputImage**, typename **TSourceImage** = *TInputImage*, typename **TOutputImage** = *TInputImage*>
**class PasteImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Paste an image (or a constant value) into another image.

PasteImageFilter allows a region in a destination image to be filled with a source image or a constant pixel value. The SetDestinationIndex() method prescribes where in the destination input to start pasting data from the source input. The SetSourceRegion method prescribes the section of the second image to paste into the first. When a constant pixel value is set, the SourceRegion describes the size of the region filled. If the output requested region does not include the SourceRegion after it has been repositioned to DestinationIndex, then the output will just be a copy of the input.

This filter supports running "InPlace" to efficiently reuse the destination image buffer for the output, removing the need to copy the destination pixels to the output.

When the source image has a lower dimension than the destination image then the DestinationSkipAxes parameter specifies which axes in the destination image are set to 1 when copying the region or filling with a constant.

The two inputs and output image will have the same pixel type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Paste Image Into Another One
- Run Image Filter On Region Of Image

See itk::PasteImageFilter for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class MedianImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>

Applies a median filter to an image.

Computes an image where a given pixel is the median value of the the pixels in a neighborhood about the corresponding input pixel.

A median filter is one of the family of nonlinear filters. It is used to smooth an image without being biased by outliers or shot noise.

This filter requires that the input pixel type provides an operator<() (LessThan Comparable).

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Median Filter Of An Image
- Median Filter Of An RGB Image

See itk::MedianImageFilter for additional documentation.

### Resample an Image

### Synopsis

Resample an image.

### Results

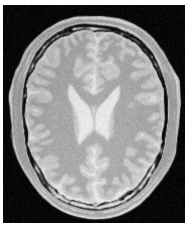


Fig. 208: Input image

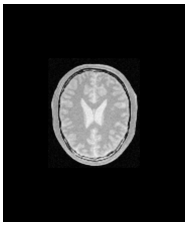


Fig. 209: Output image

### Code

### Python

```python
#!/usr/bin/env python

import sys
import itk

if len(sys.argv) != 4:
    print("Usage: " + sys.argv[0] + " <input_image> <output_image> <scale>")
    sys.exit(1)

input_file_name = sys.argv[1]
output_file_name = sys.argv[2]
scale = float(sys.argv[3])

input_image = itk.imread(input_file_name)
input_size = itk.size(input_image)
input_spacing = itk.spacing(input_image)
input_origin = itk.origin(input_image)
Dimension = input_image.GetImageDimension()

# We will scale the objects in the image by the factor `scale`; that is they
# will be shrunk (scale < 1.0) or enlarged (scale > 1.0).  However, the number
# of pixels for each dimension of the output image will equal the corresponding
# number of pixels in the input image, with cropping or padding as necessary.
# Furthermore, the physical distance between adjacent pixels will be the same
# in the input and the output images.  In contrast, if you want to change the
# resolution of the image without changing the represented physical size of the
# objects in the image, omit the transform and instead supply:
#
# output_size = [int(input_size[d] * scale) for d in range(Dimension)]
# output_spacing = [input_spacing[d] / scale for d in range(Dimension)]
# output_origin = [input_origin[d] + 0.5 * (output_spacing[d] - input_spacing[d])
#                  for d in range(Dimension)]

output_size = input_size
output_spacing = input_spacing
output_origin = input_origin
scale_transform = itk.ScaleTransform[itk.D, Dimension].New()
scale_transform_parameters = scale_transform.GetParameters()
for i in range(len(scale_transform_parameters)):
    scale_transform_parameters[i] = scale
scale_transform_center = [float(int(s / 2)) for s in input_size]
scale_transform.SetParameters(scale_transform_parameters)
scale_transform.SetCenter(scale_transform_center)

interpolator = itk.LinearInterpolateImageFunction.New(input_image)

resampled = itk.resample_image_filter(
    input_image,
    transform=scale_transform,
    interpolator=interpolator,
    size=output_size,
    output_spacing=output_spacing,
    output_origin=output_origin,
```

(continues on next page)

(continued from previous page)

```
)

itk.imwrite(resampled, output_file_name)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkResampleImageFilter.h"
#include "itkScaleTransform.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <scale>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const float  scale = std::stod(argv[3]);

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
  using ScalarType = double;
  using IndexValueType = typename itk::Index<Dimension>::IndexValueType;

  using ReaderType = itk::ImageFileReader<ImageType>;
  typename ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  reader->Update();

  const typename ImageType::Pointer     inputImage = reader->GetOutput();
  const typename ImageType::RegionType  inputRegion = inputImage->
↪GetLargestPossibleRegion();
  const typename ImageType::SizeType    inputSize = inputRegion.GetSize();
  const typename ImageType::SpacingType inputSpacing = inputImage->GetSpacing();
  const typename ImageType::PointType   inputOrigin = inputImage->GetOrigin();

  /*
   * We will scale the objects in the image by the factor `scale`; that is they
   * will be shrunk (scale < 1.0) or enlarged (scale > 1.0).  However, the
   * number of pixels for each dimension of the output image will equal the
   * corresponding number of pixels in the input image, with padding (if
   * shrunk) or cropping (if enlarged) as necessary.  Furthermore, the physical
   * distance between adjacent pixels will be the same in the input and the
```

(continues on next page)

(continued from previous page)

```
   * output images.  In contrast, if you want to change the resolution of the
   * image without changing the represented physical size of the objects in the
   * image, omit the transform and instead use:
   *
   *   outputSize[d] = inputSize[d] * scale;
   *   outputSpacing[d] = inputSpacing[d] / scale;
   *   outputOrigin[d] = inputOrigin[d] + 0.5 * (outputSpacing[d] - inputSpacing[d]);
   *
   * in the loop over dimensions.
   */

  typename ImageType::SizeType    outputSize = inputSize;
  typename ImageType::SpacingType outputSpacing = inputSpacing;
  typename ImageType::PointType   outputOrigin = inputOrigin;

  using ScaleTransformType = itk::ScaleTransform<ScalarType, Dimension>;
  typename ScaleTransformType::Pointer scaleTransform = ScaleTransformType::New();

  typename ScaleTransformType::ParametersType scaleTransformParameters =␣
↪scaleTransform->GetParameters();
  itk::Point<ScalarType, Dimension>           scaleTransformCenter;
  for (unsigned int d = 0; d < Dimension; ++d)
  {
    scaleTransformParameters[d] = scale;
    scaleTransformCenter[d] = static_cast<ScalarType>(static_cast<IndexValueType>
↪(inputSize[d] / 2));
  }
  scaleTransform->SetParameters(scaleTransformParameters);
  scaleTransform->SetCenter(scaleTransformCenter);

  using LinearInterpolatorType = itk::LinearInterpolateImageFunction<ImageType,␣
↪ScalarType>;
  typename LinearInterpolatorType::Pointer interpolator = LinearInterpolatorType::
↪New();

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  typename ResampleFilterType::Pointer resampleFilter = ResampleFilterType::New();

  resampleFilter->SetInput(inputImage);
  resampleFilter->SetTransform(scaleTransform);
  resampleFilter->SetInterpolator(interpolator);
  resampleFilter->SetSize(outputSize);
  resampleFilter->SetOutputSpacing(outputSpacing);
  resampleFilter->SetOutputOrigin(outputOrigin);

  using WriterType = itk::ImageFileWriter<ImageType>;
  typename WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(resampleFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TInterpolatorPrecisionType** = double, typenan
**class ResampleImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Resample an image via a coordinate transform.

ResampleImageFilter resamples an existing image through some coordinate transform, interpolating via some image function. The class is templated over the types of the input and output images.

Note that the choice of interpolator function can be important. This function is set via SetInterpolator(). The default is LinearInterpolateImageFunction<InputImageType, TInterpolatorPrecisionType>, which is reasonable for ordinary medical images. However, some synthetic images have pixels drawn from a finite prescribed set. An example would be a mask indicating the segmentation of a brain into a small number of tissue types. For such an image, one does not want to interpolate between different pixel values, and so NearestNeighborInterpolateImageFunction< InputImageType, TCoordRep > would be a better choice.

If an sample is taken from outside the image domain, the default behavior is to use a default pixel value. If different behavior is desired, an extrapolator function can be set with SetExtrapolator().

Output information (spacing, size and direction) for the output image should be set. This information has the normal defaults of unit spacing, zero origin and identity direction. Optionally, the output information can be obtained from a reference image. If the reference image is provided and UseReferenceImage is On, then the spacing, origin and direction of the reference image will be used.

Since this filter produces an image which is a different size than its input, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Warning** For multithreading, the TransformPoint method of the user-designated coordinate transform must be threadsafe.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Image
- Upsampling An Image
- Resample An Image

See itk::ResampleImageFilter for additional documentation.

### Resample a Scalar Image

### Synopsis

Resample a scalar image.

### Results

Fig. 210: Input image

Fig. 211: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Resample A Scalar Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("size_x", type=int)
parser.add_argument("size_y", type=int)
args = parser.parse_args()

output_size = [args.size_x, args.size_y]
```

(continues on next page)

(continued from previous page)

```
input_image = itk.imread(args.input_image)

input_spacing = itk.spacing(input_image)
input_size = itk.size(input_image)
dimension = input_image.GetImageDimension()
output_spacing = [
    input_spacing[dim] * input_size[dim] / output_size[dim] for dim in␣
↪range(dimension)
]

transform = itk.IdentityTransform[itk.D, dimension].New()

output_image = itk.resample_image_filter(
    input_image,
    size=output_size,
    output_spacing=output_spacing,
    output_origin=itk.origin(input_image),
    transform=transform,
)

itk.imwrite(output_image, args.output_image)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkResampleImageFilter.h"
#include "itkIdentityTransform.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <size X> <size Y>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  ImageType::SizeType outputSize;

  for (unsigned int dim = 0, k = 3; dim < Dimension; dim++)
  {
```

(continues on next page)

(continued from previous page)

```
    outputSize[dim] = std::stoi(argv[k++]);
  }

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  reader->UpdateOutputInformation();

  ImageType::Pointer inputImage = reader->GetOutput();

  ImageType::SizeType inputSize = inputImage->GetLargestPossibleRegion().GetSize();
  std::cout << "Input Size: " << inputSize << std::endl;

  ImageType::SpacingType inputSpacing = inputImage->GetSpacing();
  std::cout << "Input Spacing: " << inputSpacing << std::endl;

  ImageType::SpacingType outputSpacing;

  for (unsigned int dim = 0; dim < Dimension; dim++)
  {
    outputSpacing[dim] = static_cast<double>(inputSpacing[dim]) * static_cast<double>
↪(inputSize[dim]) /
                         static_cast<double>(outputSize[dim]);
  }

  std::cout << "Output Size: " << outputSize << std::endl;
  std::cout << "Output Spacing: " << outputSpacing << std::endl;

  using TransformPrecisionType = double;
  using TransformType = itk::IdentityTransform<TransformPrecisionType, Dimension>;
  using FilterType = itk::ResampleImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(inputImage);
  filter->SetSize(outputSize);
  filter->SetOutputSpacing(outputSpacing);
  filter->SetOutputOrigin(inputImage->GetOrigin());
  filter->SetTransform(TransformType::New());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TInterpolatorPrecisionType** = double, typenan
**class ResampleImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Resample an image via a coordinate transform.

ResampleImageFilter resamples an existing image through some coordinate transform, interpolating via some image function. The class is templated over the types of the input and output images.

Note that the choice of interpolator function can be important. This function is set via SetInterpolator(). The default is LinearInterpolateImageFunction<InputImageType, TInterpolatorPrecisionType>, which is reasonable for ordinary medical images. However, some synthetic images have pixels drawn from a finite prescribed set. An example would be a mask indicating the segmentation of a brain into a small number of tissue types. For such an image, one does not want to interpolate between different pixel values, and so NearestNeighborInterpolateImageFunction< InputImageType, TCoordRep > would be a better choice.

If an sample is taken from outside the image domain, the default behavior is to use a default pixel value. If different behavior is desired, an extrapolator function can be set with SetExtrapolator().

Output information (spacing, size and direction) for the output image should be set. This information has the normal defaults of unit spacing, zero origin and identity direction. Optionally, the output information can be obtained from a reference image. If the reference image is provided and UseReferenceImage is On, then the spacing, origin and direction of the reference image will be used.

Since this filter produces an image which is a different size than its input, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Warning** For multithreading, the TransformPoint method of the user-designated coordinate transform must be threadsafe.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Image
- Upsampling An Image
- Resample An Image

See itk::ResampleImageFilter for additional documentation.

### Resample a Vector Image

### Synopsis

Linearly interpolate a vector image.

The Python wrapping for the LinearInterpolateImageFunction using vector images was added in ITK 4.7.0.

### Results

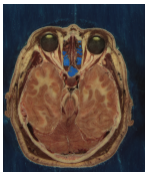


Fig. 212: Input image

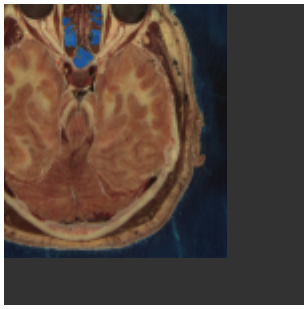


Fig. 213: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Resample A Vector Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

PixelType = itk.RGBPixel[itk.UC]

input_image = itk.imread(args.input_image, pixel_type=PixelType)

ImageType = type(input_image)
interpolator = itk.LinearInterpolateImageFunction[ImageType, itk.D].New()

Dimension = input_image.GetImageDimension()
transform = itk.IdentityTransform[itk.D, Dimension].New()

output_image = itk.resample_image_filter(
    input_image,
    interpolator=interpolator,
    transform=transform,
    default_pixel_value=[50, 50, 50],
```

(continues on next page)

(continued from previous page)

```
    output_spacing=[0.5, 0.5],
    output_origin=[30, 40],
    size=[300, 300],
)

itk.imwrite(output_image, args.output_image)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkResampleImageFilter.h"
#include "itkIdentityTransform.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkRGBPixel.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelComponentType = unsigned char;
  using PixelType = itk::RGBPixel<PixelComponentType>;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::ResampleImageFilter<ImageType, ImageType>;

  FilterType::Pointer filter = FilterType::New();

  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;

  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  filter->SetInterpolator(interpolator);

  using TransformType = itk::IdentityTransform<double, Dimension>;
  TransformType::Pointer transform = TransformType::New();
```

(continues on next page)

(continued from previous page)

```
  filter->SetTransform(transform);

  PixelType defaultValue;
  defaultValue.Fill(50);
  filter->SetDefaultPixelValue(defaultValue);

  ImageType::SpacingType spacing;
  spacing[0] = .5; // pixel spacing in millimeters along X
  spacing[1] = .5; // pixel spacing in millimeters along Y

  filter->SetOutputSpacing(spacing);

  ImageType::PointType origin;
  origin[0] = 30.0; // X space coordinate of origin
  origin[1] = 40.0; // Y space coordinate of origin
  filter->SetOutputOrigin(origin);

  ImageType::DirectionType direction;
  direction.SetIdentity();
  filter->SetOutputDirection(direction);

  ImageType::SizeType size;

  size[0] = 300; // number of pixels along X
  size[1] = 300; // number of pixels along Y

  filter->SetSize(size);
  filter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TInterpolatorPrecisionType** = double, typenan
**class ResampleImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Resample an image via a coordinate transform.

ResampleImageFilter resamples an existing image through some coordinate transform, interpolating via some image function. The class is templated over the types of the input and output images.

Note that the choice of interpolator function can be important. This function is set via SetInterpolator(). The default is LinearInterpolateImageFunction<InputImageType, TInterpolatorPrecisionType>, which is reasonable for ordinary medical images. However, some synthetic images have pixels drawn from a finite prescribed set. An example would be a mask indicating the segmentation of a brain into a small number of tissue types. For such an image, one does not want to interpolate between different pixel values, and so NearestNeighborInterpolateImageFunction< InputImageType, TCoordRep > would be a better choice.

If an sample is taken from outside the image domain, the default behavior is to use a default pixel value. If different behavior is desired, an extrapolator function can be set with SetExtrapolator().

Output information (spacing, size and direction) for the output image should be set. This information has the normal defaults of unit spacing, zero origin and identity direction. Optionally, the output information can be obtained from a reference image. If the reference image is provided and UseReferenceImage is On, then the spacing, origin and direction of the reference image will be used.

Since this filter produces an image which is a different size than its input, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Warning** For multithreading, the TransformPoint method of the user-designated coordinate transform must be threadsafe.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Image
- Upsampling An Image
- Resample An Image

See itk::ResampleImageFilter for additional documentation.

### Run Image Filter on Region of Image

### Synopsis

Run an image filter on a region of an image.

### Results

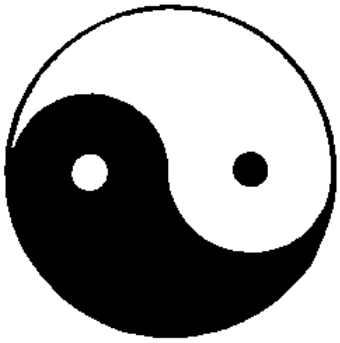


Fig. 214: Input image

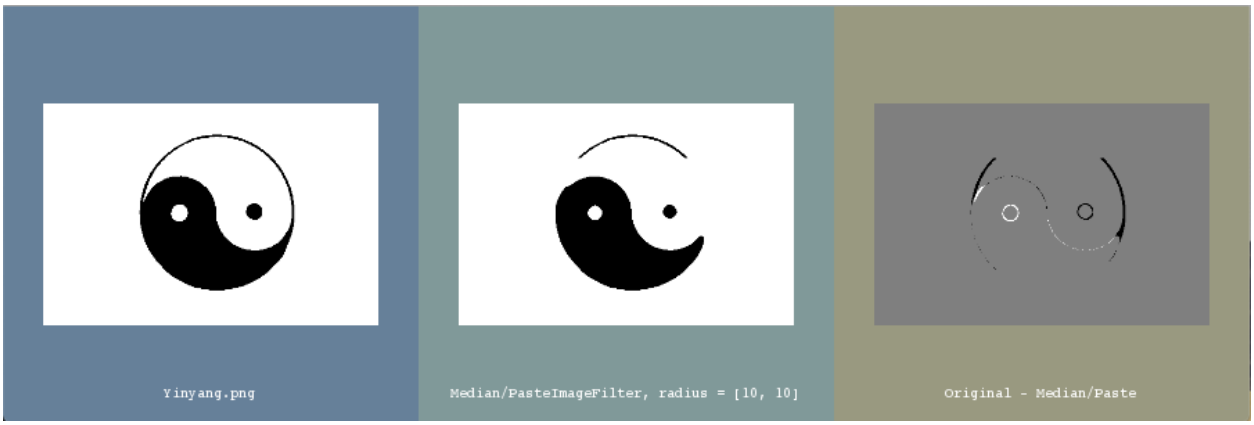



Fig. 215: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkMedianImageFilter.h"
#include "itkPasteImageFilter.h"
#include "itkSubtractImageFilter.h"

#include "itksys/SystemTools.hxx"

#include <sstream>

#ifdef ENABLE_QUICKVIEW
```

(continues on next page)

(continued from previous page)

```
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " InputImageFile [radius]" << std::endl;
    return EXIT_FAILURE;
  }
  std::string inputFilename = argv[1];

  // Setup types
  using ImageType = itk::Image<float, 2>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  using FilterType = itk::MedianImageFilter<ImageType, ImageType>;
  using SubtractType = itk::SubtractImageFilter<ImageType>;
  using PasteType = itk::PasteImageFilter<ImageType, ImageType>;

  // Create and setup a reader
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);

  // Create and setup a median filter
  FilterType::Pointer       medianFilter = FilterType::New();
  FilterType::InputSizeType radius;
  radius.Fill(2);
  if (argc > 2)
  {
    radius.Fill(atoi(argv[2]));
  }

  reader->Update();

  itk::Size<2> processSize;
  processSize[0] = reader->GetOutput()->GetLargestPossibleRegion().GetSize()[0] / 2;
  processSize[1] = reader->GetOutput()->GetLargestPossibleRegion().GetSize()[1] / 2;

  itk::Index<2> processIndex;
  processIndex[0] = processSize[0] / 2;
  processIndex[1] = processSize[1] / 2;

  itk::ImageRegion<2> processRegion(processIndex, processSize);

  medianFilter->SetRadius(radius);
  medianFilter->SetInput(reader->GetOutput());
  medianFilter->GetOutput()->SetRequestedRegion(processRegion);

  PasteType::Pointer pasteFilter = PasteType::New();

  pasteFilter->SetSourceImage(medianFilter->GetOutput());
  pasteFilter->SetSourceRegion(medianFilter->GetOutput()->GetRequestedRegion());
  pasteFilter->SetDestinationImage(reader->GetOutput());
  pasteFilter->SetDestinationIndex(processIndex);
```

(continues on next page)

(continued from previous page)

```
  SubtractType::Pointer diff = SubtractType::New();
  diff->SetInput1(reader->GetOutput());
  diff->SetInput2(pasteFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFilename));

  std::stringstream desc;
  desc << "Median/PasteImageFilter, radius = " << radius;
  viewer.AddImage(pasteFilter->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Original - Median/Paste";
  viewer.AddImage(diff->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TSourceImage** = *TInputImage*, typename **TOutputImage** = *TInputImage*>
**class PasteImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Paste an image (or a constant value) into another image.

PasteImageFilter allows a region in a destination image to be filled with a source image or a constant pixel value. The SetDestinationIndex() method prescribes where in the destination input to start pasting data from the source input. The SetSourceRegion method prescribes the section of the second image to paste into the first. When a constant pixel value is set, the SourceRegion describes the size of the region filled. If the output requested region does not include the SourceRegion after it has been repositioned to DestinationIndex, then the output will just be a copy of the input.

This filter supports running "InPlace" to efficiently reuse the destination image buffer for the output, removing the need to copy the destination pixels to the output.

When the source image has a lower dimension than the destination image then the DestinationSkipAxes parameter specifies which axes in the destination image are set to 1 when copying the region or filling with a constant.

The two inputs and output image will have the same pixel type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Paste Image Into Another One
- Run Image Filter On Region Of Image

See itk::PasteImageFilter for additional documentation.

### Shrink Image

### Synopsis

Shrink an image.

### Results

Output:

```
Original size: [100, 100]
New size: [50, 33]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkShrinkImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  std::cout << "Original size: " << image->GetLargestPossibleRegion().GetSize() <<␣
↪std::endl;

  using ShrinkImageFilterType = itk::ShrinkImageFilter<ImageType, ImageType>;

  ShrinkImageFilterType::Pointer shrinkFilter = ShrinkImageFilterType::New();
  shrinkFilter->SetInput(image);
  shrinkFilter->SetShrinkFactor(0, 2); // shrink the first dimension by a factor of 2␣
↪(i.e. 100 gets changed to 50)
  shrinkFilter->SetShrinkFactor(1, 3); // shrink the second dimension by a factor of␣
↪3 (i.e. 100 gets changed to 33)
  shrinkFilter->Update();

  std::cout << "New size: " << shrinkFilter->GetOutput()->GetLargestPossibleRegion().
↪GetSize() << std::endl;


  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a white square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 30; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ShrinkImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Reduce the size of an image by an integer factor in each dimension.

ShrinkImageFilter reduces the size of an image by an integer factor in each dimension. The algorithm implemented is a simple subsample. The output image size in each dimension is given by:

outputSize[j] = max( std::floor(inputSize[j]/shrinkFactor[j]), 1 );

NOTE: The physical centers of the input and output will be the same. Because of this, the Origin of the output may not be the same as the Origin of the input. Since this filter produces an image which is a different resolution, origin and with different pixel spacing than its input image, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Shrink Image

See itk::ShrinkImageFilter for additional documentation.

### Stack 2D Images Into 3D Image

### Synopsis

TileImageFilter

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkTileImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"

int
main(int argc, char * argv[])
{

  using PixelType = unsigned char;
  constexpr unsigned int InputImageDimension = 2;
  constexpr unsigned int OutputImageDimension = 3;

  using InputImageType = itk::Image<PixelType, InputImageDimension>;
  using OutputImageType = itk::Image<PixelType, OutputImageDimension>;

  using ImageReaderType = itk::ImageFileReader<InputImageType>;

  using TilerType = itk::TileImageFilter<InputImageType, OutputImageType>;

  using WriterType = itk::ImageFileWriter<OutputImageType>;

  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "input1 input2 ... inputn output" << std::endl;
    return EXIT_FAILURE;
  }

  TilerType::Pointer tiler = TilerType::New();

  itk::FixedArray<unsigned int, OutputImageDimension> layout;

  layout[0] = 1;
  layout[1] = 1;
```

(continues on next page)

(continued from previous page)

```
  layout[2] = 0;

  tiler->SetLayout(layout);

  unsigned int inputImageNumber = 0;

  ImageReaderType::Pointer reader = ImageReaderType::New();

  InputImageType::Pointer inputImageTile;

  for (int i = 1; i < argc - 1; i++)
  {
    reader->SetFileName(argv[i]);
    reader->UpdateLargestPossibleRegion();
    inputImageTile = reader->GetOutput();
    inputImageTile->DisconnectPipeline();
    tiler->SetInput(inputImageNumber++, inputImageTile);
  }

  PixelType filler = 128;

  tiler->SetDefaultPixelValue(filler);

  tiler->Update();

  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(tiler->GetOutput());
  writer->SetFileName(argv[argc - 1]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & excp)
  {
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class TileImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Tile multiple input images into a single output image.

This filter will tile multiple images using a user-specified layout. The tile sizes will be large enough to accommodate the largest image for each tile. The layout is specified with the SetLayout method. The layout has the same dimension as the output image. If all entries of the layout are positive, the tiled output will contain the exact number of tiles. If the layout contains a 0 in the last dimension, the filter will compute a size that will accommodate all of the images. Empty tiles are filled with the value specified with the SetDefault value method. The input images must have a dimension less than or equal to the output image. The output image have a larger dimension than the input images. This filter can be used to create a volume from a series of inputs by specifying

a layout of 1,1,0.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Stack 2D Images Into 3D Image
- Tile Images Side By Side

See itk::TileImageFilter for additional documentation.

### Tile Images Side by Side

### Synopsis

Tile multiple images side by side.

### Results

Fig. 216: Gourds.png

Fig. 217: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkTileImageFilter.h"

int
main(int argc, char * argv[])
{
  // Verify arguments
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "input1 input2 output" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse arguments
  std::string input1FileName = argv[1];
  std::string input2FileName = argv[2];
  std::string outputFileName = argv[3];

  // Output arguments
  std::cout << "input1FileName " << input1FileName << std::endl;
  std::cout << "input2FileName " << input2FileName << std::endl;
  ;
  std::cout << "outputFileName " << outputFileName << std::endl;
  ;

  using ImageType = itk::Image<unsigned char, 2>;

  // Read images
  using ImageReaderType = itk::ImageFileReader<ImageType>;
  ImageReaderType::Pointer reader1 = ImageReaderType::New();
```

(continues on next page)

(continued from previous page)

```
  reader1->SetFileName(input1FileName);
  reader1->Update();

  ImageReaderType::Pointer reader2 = ImageReaderType::New();
  reader2->SetFileName(input2FileName);
  reader2->Update();

  // Tile the images side-by-side
  using TileFilterType = itk::TileImageFilter<ImageType, ImageType>;

  TileFilterType::Pointer tileFilter = TileFilterType::New();

  itk::FixedArray<unsigned int, 2> layout;

  layout[0] = 2;
  layout[1] = 0;

  tileFilter->SetLayout(layout);

  tileFilter->SetInput(0, reader1->GetOutput());
  tileFilter->SetInput(1, reader2->GetOutput());

  // Set the value of output pixels which are created by mismatched size input images.
  // If the two images are the same height, this will not be used.
  unsigned char fillerValue = 128;
  tileFilter->SetDefaultPixelValue(fillerValue);

  tileFilter->Update();

  // Write the output image
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(tileFilter->GetOutput());
  writer->SetFileName(outputFileName);
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class TileImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Tile multiple input images into a single output image.

This filter will tile multiple images using a user-specified layout. The tile sizes will be large enough to accommodate the largest image for each tile. The layout is specified with the SetLayout method. The layout has the same dimension as the output image. If all entries of the layout are positive, the tiled output will contain the exact number of tiles. If the layout contains a 0 in the last dimension, the filter will compute a size that will accommodate all of the images. Empty tiles are filled with the value specified with the SetDefault value method. The input images must have a dimension less than or equal to the output image. The output image have a larger dimension than the input images. This filter can be used to create a volume from a series of inputs by specifying a layout of 1,1,0.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Stack 2D Images Into 3D Image
- Tile Images Side By Side

See itk::TileImageFilter for additional documentation.

### Upsample an Image

### Synopsis

Upsample an image.

### Results

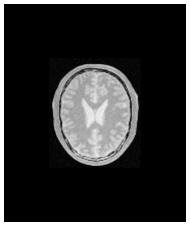


Fig. 218: Input image

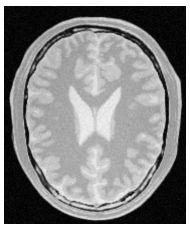


Fig. 219: Output image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkIdentityTransform.h"
#include "itkBSplineInterpolateImageFunction.h"
#include "itkResampleImageFilter.h"

#include <iostream>
```

(continues on next page)

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl
              << argv[0] << " inputImageFile outputImageFile nNewWidth nNewHeight" <<␣
↪std::endl;

    return EXIT_FAILURE;
  }

  // Typedef's for pixel, image, reader and writer types
  using T_InputPixel = unsigned char;

  // Doesn't work for RGB pixels
  // using T_OutputPixel = unsigned char;
  // using T_InputPixel = itk::CovariantVector<unsigned char, 3>;
  // using T_OutputPixel = itk::CovariantVector<unsigned char, 3>;

  using T_Image = itk::Image<T_InputPixel, 2>;
  using T_Reader = itk::ImageFileReader<T_Image>;

  using T_WritePixel = unsigned char;
  using T_WriteImage = itk::Image<T_WritePixel, 2>;
  using T_Writer = itk::ImageFileWriter<T_WriteImage>;

  // Typedefs for the different (numerous!) elements of the "resampling"

  // Identity transform.
  // We don't want any transform on our image except rescaling which is not
  // specified by a transform but by the input/output spacing as we will see
  // later.
  // So no transform will be specified.
  using T_Transform = itk::IdentityTransform<double, 2>;

  // If ITK resampler determines there is something to interpolate which is
  // usually the case when upscaling (!) then we must specify the interpolation
  // algorithm. In our case, we want bicubic interpolation. One way to implement
  // it is with a third order b-spline. So the type is specified here and the
  // order will be specified with a method call later on.
  using T_Interpolator = itk::BSplineInterpolateImageFunction<T_Image, double, double>
↪;

  // The resampler type itself.
  using T_ResampleFilter = itk::ResampleImageFilter<T_Image, T_Image>;


  // Prepare the reader and update it right away to know the sizes beforehand.

  T_Reader::Pointer pReader = T_Reader::New();
  pReader->SetFileName(argv[1]);
  pReader->Update();

  // Prepare the resampler.

  // Instantiate the transform and specify it should be the id transform.
```

(continues on next page)

(continued from previous page)

```
  T_Transform::Pointer _pTransform = T_Transform::New();
  _pTransform->SetIdentity();

  // Instantiate the b-spline interpolator and set it as the third order
  // for bicubic.
  T_Interpolator::Pointer _pInterpolator = T_Interpolator::New();
  _pInterpolator->SetSplineOrder(3);

  // Instantiate the resampler. Wire in the transform and the interpolator.
  T_ResampleFilter::Pointer _pResizeFilter = T_ResampleFilter::New();
  _pResizeFilter->SetTransform(_pTransform);
  _pResizeFilter->SetInterpolator(_pInterpolator);

  // Set the output origin. You may shift the original image "inside" the
  // new image size by specifying something else than 0.0, 0.0 here.

  const double vfOutputOrigin[2] = { 0.0, 0.0 };
  _pResizeFilter->SetOutputOrigin(vfOutputOrigin);

  //     Compute and set the output spacing
  //     Compute the output spacing from input spacing and old and new sizes.
  //
  //     The computation must be so that the following holds:
  //
  //     new width         old x spacing
  //     ----------   =   ---------------
  //     old width         new x spacing
  //
  //
  //     new height         old y spacing
  //    ------------  =   ---------------
  //     old height         new y spacing
  //
  //     So either we specify new height and width and compute new spacings (as
  //     we do here) or we specify new spacing and compute new height and width
  //     and computations that follows need to be modified a little (as it is
  //     done at step 2 there:
  //       http://itk.org/Wiki/ITK/Examples/DICOM/ResampleDICOM)
  //
  unsigned int nNewWidth = std::stoi(argv[3]);
  unsigned int nNewHeight = std::stoi(argv[4]);

  // Fetch original image size.
  const T_Image::RegionType & inputRegion = pReader->GetOutput()->
↪GetLargestPossibleRegion();
  const T_Image::SizeType &   vnInputSize = inputRegion.GetSize();
  unsigned int                nOldWidth = vnInputSize[0];
  unsigned int                nOldHeight = vnInputSize[1];

  // Fetch original image spacing.
  const T_Image::SpacingType & vfInputSpacing = pReader->GetOutput()->GetSpacing();
  // Will be {1.0, 1.0} in the usual
  // case.

  double vfOutputSpacing[2];
  vfOutputSpacing[0] = vfInputSpacing[0] * (double)nOldWidth / (double)nNewWidth;
  vfOutputSpacing[1] = vfInputSpacing[1] * (double)nOldHeight / (double)nNewHeight;
```

(continues on next page)

(continued from previous page)

```
  // Set the output spacing. If you comment out the following line, the original
  // image will be simply put in the upper left corner of the new image without
  // any scaling.
  _pResizeFilter->SetOutputSpacing(vfOutputSpacing);

  // Set the output size as specified on the command line.

  itk::Size<2> vnOutputSize = { { nNewWidth, nNewHeight } };
  _pResizeFilter->SetSize(vnOutputSize);

  // Specify the input.

  _pResizeFilter->SetInput(pReader->GetOutput());

  // Write the result
  T_Writer::Pointer pWriter = T_Writer::New();
  pWriter->SetFileName(argv[2]);
  pWriter->SetInput(_pResizeFilter->GetOutput());
  pWriter->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TInterpolatorPrecisionType** = double, typenan
**class ResampleImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Resample an image via a coordinate transform.

ResampleImageFilter resamples an existing image through some coordinate transform, interpolating via some image function. The class is templated over the types of the input and output images.

Note that the choice of interpolator function can be important. This function is set via SetInterpolator(). The default is LinearInterpolateImageFunction<InputImageType, TInterpolatorPrecisionType>, which is reasonable for ordinary medical images. However, some synthetic images have pixels drawn from a finite prescribed set. An example would be a mask indicating the segmentation of a brain into a small number of tissue types. For such an image, one does not want to interpolate between different pixel values, and so NearestNeighborInterpolateImageFunction< InputImageType, TCoordRep > would be a better choice.

If an sample is taken from outside the image domain, the default behavior is to use a default pixel value. If different behavior is desired, an extrapolator function can be set with SetExtrapolator().

Output information (spacing, size and direction) for the output image should be set. This information has the normal defaults of unit spacing, zero origin and identity direction. Optionally, the output information can be obtained from a reference image. If the reference image is provided and UseReferenceImage is On, then the spacing, origin and direction of the reference image will be used.

Since this filter produces an image which is a different size than its input, it needs to override several of the methods defined in ProcessObject in order to properly manage the pipeline execution model. In particular, this filter overrides ProcessObject::GenerateInputRequestedRegion() and ProcessObject::GenerateOutputInformation().

This filter is implemented as a multithreaded filter. It provides a DynamicThreadedGenerateData() method for its implementation.

**Warning** For multithreading, the TransformPoint method of the user-designated coordinate transform must be threadsafe.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Image
- Upsampling An Image
- Resample An Image

See itk::ResampleImageFilter for additional documentation.

template<typename **TImageType**, typename **TCoordRep** = double, typename **TCoefficientType** = double>
**class BSplineInterpolateImageFunction** : **public** itk::InterpolateImageFunction<*TImageType*, *TCoordRep*>

Evaluates the B-Spline interpolation of an image. Spline order may be from 0 to 5.

This class defines N-Dimension B-Spline transformation. It is based on:

```
[1] M. Unser,
    "Splines: A Perfect Fit for Signal and Image Processing,"
    IEEE Signal Processing Magazine, vol. 16, no. 6, pp. 22-38,
    November 1999.
[2] M. Unser, A. Aldroubi and M. Eden,
    "B-Spline Signal Processing: Part I--Theory,"
    IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 821-832,
    February 1993.
[3] M. Unser, A. Aldroubi and M. Eden,
    "B-Spline Signal Processing: Part II--Efficient Design and Applications,"
     IEEE Transactions on Signal Processing, vol. 41, no. 2, pp. 834-848,
     February 1993.
```

And code obtained from bigwww.epfl.ch by Philippe Thevenaz

The B spline coefficients are calculated through the BSplineDecompositionImageFilter

Limitations: Spline order must be between 0 and 5. Spline order must be set before setting the image. Uses mirror boundary conditions. Requires the same order of Spline for each dimension. Spline is determined in all dimensions, cannot selectively pick dimension for calculating spline.

**See** BSplineDecompositionImageFilter

See itk::BSplineInterpolateImageFunction for additional documentation.

### Warp an Image Using a Deformation Field

### Synopsis

Use WarpImageFilter and a deformation field for resampling an image.

### Results

Fig. 220: Input image

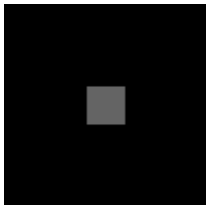


Fig. 221: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Wrap An Image Using A Deformation Field.
↪")
parser.add_argument("input_image")
parser.add_argument("displacement_field")
parser.add_argument("output_image")
args = parser.parse_args()

Dimension = 2

VectorComponentType = itk.F
VectorPixelType = itk.Vector[VectorComponentType, Dimension]

DisplacementFieldType = itk.Image[VectorPixelType, Dimension]

PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

fieldReader = itk.ImageFileReader[DisplacementFieldType].New()
fieldReader.SetFileName(args.displacement_field)
fieldReader.Update()
```

(continues on next page)

(continued from previous page)

```
deformationField = fieldReader.GetOutput()

warpFilter = itk.WarpImageFilter[ImageType, ImageType, DisplacementFieldType].New()

interpolator = itk.LinearInterpolateImageFunction[ImageType, itk.D].New()

warpFilter.SetInterpolator(interpolator)

warpFilter.SetOutputSpacing(deformationField.GetSpacing())
warpFilter.SetOutputOrigin(deformationField.GetOrigin())
warpFilter.SetOutputDirection(deformationField.GetDirection())

warpFilter.SetDisplacementField(deformationField)

warpFilter.SetInput(reader.GetOutput())

writer = itk.ImageFileWriter[ImageType].New()
writer.SetInput(warpFilter.GetOutput())
writer.SetFileName(args.output_image)

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkVector.h"
#include "itkWarpImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: \n" << argv[0] << "<InputFileName>
↪<DisplacementFieldFileName> <OutputFileName>" << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * displacementFieldFileName = argv[2];
  const char * outputFileName = argv[3];

  constexpr unsigned int Dimension = 2;

  using VectorComponentType = float;
  using VectorPixelType = itk::Vector<VectorComponentType, Dimension>;

  using DisplacementFieldType = itk::Image<VectorPixelType, Dimension>;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
```

(continues on next page)

(continued from previous page)

```
  using ReaderType = itk::ImageFileReader<ImageType>;
  using WriterType = itk::ImageFileWriter<ImageType>;

  using FieldReaderType = itk::ImageFileReader<DisplacementFieldType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  FieldReaderType::Pointer fieldReader = FieldReaderType::New();
  fieldReader->SetFileName(displacementFieldFileName);
  fieldReader->Update();

  DisplacementFieldType::ConstPointer deformationField = fieldReader->GetOutput();

  using FilterType = itk::WarpImageFilter<ImageType, ImageType, DisplacementFieldType>
↪;

  FilterType::Pointer filter = FilterType::New();

  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;

  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  filter->SetInterpolator(interpolator);

  filter->SetOutputSpacing(deformationField->GetSpacing());
  filter->SetOutputOrigin(deformationField->GetOrigin());
  filter->SetOutputDirection(deformationField->GetDirection());

  filter->SetDisplacementField(deformationField);

  filter->SetInput(reader->GetOutput());

  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(filter->GetOutput());
  writer->SetFileName(outputFileName);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TDisplacementField**>
**class WarpImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Warps an image using an input displacement field.

WarpImageFilter warps an existing image with respect to a given displacement field.

A displacement field is represented as a image whose pixel type is some vector type with at least N elements, where N is the dimension of the input image. The vector type must support element access via operator [].

The output image is produced by inverse mapping: the output pixels are mapped back onto the input image. This scheme avoids the creation of any holes and overlaps in the output image.

Each vector in the displacement field represent the distance between a geometric point in the input space and a point in the output space such that:

$$p_{in} = p_{out} + d$$

Typically the mapped position does not correspond to an integer pixel position in the input image. Interpolation via an image function is used to compute values at non-integer positions. The default interpolation typed used is the LinearInterpolateImageFunction. The user can specify a particular interpolation function via SetInterpolator(). Note that the input interpolator must derive from base class InterpolateImageFunction.

Position mapped to outside of the input image buffer are assigned a edge padding value.

The LargetPossibleRegion for the output is inherited from the input displacement field. The output image spacing, origin and orientation may be set via SetOutputSpacing, SetOutputOrigin and SetOutputDirection. The default are respectively a vector of 1's, a vector of 0's and an identity matrix.

This class is templated over the type of the input image, the type of the output image and the type of the displacement field.

The input image is set via SetInput. The input displacement field is set via SetDisplacementField.

This filter is implemented as a multithreaded filter.

**Warning** This filter assumes that the input type, output type and displacement field type all have the same number of dimensions.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Warp An Image Using A Deformation Field

See itk::WarpImageFilter for additional documentation.

## 3.4.17 ImageIntensity

### Absolute Value of Image

### Synopsis

Compute the absolute value of an image.

### Results

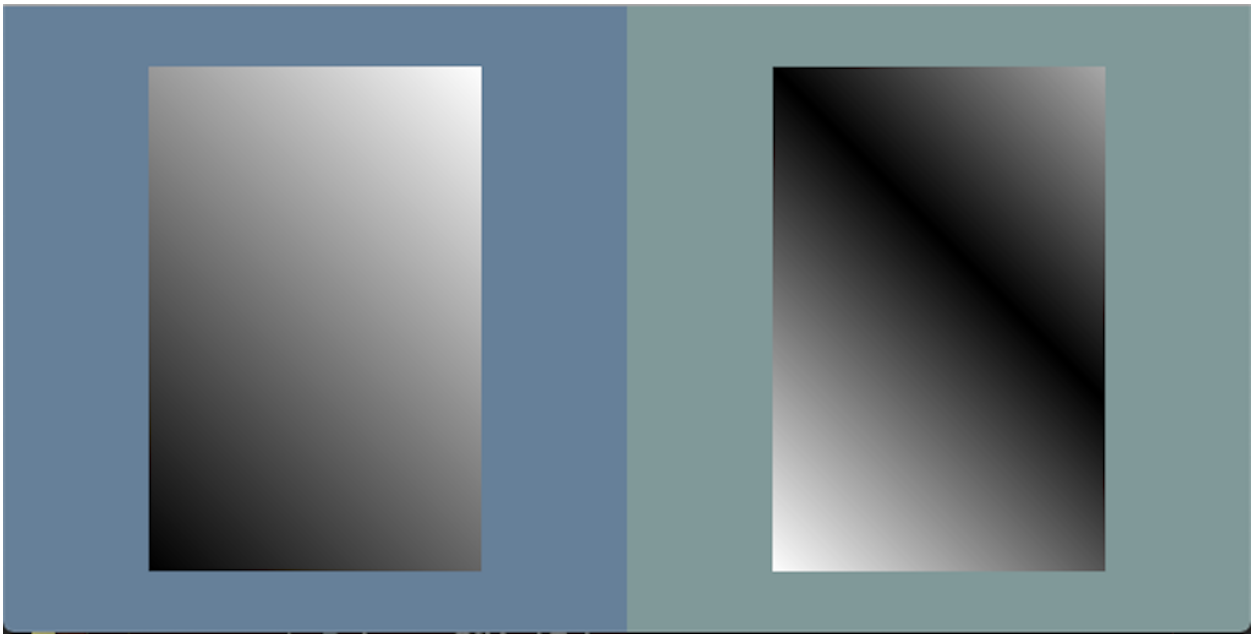


Fig. 222: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkAbsImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
```

(continues on next page)

(continued from previous page)

```
CreateImage(FloatImageType::Pointer image);

int
main(int, char *[])
{
  FloatImageType::Pointer image = FloatImageType::New();
  CreateImage(image);

  // Take the absolute value of the image
  using AbsImageFilterType = itk::AbsImageFilter<FloatImageType, FloatImageType>;

  AbsImageFilterType::Pointer absFilter = AbsImageFilterType::New();
  absFilter->SetInput(image);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage<FloatImageType>(image);
  viewer.AddImage<FloatImageType>(absFilter->GetOutput());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(FloatImageType::Pointer image)
{
  // Create an image with negative values
  FloatImageType::RegionType region;
  FloatImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  FloatImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<FloatImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(imageIterator.GetIndex()[0] - imageIterator.GetIndex()[1]);
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class AbsImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the absolute value of each pixel.

itk::Math::abs() is used to perform the computation.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Absolute Value Of Image

See itk::AbsImageFilter for additional documentation.

### Add Constant to Every Pixel

### Synopsis

Add a constant to every pixel in an image.

### Results

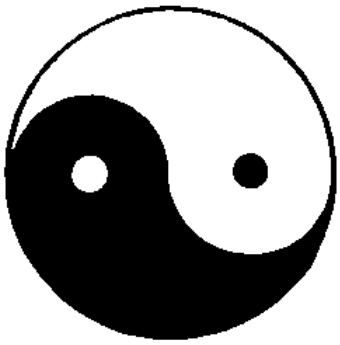


Fig. 223: Yinyang.png

Fig. 224: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkAddImageFilter.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using AddImageFilterType = itk::AddImageFilter<ImageType, ImageType, ImageType>;
  AddImageFilterType::Pointer addImageFilter = AddImageFilterType::New();
  addImageFilter->SetInput(image);
  addImageFilter->SetConstant2(2);
  addImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(addImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
```

(continues on next page)

(continued from previous page)

```
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

## Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class AddImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Pixel-wise addition of two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

The pixel type of the input 1 image must have a valid definition of the operator+ with a pixel type of the image 2. This condition is required because internally this filter will perform the operation

```
pixel_from_image_1 + pixel_from_image_2
```

Additionally the type resulting from the sum, will be cast to the pixel type of the output image.

The total operation over one pixel will be

```
output_pixel = static_cast<OutputPixelType>( input1_pixel + input2_pixel )
```

For example, this filter could be used directly for adding images whose pixels are vectors of the same dimension, and to store the resulting vector in an output image of vector pixels.

The images to be added are set using the methods:

```
SetInput1( image1 );
SetInput2( image2 );
```

Additionally, this filter can be used to add a constant to every pixel of an image by using

```
SetInput1( image1 );
SetConstant2( constant );
```

**Warning** No numeric overflow checking is performed in this filter.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Two Images Together
- Add Constant To Every Pixel

See itk::AddImageFilter for additional documentation.

### Add Two Images Together

### Synopsis

Add two images together.

### Results

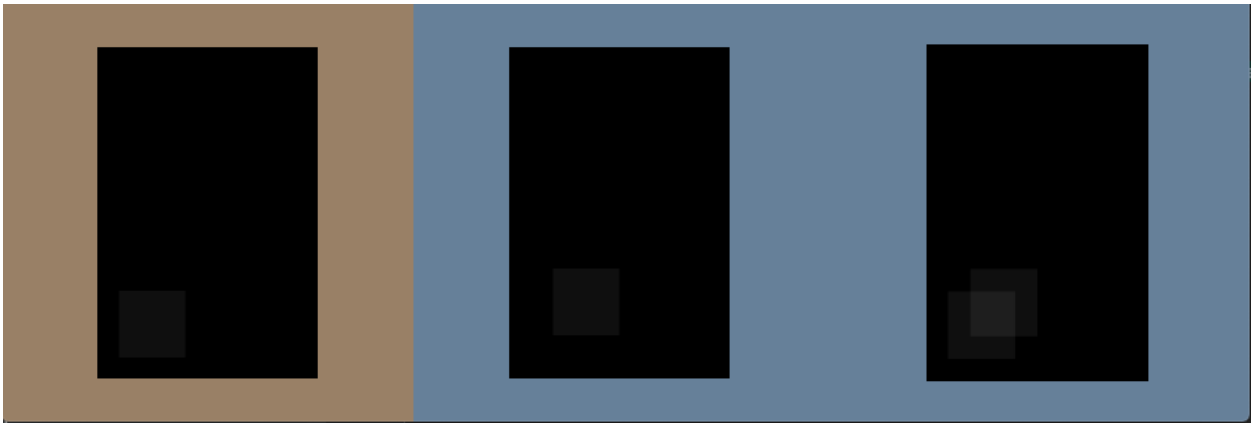


Fig. 225: Output In VTK Image

### Code

### C++

```
#include "itkImage.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkAddImageFilter.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage1(ImageType::Pointer image);
static void
CreateImage2(ImageType::Pointer image);

int
```

(continues on next page)

(continued from previous page)

```
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using AddImageFilterType = itk::AddImageFilter<ImageType, ImageType>;

  AddImageFilterType::Pointer addFilter = AddImageFilterType::New();
  addFilter->SetInput1(image1);
  addFilter->SetInput2(image2);
  addFilter->Update();

  // Visualize first image
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector1 = ConnectorType::New();
  connector1->SetInput(image1);

  vtkSmartPointer<vtkImageActor> actor1 = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor1->SetInput(connector1->GetOutput());
#else
  connector1->Update();
  actor1->GetMapper()->SetInputData(connector1->GetOutput());
#endif
  // Visualize first image
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector2 = ConnectorType::New();
  connector2->SetInput(image2);

  vtkSmartPointer<vtkImageActor> actor2 = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor2->SetInput(connector2->GetOutput());
#else
  connector2->Update();
  actor2->GetMapper()->SetInputData(connector2->GetOutput());
#endif

  // Visualize joined image
  ConnectorType::Pointer addConnector = ConnectorType::New();
  addConnector->SetInput(addFilter->GetOutput());

  vtkSmartPointer<vtkImageActor> addActor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  addActor->SetInput(addConnector->GetOutput());
#else
  addConnector->Update();
  addActor->GetMapper()->SetInputData(addConnector->GetOutput());
#endif
  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(900, 300);

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
```

(continues on next page)

(continued from previous page)

```
  interactor->SetRenderWindow(renderWindow);

  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.33, 1.0 };
  double centerViewport[4] = { 0.33, 0.0, 0.66, 1.0 };
  double rightViewport[4] = { 0.66, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);
  leftRenderer->SetBackground(.6, .5, .4);

  vtkSmartPointer<vtkRenderer> centerRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(centerRenderer);
  centerRenderer->SetViewport(centerViewport);
  centerRenderer->SetBackground(.4, .5, .6);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(.4, .5, .6);

  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(actor1);
  centerRenderer->AddActor(actor2);
  rightRenderer->AddActor(addActor);

  leftRenderer->ResetCamera();
  centerRenderer->ResetCamera();
  rightRenderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
```

(continues on next page)

(continued from previous page)

```
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 80; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}


void
CreateImage2(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make another square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class AddImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Pixel-wise addition of two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

The pixel type of the input 1 image must have a valid definition of the operator+ with a pixel type of the image 2. This condition is required because internally this filter will perform the operation

```
pixel_from_image_1 + pixel_from_image_2
```

Additionally the type resulting from the sum, will be cast to the pixel type of the output image.

The total operation over one pixel will be

```
output_pixel = static_cast<OutputPixelType>( input1_pixel + input2_pixel )
```

For example, this filter could be used directly for adding images whose pixels are vectors of the same dimension, and to store the resulting vector in an output image of vector pixels.

The images to be added are set using the methods:

```
SetInput1( image1 );
SetInput2( image2 );
```

Additionally, this filter can be used to add a constant to every pixel of an image by using

```
SetInput1( image1 );
SetConstant2( constant );
```

**Warning** No numeric overflow checking is performed in this filter.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Add Two Images Together
- Add Constant To Every Pixel

See itk::AddImageFilter for additional documentation.

### Apply Atan Image Filter

### Synopsis

Compute the arctangent of each pixel.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkImageFileWriter.h"
#include "itkAtanImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using InputPixelType = unsigned char;

  using InputImageType = itk::Image<InputPixelType, Dimension>;

  InputImageType::SizeType size;
  size.Fill(100);

  using RandomImageSourceType = itk::RandomImageSource<InputImageType>;

  RandomImageSourceType::Pointer randomImageSource = RandomImageSourceType::New();
  randomImageSource->SetNumberOfWorkUnits(1); // to produce non-random results
  randomImageSource->SetSize(size);

  using OutputPixelType = float;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::AtanImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(randomImageSource->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[1]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class AtanImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the one-argument inverse tangent of each pixel.

This filter is templated over the pixel type of the input image and the pixel type of the output image.

The filter walks over all the pixels in the input image, and for each pixel does the following:

- cast the pixel value to `double`,
- apply the `std::atan()` function to the `double` value,
- cast the `double` value resulting from `std::atan()` to the pixel type of the output image,
- store the cast value into the output image.

See itk::AtanImageFilter for additional documentation.

### Apply Cos Image Filter

### Synopsis

Compute the cosine of each pixel

### Results

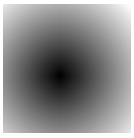


Fig. 226: Rescaled Input image [0, 255]

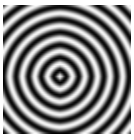


Fig. 227: Rescaled Output image [0, 255]

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCosImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << " <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::CosImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class CosImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the cosine of each pixel.

This filter is templated over the pixel type of the input image and the pixel type of the output image.

The filter walks over all of the pixels in the input image, and for each pixel does the following:

- cast the pixel value to `double`,
- apply the `std::cos()` function to the `double` value,
- cast the `double` value resulting from `std::cos()` to the pixel type of the output image,
- store the cast value into the output image.

The filter expects both images to have the same dimension (e.g. both 2D, or both 3D, or both ND)

See itk::CosImageFilter for additional documentation.

### Apply Exp Negative Image Filter

### Synopsis

Compute $exp(-Kx)$ of each pixel.

### Results

Fig. 228: Input image

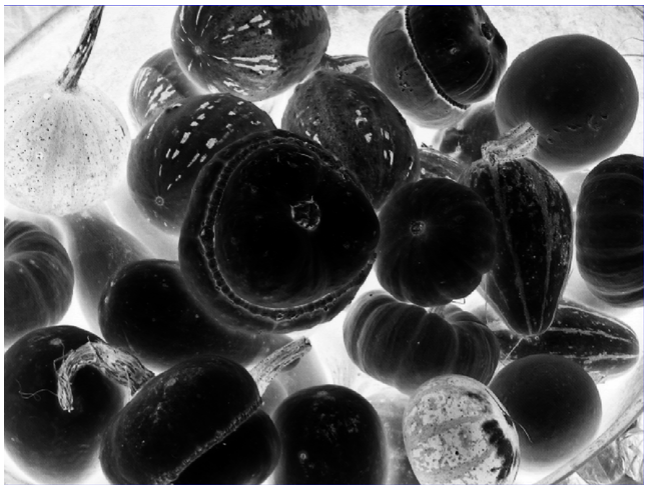


Fig. 229: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkExpNegativeImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <K: ExpNegative parameter>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const double k = std::stod(argv[3]);

  constexpr unsigned int Dimension = 2;
  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::ExpNegativeImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetFactor(k);
```

(continues on next page)

(continued from previous page)

```
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Apply Exp Negative Image Filter.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("k", help="ExpNegative parameter.", type=float)
args = parser.parse_args()

Dimension = 2
PixelType = itk.F
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

expFilter = itk.ExpNegativeImageFilter[ImageType, ImageType].New()
expFilter.SetInput(reader.GetOutput())
expFilter.SetFactor(args.k)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(expFilter.GetOutput())

writer.Update()
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ExpNegativeImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::ExpNegativ
Computes the function exp(-K.x) for each input pixel.

Every output pixel is equal to std::exp(-K.x ). where x is the intensity of the homologous input pixel, and K is a user-provided constant.

See itk::ExpNegativeImageFilter for additional documentation.

### Apply Sin Image Filter

### Synopsis

Compute the sine of each pixel

### Results

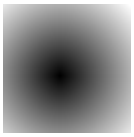


Fig. 230: Rescaled Input image [0, 255]

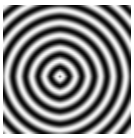


Fig. 231: Rescaled Output image [0, 255]

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSinImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::SinImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SinImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>

Computes the sine of each pixel.

The computations are performed using std::sin(x).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Sin Image Filter.

See itk::SinImageFilter for additional documentation.

### Binary AND Two Images

### Synopsis

Binary AND two images.

### Results

Fig. 232: input1.png

Fig. 233: input2.png

Fig. 234: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkSimpleFilterWatcher.h"
#include "itkAndImageFilter.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage1(ImageType::Pointer image);
static void
CreateImage2(ImageType::Pointer image);

int
main(int, char *[])
{
```

(continues on next page)

(continued from previous page)

```
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input1.png");
  writer->SetInput(image1);
  writer->Update();

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  writer->SetFileName("input2.png");
  writer->SetInput(image2);
  writer->Update();

  using AndImageFilterType = itk::AndImageFilter<ImageType>;

  AndImageFilterType::Pointer andFilter = AndImageFilterType::New();
  andFilter->SetInput(0, image1);
  andFilter->SetInput(1, image2);
  andFilter->Update();

  writer->SetFileName("output.png");
  writer->SetInput(andFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
```

(continues on next page)

(continued from previous page)

```
    }

    ++imageIterator;
  }
}

void
CreateImage2(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 30)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class AndImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Implements the AND bitwise operator pixel-wise between two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

Since the bitwise AND operation is only defined in C++ for integer types, the images passed to this filter must comply with the requirement of using integer pixel type.

The total operation over one pixel will be

```
output_pixel = static_cast<OutputPixelType>( input1_pixel & input2_pixel )
```

Where "&" is the bitwise AND operator in C++.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Binary AND Two Images

See itk::AndImageFilter for additional documentation.

### Binary OR Two Images

### Synopsis

Binary OR two images.

### Results

Fig. 235: input1.png

Fig. 236: input2.png

Fig. 237: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkSimpleFilterWatcher.h"
#include "itkOrImageFilter.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileWriter.h"
```

(continues on next page)

(continued from previous page)

```
using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage1(ImageType::Pointer image);
static void
CreateImage2(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input1.png");
  writer->SetInput(image1);
  writer->Update();

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  writer->SetFileName("input2.png");
  writer->SetInput(image2);
  writer->Update();

  using OrImageFilterType = itk::OrImageFilter<ImageType>;
  OrImageFilterType::Pointer orFilter = OrImageFilterType::New();
  orFilter->SetInput(0, image1);
  orFilter->SetInput(1, image2);
  orFilter->Update();

  writer->SetFileName("output.png");
  writer->SetInput(orFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
```

(continues on next page)

(continued from previous page)

```
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}

void
CreateImage2(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 30)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class OrImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Implements the OR bitwise operator pixel-wise between two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

Since the bitwise OR operation is only defined in C++ for integer types, the images passed to this filter must comply with the requirement of using integer pixel type.

The total operation over one pixel will be

```
output_pixel = static_cast<OutputPixelType>( input1_pixel | input2_pixel )
```

Where "|" is the boolean OR operator in C++.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Binary OR Two Images

See itk::OrImageFilter for additional documentation.

### Binary XOR Two Images

### Synopsis

Binary XOR (exclusive OR) two images.

### Results

Fig. 238: input1.png

Fig. 239: input2.png

Fig. 240: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkSimpleFilterWatcher.h"
#include "itkXorImageFilter.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage1(ImageType::Pointer image);
static void
CreateImage2(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using XorImageFilterType = itk::XorImageFilter<ImageType>;
  XorImageFilterType::Pointer xorFilter = XorImageFilterType::New();
  xorFilter->SetInput1(image1);
  xorFilter->SetInput2(image2);
  xorFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(xorFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);
```

(continues on next page)

(continued from previous page)

```
  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}

void
CreateImage2(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 30)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class XorImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Computes the XOR bitwise operator pixel-wise between two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

Since the bitwise XOR operation is only defined in C++ for integer types, the images passed to this filter must comply with the requirement of using integer pixel type.

The total operation over one pixel will be

```
output_pixel = static_cast<OutputPixelType>( input1_pixel ^ input2_pixel )
```

Where "^" is the boolean XOR operator in C++.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Binary XOR Two Images

See itk::XorImageFilter for additional documentation.

### Cast Image to Another Type but Clamp to Output Range

### Synopsis

Cast an image from one type to another but clamp to the output value range.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkClampImageFilter.h"

using FloatImageType = itk::Image<float, 2>;
using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(FloatImageType * const image);

int
main(int argc, char * argv[])
{
  FloatImageType::Pointer image;
```

(continues on next page)

(continued from previous page)

```
  // No input image argument provided
  if (argc < 2)
  {
    image = FloatImageType::New();
    CreateImage(image);
  }
  else // Input image argument provided
  {
    using ReaderType = itk::ImageFileReader<FloatImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
  }

  using ClampFilterType = itk::ClampImageFilter<FloatImageType, UnsignedCharImageType>
↪;
  ClampFilterType::Pointer clampFilter = ClampFilterType::New();
  clampFilter->SetInput(image);
  clampFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(FloatImageType * const image)
{
  // Create an image with 2 connected components
  FloatImageType::IndexType corner = { { 0, 0 } };

  FloatImageType::SizeType size;
  unsigned int             NumRows = 200;
  unsigned int             NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  FloatImageType::RegionType region(corner, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      FloatImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ClampImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::Clamp<*TInputImag*

Casts input pixels to output pixel type and clamps the output pixel values to a specified range.

Default range corresponds to the range supported by the pixel type of the output image.

This filter is templated over the input image type and the output image type.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** UnaryFunctorImageFilter

**See** CastImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cast Image To Another Type But Clamp To Output Range

See itk::ClampImageFilter for additional documentation.

### Compare Two Images and Set Output Pixel to Max

### Synopsis

Pixel wise compare two input images and set the output pixel to their max.

### Results

**Note:** No output is printed, this example simply displays functionality.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkMaximumImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage1(ImageType * image);
static void
CreateImage2(ImageType * image);
```

(continues on next page)

(continued from previous page)

```
int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using MaximumImageFilterType = itk::MaximumImageFilter<ImageType>;

  MaximumImageFilterType::Pointer maximumImageFilter = MaximumImageFilterType::New();
  maximumImageFilter->SetInput(0, image1);
  maximumImageFilter->SetInput(1, image2);
  maximumImageFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType * image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 30)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}

void
CreateImage2(ImageType * image)
{
  ImageType::IndexType start;
  start.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class MaximumImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Implements a pixel-wise operator Max(a,b) between two images.

The pixel values of the output image are the maximum between the corresponding pixels of the two input images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compare Two Images And Set Output Pixel To Max

See itk::MaximumImageFilter for additional documentation.

### Compare Two Images and Set Output Pixel to Min

### Synopsis

Pixel wise compare two input images and set the output pixel to their min.

### Results

**Note:** No output is printed, this example simply displays functionality.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkMinimumImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage1(ImageType * image);
static void
CreateImage2(ImageType * image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using MinimumImageFilterType = itk::MinimumImageFilter<ImageType>;

  MinimumImageFilterType::Pointer minimumImageFilter = MinimumImageFilterType::New();
  minimumImageFilter->SetInput(0, image1);
  minimumImageFilter->SetInput(1, image2);
  minimumImageFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType * image)
{
  ImageType::IndexType start;
  start.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 30)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}

void
CreateImage2(ImageType * image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }
```

(continues on next page)

(continued from previous page)

```
    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class MinimumImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Implements a pixel-wise operator Min(a,b) between two images.

The pixel values of the output image are the minimum between the corresponding pixels of the two input images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compare Two Images And Set Output Pixel To Min

See itk::MinimumImageFilter for additional documentation.

### Compute Edge Potential

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Compute edge potential.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkCovariantVector.h"
#include "itkEdgePotentialImageFilter.h"
#include "itkGradientImageFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int /*argc*/, char * /*argv*/[])
{
  // Setup types
  using FloatImageType = itk::Image<float, 2>;
  using VectorImageType = itk::Image<itk::CovariantVector<float, 2>, 2>;

  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  // Create and setup a gradient filter
  using GradientFilterType = itk::GradientImageFilter<UnsignedCharImageType, float>;
  GradientFilterType::Pointer gradientFilter = GradientFilterType::New();
  gradientFilter->SetInput(image);
  gradientFilter->Update();

  // Create and setup an edge potential filter
  using EdgePotentialImageFilterType = itk::EdgePotentialImageFilter<VectorImageType,␣
↪FloatImageType>;
  EdgePotentialImageFilterType::Pointer edgePotentialImageFilter =␣
↪EdgePotentialImageFilterType::New();
  edgePotentialImageFilter->SetInput(gradientFilter->GetOutput());
  edgePotentialImageFilter->Update();

  // Scale so we can write to a PNG
  using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(edgePotentialImageFilter->GetOutput());
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);
  rescaleFilter->Update();

  using FileWriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  FileWriterType::Pointer writer = FileWriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create a black image with 2 white regions

  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(200);

  UnsignedCharImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      UnsignedCharImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class EdgePotentialImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the edge potential of an image from the image gradient.

Input to this filter should be a CovariantVector image representing the image gradient.

The filter expect both the input and output images to have the same number of dimensions, and the output to be of a scalar image type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Edge Potential

See itk::EdgePotentialImageFilter for additional documentation.

### Compute Sigmoid

### Synopsis

Computes the sigmoid function pixel-wise.

### Results

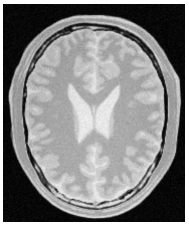


Fig. 241: Input image

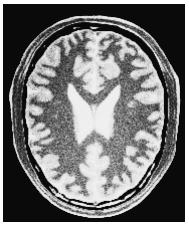


Fig. 242: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Sigmoid.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("output_min", type=int)
parser.add_argument("output_max", type=int)
parser.add_argument("alpha", type=float)
parser.add_argument("beta", type=float)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]
```

(continues on next page)

(continued from previous page)

```
reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

sigmoidFilter = itk.SigmoidImageFilter[ImageType, ImageType].New()
sigmoidFilter.SetInput(reader.GetOutput())
sigmoidFilter.SetOutputMinimum(args.output_min)
sigmoidFilter.SetOutputMaximum(args.output_max)
sigmoidFilter.SetAlpha(args.alpha)
sigmoidFilter.SetBeta(args.beta)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(sigmoidFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkSigmoidImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 7)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " <OutputMin> <OutputMax> <Alpha> <Beta>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
  using ScalarType = float;

  const auto       outputMinimum = static_cast<PixelType>(atoi(argv[3]));
  const auto       outputMaximum = static_cast<PixelType>(atoi(argv[4]));
  const ScalarType alpha = std::stod(argv[5]);
  const ScalarType beta = std::stod(argv[6]);

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::SigmoidImageFilter<ImageType, ImageType>;
```

(continues on next page)

(continued from previous page)

```
  FilterType::Pointer sigmoidFilter = FilterType::New();
  sigmoidFilter->SetInput(reader->GetOutput());
  sigmoidFilter->SetOutputMinimum(outputMinimum);
  sigmoidFilter->SetOutputMaximum(outputMaximum);
  sigmoidFilter->SetAlpha(alpha);
  sigmoidFilter->SetBeta(beta);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(sigmoidFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SigmoidImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::Sigmoid<*TInput*

Computes the sigmoid function pixel-wise.

A linear transformation is applied first on the argument of the sigmoid function. The resulting total transform is given by

$$f(x) = (Max - Min) \cdot \frac{1}{\left(1 + e^{-\frac{x-\beta}{\alpha}}\right)} + Min$$

Every output pixel is equal to f(x). Where x is the intensity of the homologous input pixel, and alpha and beta are user-provided constants.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Sigmoid

See itk::SigmoidImageFilter for additional documentation.

### Computer Magnitude in Vector Image to Make Magnitude Image

### Synopsis

Compute the magnitude of each pixel in a vector image to produce a magnitude image

### Results

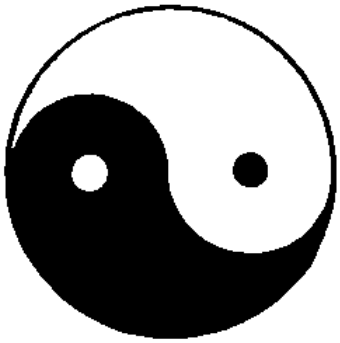


Fig. 243: Yinyang.png

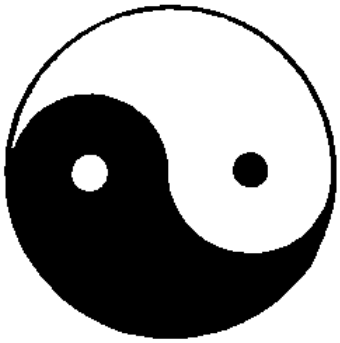


Fig. 244: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkVectorImage.h"
#include "itkVectorMagnitudeImageFilter.h"

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile outputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];
  std::string outputFilename = argv[2];

  // Setup types
  using VectorImageType = itk::VectorImage<float, 2>;
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;

  // Create and setup a reader
  using ReaderType = itk::ImageFileReader<VectorImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;

  using VectorMagnitudeFilterType = itk::VectorMagnitudeImageFilter<VectorImageType,␣
↪UnsignedCharImageType>;
  VectorMagnitudeFilterType::Pointer magnitudeFilter = VectorMagnitudeFilterType::
↪New();
  magnitudeFilter->SetInput(reader->GetOutput());

  // To write the magnitude image file, we should rescale the gradient values
  // to a reasonable range
  using rescaleFilterType = itk::RescaleIntensityImageFilter<UnsignedCharImageType,␣
↪UnsignedCharImageType>;

  rescaleFilterType::Pointer rescaler = rescaleFilterType::New();
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(255);
  rescaler->SetInput(magnitudeFilter->GetOutput());

  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName(outputFilename);
  writer->SetInput(rescaler->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class VectorMagnitudeImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Take an image of vectors as input and produce an image with the magnitude of those vectors.

The filter expects the input image pixel type to be a vector and the output image pixel type to be a scalar.

This filter assumes that the PixelType of the input image is a VectorType that provides a GetNorm() method.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Computer Magnitude In Vector Image To Make Magnitude Image

See itk::VectorMagnitudeImageFilter for additional documentation.

### Convert RGB Image to Grayscale Image

### Synopsis

Convert a RGB Image to its luminance image (grayscale one).

### Results

Fig. 245: Input image

Fig. 246: Output image

### Code

### Python

```python
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Compute RBG Image To Grayscale Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

Dimension = 2

ComponentType = itk.UC
InputPixelType = itk.RGBPixel[ComponentType]
InputImageType = itk.Image[InputPixelType, Dimension]

OutputPixelType = itk.UC
OutputImageType = itk.Image[OutputPixelType, Dimension]

reader = itk.ImageFileReader[InputImageType].New()
reader.SetFileName(args.input_image)

rgbFilter = itk.RGBToLuminanceImageFilter.New(reader)

writer = itk.ImageFileWriter[OutputImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(rgbFilter.GetOutput())

writer.Update()
```

### C++

```cpp
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRGBToLuminanceImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  using ComponentType = unsigned char;
  using InputPixelType = itk::RGBPixel<ComponentType>;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::RGBToLuminanceImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RGBToLuminanceImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Converts an RGB image into a grayscale image.

This filters converts an RGB image into a Luminance on by computing pixel-wise a linear combination on the Red, Green and Blue channels. The pixel type of the input image must have a GetLuminance() method. This is the case of the itk::RGBPixel class.

See itk::RGBToLuminanceImageFilter for additional documentation.

### Extract Component of Vector Image

### Synopsis

Extract a component/channel of a vector image.

### Results

### Code

### C++

```
#include "itkVectorImage.h"
#include "itkImageRegionIterator.h"
#include "itkMinimumMaximumImageCalculator.h"
#include "itkVectorIndexSelectionCastImageFilter.h"

using VectorImageType = itk::VectorImage<float, 2>;
using ScalarImageType = itk::Image<float, 2>;

static void
CreateImage(VectorImageType::Pointer image);

int
main(int, char *[])
{
  VectorImageType::Pointer image = VectorImageType::New();
  CreateImage(image);

  using IndexSelectionType = itk::VectorIndexSelectionCastImageFilter<VectorImageType,
↪ ScalarImageType>;
  IndexSelectionType::Pointer indexSelectionFilter = IndexSelectionType::New();
  indexSelectionFilter->SetIndex(0);
  indexSelectionFilter->SetInput(image);

  using ImageCalculatorFilterType = itk::MinimumMaximumImageCalculator
↪<ScalarImageType>;
  ImageCalculatorFilterType::Pointer imageCalculatorFilter =␣
↪ImageCalculatorFilterType::New();
  imageCalculatorFilter->SetImage(indexSelectionFilter->GetOutput());
  imageCalculatorFilter->Compute();

  return EXIT_SUCCESS;
}

void
CreateImage(VectorImageType::Pointer image)
{
  VectorImageType::IndexType start;
  start.Fill(0);

  VectorImageType::SizeType size;
  size.Fill(2);
```

(continues on next page)

(continued from previous page)

```
  VectorImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->SetNumberOfComponentsPerPixel(3);
  image->Allocate();

  using VariableVectorType = itk::VariableLengthVector<double>;
  VariableVectorType variableLengthVector;
  variableLengthVector.SetSize(3);
  variableLengthVector[0] = 1.1;
  variableLengthVector[1] = 2.2;
  variableLengthVector[2] = 3.3;

  image->FillBuffer(variableLengthVector);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class VectorIndexSelectionCastImageFilter** : **public** itk::*UnaryFunctorImageFilter<TInputImage*, *TOutputImage*, F

Extracts the selected index of the vector that is the input pixel type.

This filter is templated over the input image type and output image type.

The filter expect the input image pixel type to be a vector and the output image pixel type to be a scalar. The only requirement on the type used for representing the vector is that it must provide an operator[].

**See** ComposeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Component Of Vector Image

See itk::VectorIndexSelectionCastImageFilter for additional documentation.

### Intensity Windowing

### Synopsis

Apply a linear intensity transform from a specified input range to a specified output range.

### Results

Fig. 247: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkIntensityWindowingImageFilter.h"
#include "itkImageRegionIterator.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using IntensityWindowingImageFilterType = itk::IntensityWindowingImageFilter
↪<ImageType, ImageType>;

  IntensityWindowingImageFilterType::Pointer filter =␣
↪IntensityWindowingImageFilterType::New();
  filter->SetInput(image);
  filter->SetWindowMinimum(0);
  filter->SetWindowMaximum(100);
  filter->SetOutputMinimum(0);
  filter->SetOutputMaximum(255);
  filter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(image);
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
```

(continues on next page)

(continued from previous page)

```
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(10);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 30)
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class IntensityWindowingImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::I

Applies a linear transformation to the intensity levels of the input Image that are inside a user-defined interval. Values below this interval are mapped to a constant. Values over the interval are mapped to another constant.

IntensityWindowingImageFilter applies pixel-wise a linear transformation to the intensity values of input image pixels. The linear transformation is defined by the user in terms of the minimum and maximum values that the output image should have and the lower and upper limits of the intensity window of the input image. This operation is very common in visualization, and can also be applied as a convenient preprocessing operation for image segmentation.

All computations are performed in the precision of the input pixel's RealType. Before assigning the computed value to the output pixel.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Intensity Windowing

**See** RescaleIntensityImageFilter

See itk::IntensityWindowingImageFilter for additional documentation.

### Inverse of Mask to Image

### Synopsis

Apply the inverse of a mask to an image.

### Results

Fig. 248: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkMaskNegatedImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateHalfMask(ImageType::Pointer image, ImageType::Pointer mask);
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  ImageType::Pointer mask = ImageType::New();
  CreateHalfMask(image, mask);

  using MaskNegatedImageFilterType = itk::MaskNegatedImageFilter<ImageType, ImageType>
↪;
  MaskNegatedImageFilterType::Pointer maskNegatedImageFilter =␣
↪MaskNegatedImageFilterType::New();
  maskNegatedImageFilter->SetInput(image);
  maskNegatedImageFilter->SetMaskImage(mask);
  maskNegatedImageFilter->Update();
  ;

  using FileWriterType = itk::ImageFileWriter<ImageType>;
  FileWriterType::Pointer writer = FileWriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(maskNegatedImageFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  writer->Update();

  return EXIT_SUCCESS;
}


void
CreateHalfMask(ImageType::Pointer image, ImageType::Pointer mask)
{
  ImageType::RegionType region = image->GetLargestPossibleRegion();

  mask->SetRegions(region);
  mask->Allocate();

  ImageType::SizeType regionSize = region.GetSize();

  itk::ImageRegionIterator<ImageType> imageIterator(mask, region);

  // Make the left half of the mask white and the right half black
  while (!imageIterator.IsAtEnd())
  {
    if (static_cast<unsigned int>(imageIterator.GetIndex()[0]) > regionSize[0] / 2)
    {
      imageIterator.Set(0);
    }
    else
    {
      imageIterator.Set(1);
    }

    ++imageIterator;
  }
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(122);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage** = *TInputImage*>
**class MaskNegatedImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage*, *TMaskImage*, *TOutputImage*>
Mask an image with the negation (or logical compliment) of a mask.

This class is templated over the types of the input image type, the mask image type and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

The pixel type of the input 2 image must have a valid definition of the operator!=. This condition is required because internally this filter will perform the operation

```
if pixel_from_mask_image != mask_value
     pixel_output_image = output_value
else
     pixel_output_image = pixel_input_image
```

The pixel from the input 1 is cast to the pixel type of the output image.

Note that the input and the mask images must be of the same size.

**Warning** Only pixel value with mask_value ( defaults to 0 ) will be preserved.

**See** MaskImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Inverse Of Mask To Image

See itk::MaskNegatedImageFilter for additional documentation.

### Invert Image

### Synopsis

Invert an image.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkInvertIntensityImageFilter.h"
#include "itkImageFileReader.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using ImageType = itk::Image<unsigned char, 2>;
```

(continues on next page)

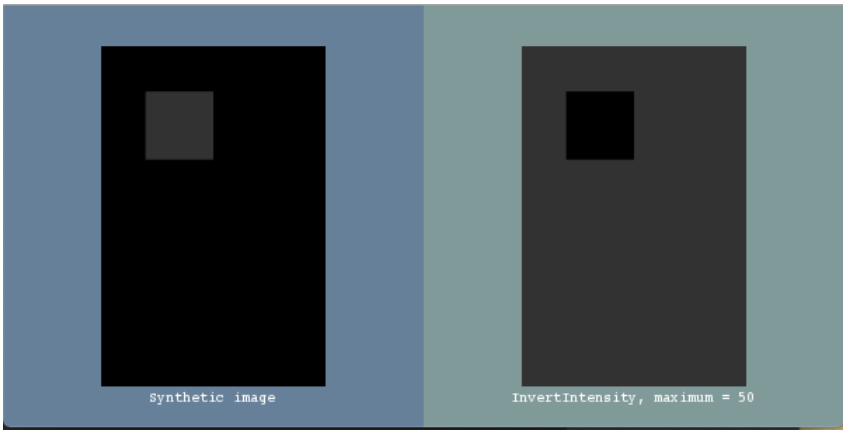



Fig. 249: Output In VTK Window

(continued from previous page)

```
static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  std::stringstream  desc;

  CreateImage(image);

  unsigned int maximum = 255;
  if (argc > 1)
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();
    image = reader->GetOutput();
    desc << itksys::SystemTools::GetFilenameName(argv[1]);
    if (argc > 2)
    {
      maximum = std::stoi(argv[2]);
    }
  }
  else
  {
    CreateImage(image);
    desc << "Synthetic image";
    maximum = 50;
  }

  using InvertIntensityImageFilterType = itk::InvertIntensityImageFilter<ImageType>;

  InvertIntensityImageFilterType::Pointer invertIntensityFilter =␣
↪InvertIntensityImageFilterType::New();
  invertIntensityFilter->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  invertIntensityFilter->SetMaximum(maximum);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image.GetPointer(), true, desc.str());

  std::stringstream desc2;
  desc2 << "InvertIntensity, maximum = " << maximum;
  viewer.AddImage(invertIntensityFilter->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 50);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class InvertIntensityImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Functor::Inver
Invert the intensity of an image.

InvertIntensityImageFilter inverts intensity of pixels by subtracting pixel value to a maximum value. The maximum value can be set with SetMaximum and defaults the maximum of input pixel type. This filter can be used to invert, for example, a binary image, a distance map, etc.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** IntensityWindowingImageFilter ShiftScaleImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Invert Image

See itk::InvertIntensityImageFilter for additional documentation.

### Mask Image

### Synopsis

Apply a mask to an image.

### Results

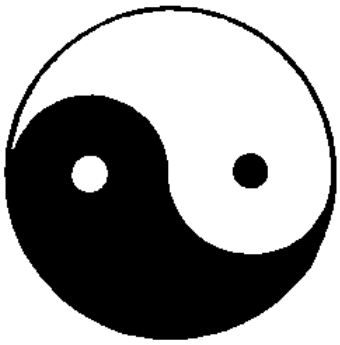


Fig. 250: Input Image

Output:

```
Image (0x7f9d9c95ccc0)
RTTI typeinfo:   itk::Image<unsigned char, 2u>
Reference Count: 2
```

(continues on next page)

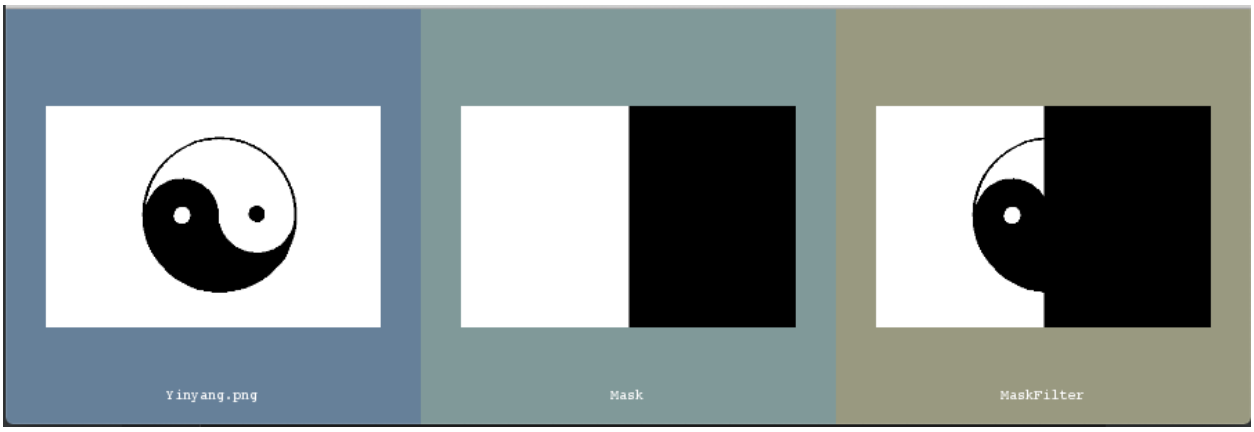



Fig. 251: Output In VTK Window

(continued from previous page)

```
Modified Time: 235
Debug: Off
Object Name:
Observers:
  none
Source: (none)
Source output name: (none)
Release Data: Off
Data Released: False
Global Release Data: Off
PipelineMTime: 0
UpdateMTime: 0
RealTimeStamp: 0 seconds
LargestPossibleRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [512, 342]
BufferedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [512, 342]
RequestedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [512, 342]
Spacing: [1, 1]
Origin: [0, 0]
Direction:
1 0
0 1

IndexToPointMatrix:
1 0
0 1

PointToIndexMatrix:
1 0
0 1
```

(continues on next page)

(continued from previous page)

```
Inverse Direction:
1 0
0 1

PixelContainer:
  ImportImageContainer (0x7f9d9c95d000)
    RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned char>
    Reference Count: 1
    Modified Time: 236
    Debug: Off
    Object Name:
    Observers:
      none
    Pointer: 0x7f9d90050000
    Container manages memory: true
    Size: 175104
    Capacity: 175104
```

### Code

### C++

```
#include "itkConfigure.h"

#if (ITK_VERSION_MAJOR < 4) // These are all defaults in ITKv4
//  Not supported in ITKv3.
int
main(int argc, char * argv[])
{
  return 0;
}
#else
#  include "itkImage.h"
#  include "itkImageFileReader.h"
#  include "itkMaskImageFilter.h"
#  include "itkImageRegionIterator.h"
#  ifdef ENABLE_QUICKVIEW
#    include "QuickView.h"
#  endif

using ImageType = itk::Image<unsigned char, 2>;

void
CreateHalfMask(ImageType::Pointer image, ImageType::Pointer & mask);

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0] << " filename" << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer mask = ImageType::New();
  CreateHalfMask(reader->GetOutput(), mask);

  using MaskFilterType = itk::MaskImageFilter<ImageType, ImageType>;
  MaskFilterType::Pointer maskFilter = MaskFilterType::New();
  maskFilter->SetInput(reader->GetOutput());
  maskFilter->SetMaskImage(mask);
  mask->Print(std::cout);
#  ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  std::stringstream desc;
  desc << "Mask";
  viewer.AddImage(mask.GetPointer(), true, desc.str());

  std::stringstream desc2;
  desc2 << "MaskFilter";
  viewer.AddImage(maskFilter->GetOutput(), true, desc2.str());

  viewer.Visualize();
#  endif
  return EXIT_SUCCESS;
}


void
CreateHalfMask(ImageType::Pointer image, ImageType::Pointer & mask)
{
  ImageType::RegionType region = image->GetLargestPossibleRegion();

  mask->SetRegions(region);
  mask->Allocate();

  ImageType::SizeType regionSize = region.GetSize();

  itk::ImageRegionIterator<ImageType> imageIterator(mask, region);

  // Make the left half of the mask white and the right half black
  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > static_cast<ImageType::IndexValueType>
↪(regionSize[0]) / 2)
    {
      imageIterator.Set(0);
    }
    else
    {
      imageIterator.Set(255);
    }
```

(continues on next page)

(continued from previous page)

```
    ++imageIterator;
  }
}
#endif
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMaskImage**, typename **TOutputImage** = *TInputImage*>
**class MaskImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage*, *TMaskImage*, *TOutputImage*>
Mask an image with a mask.

This class is templated over the types of the input image type, the mask image type and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

The pixel type of the input 2 image must have a valid definition of the operator != with zero. This condition is required because internally this filter will perform the operation

```
if pixel_from_mask_image != masking_value
     pixel_output_image = pixel_input_image
else
     pixel_output_image = outside_value
```

The pixel from the input 1 is cast to the pixel type of the output image.

Note that the input and the mask images must be of the same size.

**Warning** Any pixel value other than masking value (0 by default) will not be masked out.

**See** MaskNegatedImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mask Image

See itk::MaskImageFilter for additional documentation.

### Multiply Image by Scalar

### Synopsis

Multiply one image by a given scalar value.

### Results

Fig. 252: Input image

Fig. 253: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMultiplyImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <Factor> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char *           inputFileName = argv[1];
  const double           factor = std::stod(argv[2]);
  const char *           outputFileName = argv[3];
  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using OutputPixelType = double;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::MultiplyImageFilter<InputImageType, InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetConstant(factor);

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class MultiplyImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>
Pixel-wise multiplication of two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiply Two Images
- Multiply Image By Scalar

See itk::MultiplyImageFilter for additional documentation.

## Multiply Two Images

### Synopsis

Multiply two images

### Results

Fig. 254: Input image 1

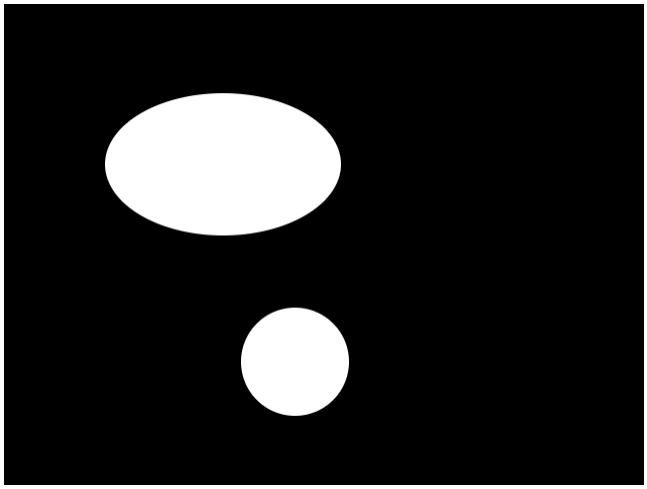


Fig. 255: Input image 2

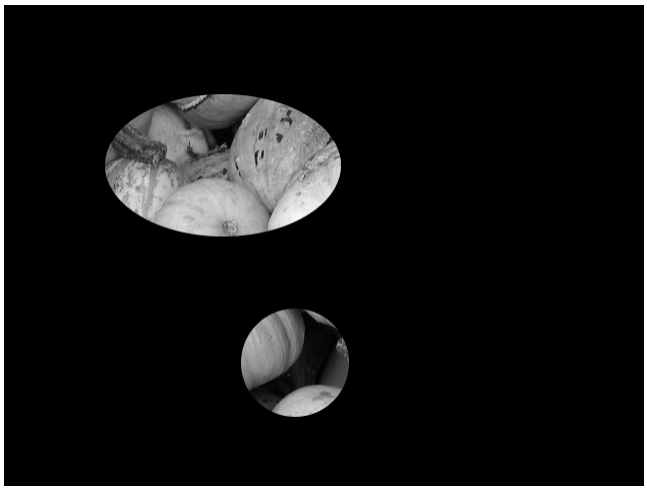


Fig. 256: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMultiplyImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName 1> <InputFileName 2> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName1 = argv[1];
  const char * inputFileName2 = argv[2];
  const char * outputFileName = argv[3];

  constexpr unsigned int Dimension = 2;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName(inputFileName1);

  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName(inputFileName2);

  using OutputPixelType = unsigned int;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::MultiplyImageFilter<InputImageType, InputImageType,␣
↪OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput1(reader1->GetOutput());
  filter->SetInput2(reader2->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class MultiplyImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>
Pixel-wise multiplication of two images.

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiply Two Images
- Multiply Image By Scalar

See itk::MultiplyImageFilter for additional documentation.

### Normalize Image

### Synopsis

Normalize an image.

### Results

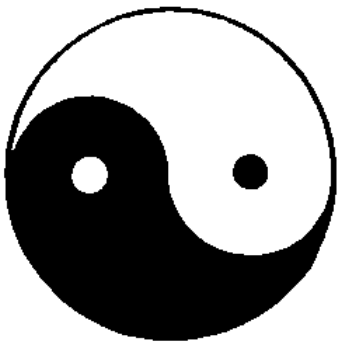


Fig. 257: Input Image

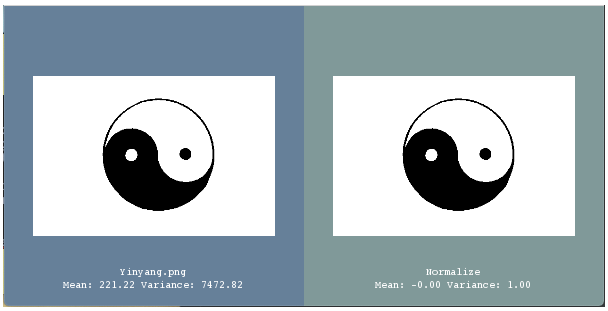



Fig. 258: Input And Output Image With Data

## Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkNormalizeImageFilter.h"
#include "itkStatisticsImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

#include <iomanip>

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0] << " filename" << std::endl;
    return EXIT_FAILURE;
  }

  using FloatImageType = itk::Image<double, 2>;

  using ReaderType = itk::ImageFileReader<FloatImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using NormalizeFilterType = itk::NormalizeImageFilter<FloatImageType,␣
↪FloatImageType>;
  NormalizeFilterType::Pointer normalizeFilter = NormalizeFilterType::New();
  normalizeFilter->SetInput(reader->GetOutput());

  using StatisticsFilterType = itk::StatisticsImageFilter<FloatImageType>;
  StatisticsFilterType::Pointer statistics1 = StatisticsFilterType::New();
  statistics1->SetInput(reader->GetOutput());

  StatisticsFilterType::Pointer statistics2 = StatisticsFilterType::New();
  statistics2->SetInput(normalizeFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
#ifdef ENABLE_QUICKVIEW
  QuickView viewer;

  std::stringstream desc1;
  statistics1->Update();
  desc1 << itksys::SystemTools::GetFilenameName(argv[1]) << "\nMean: " << statistics1-
↪>GetMean()
        << " Variance: " << statistics1->GetVariance();
  viewer.AddImage(reader->GetOutput(), true, desc1.str());

  std::stringstream desc2;
  statistics2->Update();
  desc2 << "Normalize"
        << "\nMean: " << std::fixed << std::setprecision(2) << statistics2->GetMean()
        << " Variance: " << statistics2->GetVariance();
  viewer.AddImage(normalizeFilter->GetOutput(), true, desc2.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class NormalizeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Normalize an image by setting its mean to zero and variance to one.

NormalizeImageFilter shifts and scales an image so that the pixels in the image have a zero mean and unit variance. This filter uses StatisticsImageFilter to compute the mean and variance of the input and then applies ShiftScaleImageFilter to shift and scale the pixels.

NB: since this filter normalizes the data such that the mean is at 0, and $-\sigma$ to $+\sigma$ is mapped to -1.0 to 1.0, output image integral types will produce an image that DOES NOT HAVE a unit variance due to 68% of the intensity values being mapped to the real number range of -1.0 to 1.0 and then cast to the output integral value.

**See** NormalizeToConstantImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Normalize Image

See itk::NormalizeImageFilter for additional documentation.

### Pixel Division of Two Images

### Synopsis

Pixel-wise division of two images.

### Results

Fig. 259: PixelDivisionOfTwoImages.png

### Code

### C++

```
#include "itkDivideImageFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;

void
CreateImage1(ImageType::Pointer image);
void
CreateImage2(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using DivideImageFilterType = itk::DivideImageFilter<ImageType, ImageType,␣
↪ImageType>;
```

(continues on next page)

(continued from previous page)

```
  DivideImageFilterType::Pointer divideImageFilter = DivideImageFilterType::New();
  divideImageFilter->SetInput1(image1);
  divideImageFilter->SetInput2(image2);
  divideImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("test.png");
  writer->SetInput(divideImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 80; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}


void
CreateImage2(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
```

(continues on next page)

(continued from previous page)

```
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make another square
  for (unsigned int r = 40; r < 100; r++)
  {
    for (unsigned int c = 40; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2**, typename **TOutputImage**>
**class DivideImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>
Pixel-wise division of two images.

This class is templated over the types of the two input images and the type of the output image. When the divisor is zero, the division result is set to the maximum number that can be represented by default to avoid exception. Numeric conversions (castings) are done by the C++ defaults.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Pixel Division Of Two Images

See itk::DivideImageFilter for additional documentation.

### Rescale an Image

### Synopsis

Rescale a grayscale image

### Results

Fig. 260: Input image

Fig. 261: Output image

### Code

### Python

```python
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Rescale Intensity.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("output_min", type=int)
parser.add_argument("output_max", type=int)
args = parser.parse_args()

image = itk.imread(args.input_image)

image = itk.rescale_intensity_image_filter(
    image, output_minimum=args.output_min, output_maximum=args.output_max
)

itk.imwrite(image, args.output_image)
```

### C++

```cpp
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName> <OutputMin> <OutputMax>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using FilterType = itk::RescaleIntensityImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
```

(continues on next page)

(continued from previous page)

```
  filter->SetInput(reader->GetOutput());
  filter->SetOutputMinimum(std::stoi(argv[3]));
  filter->SetOutputMaximum(std::stoi(argv[4]));

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class RescaleIntensityImageFilter** : **public** itk::*UnaryFunctorImageFilter<TInputImage*, *TOutputImage*, Functor::Inte

Applies a linear transformation to the intensity levels of the input Image.

RescaleIntensityImageFilter applies pixel-wise a linear transformation to the intensity values of input image pixels. The linear transformation is defined by the user in terms of the minimum and maximum values that the output image should have.

The following equation gives the mapping of the intensity values

All computations are performed in the precision of the input pixel's RealType. Before assigning the computed value to the output pixel.

$$outputPixel = (inputPixel - inputMin) \cdot \frac{(outputMax - outputMin)}{(inputMax - inputMin)} + outputMin$$

NOTE: In this filter the minimum and maximum values of the input image are computed internally using the MinimumMaximumImageCalculator. Users are not supposed to set those values in this filter. If you need a filter where you can set the minimum and maximum values of the input, please use the IntensityWindowingImage-Filter. If you want a filter that can use a user-defined linear transformation for the intensity, then please use the ShiftScaleImageFilter.

**See** IntensityWindowingImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Rescale An Image

See itk::RescaleIntensityImageFilter for additional documentation.

### Scale All Pixel's Sum to Constant

### Synopsis

Scale all pixels so that their sum is a specified constant.

### Results

Output:

```
0.111111
0.111111
0.111111
0.111111
0.111111
0.111111
0.111111
0.111111
0.111111
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageRegionConstIterator.h"
#include "itkNormalizeToConstantImageFilter.h"

using ImageType = itk::Image<float, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  // Create an image
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using NormalizeToConstantImageFilterType = itk::NormalizeToConstantImageFilter
↪<ImageType, ImageType>;
  NormalizeToConstantImageFilterType::Pointer normalizeToConstantImageFilter =
    NormalizeToConstantImageFilterType::New();
  normalizeToConstantImageFilter->SetInput(image);
  normalizeToConstantImageFilter->SetConstant(1);
  normalizeToConstantImageFilter->Update();

  itk::ImageRegionConstIterator<ImageType> imageIterator(
    normalizeToConstantImageFilter->GetOutput(),
    normalizeToConstantImageFilter->GetOutput()->GetLargestPossibleRegion());

  // The output pixels should all be 1/9 (=0.11111)
```

(continues on next page)

(continued from previous page)

```
  while (!imageIterator.IsAtEnd())
  {
    std::cout << imageIterator.Get() << std::endl;
    ++imageIterator;
  }

  return EXIT_SUCCESS;
}

static void
CreateImage(ImageType::Pointer image)
{
  // Create an image full of 1's

  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(3);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(1);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class NormalizeToConstantImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Scales image pixel intensities to make the sum of all pixels equal a user-defined constant.

The default value of the constant is 1. It can be changed with SetConstant().

This transform is especially useful for normalizing a convolution kernel.

This code was contributed in the Insight Journal paper: "FFT based

convolution" by Lehmann G. http://insight-journal.org/browse/publication/717

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** NormalizeImageFilter

**See** StatisticsImageFilter

**See** DivideImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Scale All Pixel's Sum To Constant

See itk::NormalizeToConstantImageFilter for additional documentation.

### Square Every Pixel

### Synopsis

Square every pixel in an image.

### Results

Fig. 262: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkSquareImageFilter.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using SquareImageFilterType = itk::SquareImageFilter<ImageType, ImageType>;
  SquareImageFilterType::Pointer squareImageFilter = SquareImageFilterType::New();
  squareImageFilter->SetInput(image);
  squareImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(squareImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class SquareImageFilter** : **public** itk::UnaryGeneratorImageFilter<*TInputImage*, *TOutputImage*>
Computes the square of the intensity values pixel-wise.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Square Every Pixel

See itk::SquareImageFilter for additional documentation.

### Subtract Constant From Every Pixel

### Synopsis

Subtract a constant from every pixel in an image.

### Results



Fig. 263: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkSubtractImageFilter.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using SubtractImageFilterType = itk::SubtractImageFilter<ImageType, ImageType,␣
↪ImageType>;
  SubtractImageFilterType::Pointer subtractConstantFromImageFilter =␣
↪SubtractImageFilterType::New();
  subtractConstantFromImageFilter->SetInput(image);
  subtractConstantFromImageFilter->SetConstant2(2);
  subtractConstantFromImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("output.png");
  writer->SetInput(subtractConstantFromImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);
```

(continues on next page)

(continued from previous page)

```
  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] < 70)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class SubtractImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Pixel-wise subtraction of two images.

Subtract each pixel from image2 from its corresponding pixel in image1:

```
Output = Input1 - Input2.
```

This is done using

```
SetInput1( image1 );
SetInput2( image2 );
```

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

Additionally, a constant can be subtracted from every pixel in an image using:

```
SetInput1( image1 );
SetConstant2( constant );
```

**Note** The result of AddImageFilter with a negative constant is not necessarily the same as SubtractImageFilter. This would be the case when the PixelType defines an operator-() that is not the inverse of operator+()

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Subtract Two Images

- Subtract Constant From Every Pixel

See itk::SubtractImageFilter for additional documentation.

### Subtract Two Images

### Synopsis

Subtract two images from one another.

### Results

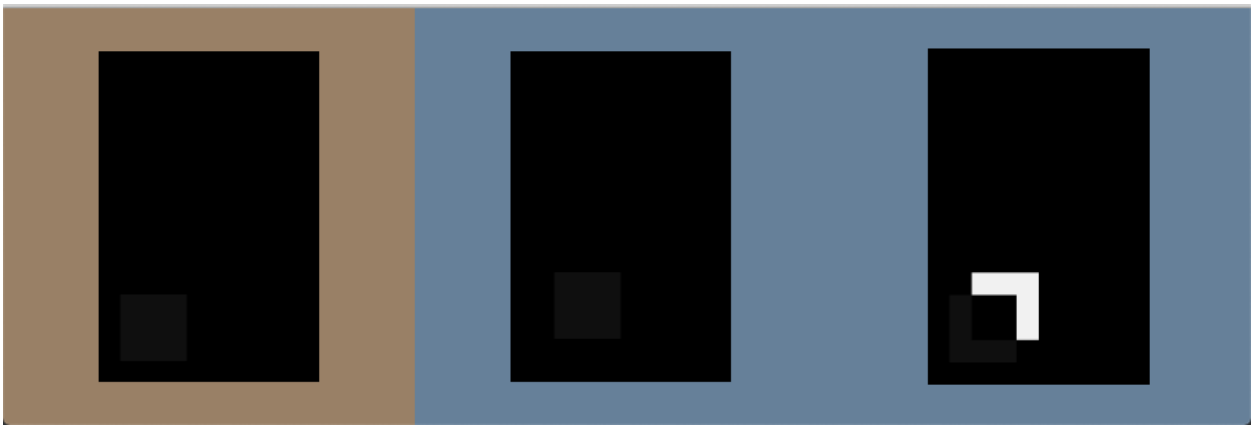


Fig. 264: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkSubtractImageFilter.h"

#include "itkImageToVTKImageFilter.h"

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkImageMapper3D.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using ImageType = itk::Image<unsigned char, 2>;
```

(continues on next page)

(continued from previous page)

```
static void
CreateImage1(ImageType::Pointer image);
static void
CreateImage2(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image1 = ImageType::New();
  CreateImage1(image1);

  ImageType::Pointer image2 = ImageType::New();
  CreateImage2(image2);

  using SubtractImageFilterType = itk::SubtractImageFilter<ImageType, ImageType>;

  SubtractImageFilterType::Pointer subtractFilter = SubtractImageFilterType::New();
  subtractFilter->SetInput1(image1);
  subtractFilter->SetInput2(image2);
  subtractFilter->Update();

  // Visualize first image
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector1 = ConnectorType::New();
  connector1->SetInput(image1);

  vtkSmartPointer<vtkImageActor> actor1 = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor1->SetInput(connector1->GetOutput());
#else
  connector1->Update();
  actor1->GetMapper()->SetInputData(connector1->GetOutput());
#endif

  // Visualize second image
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector2 = ConnectorType::New();
  connector2->SetInput(image2);

  vtkSmartPointer<vtkImageActor> actor2 = vtkSmartPointer<vtkImageActor>::New();

#if VTK_MAJOR_VERSION <= 5
  actor2->SetInput(connector2->GetOutput());
#else
  connector2->Update();
  actor2->GetMapper()->SetInputData(connector2->GetOutput());
#endif

  // Visualize subtracted image
  ConnectorType::Pointer subtractConnector = ConnectorType::New();
  subtractConnector->SetInput(subtractFilter->GetOutput());

  vtkSmartPointer<vtkImageActor> subtractActor = vtkSmartPointer<vtkImageActor>::
↪New();
#if VTK_MAJOR_VERSION <= 5
  subtractActor->SetInput(subtractConnector->GetOutput());
#else
```

(continues on next page)

(continued from previous page)

```
  subtractConnector->Update();
  subtractActor->GetMapper()->SetInputData(subtractConnector->GetOutput());
#endif
  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(900, 300);

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.33, 1.0 };
  double centerViewport[4] = { 0.33, 0.0, 0.66, 1.0 };
  double rightViewport[4] = { 0.66, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);
  leftRenderer->SetBackground(.6, .5, .4);

  vtkSmartPointer<vtkRenderer> centerRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(centerRenderer);
  centerRenderer->SetViewport(centerViewport);
  centerRenderer->SetBackground(.4, .5, .6);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(.4, .5, .6);

  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(actor1);
  centerRenderer->AddActor(actor2);
  rightRenderer->AddActor(subtractActor);

  leftRenderer->ResetCamera();
  centerRenderer->ResetCamera();
  rightRenderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}

void
CreateImage1(ImageType::Pointer image)
{
```

(continues on next page)

(continued from previous page)

```
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 20; c < 80; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}


void
CreateImage2(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make another square
  for (unsigned int r = 40; r < 100; r++)
  {
```

(continues on next page)

(continued from previous page)

```
    for (unsigned int c = 40; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 15);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage1**, typename **TInputImage2** = *TInputImage1*, typename **TOutputImage** = *TInputImage1*>
**class SubtractImageFilter** : **public** itk::BinaryGeneratorImageFilter<*TInputImage1*, *TInputImage2*, *TOutputImage*>

Pixel-wise subtraction of two images.

Subtract each pixel from image2 from its corresponding pixel in image1:

```
Output = Input1 - Input2.
```

This is done using

```
SetInput1( image1 );
SetInput2( image2 );
```

This class is templated over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

Additionally, a constant can be subtracted from every pixel in an image using:

```
SetInput1( image1 );
SetConstant2( constant );
```

**Note** The result of AddImageFilter with a negative constant is not necessarily the same as SubtractImageFilter. This would be the case when the PixelType defines an operator-() that is not the inverse of operator+()

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Subtract Two Images
- Subtract Constant From Every Pixel

See itk::SubtractImageFilter for additional documentation.

### Transform Magnitude of Vector Valued Image Pixels

### Synopsis

Apply a transformation to the magnitude of vector valued image pixels.

### Results

Fig. 265: Input image.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkVectorRescaleIntensityImageFilter.h"
#include "itkCastImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Required: input output" << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

Fig. 266: output.png

(continued from previous page)

```
  std::string inputFilename = argv[1];
  std::string outputFilename = argv[2];

  using FloatImageType = itk::Image<itk::CovariantVector<float, 3>, 2>;
  using UnsignedCharImageType = itk::Image<itk::CovariantVector<unsigned char, 3>, 2>;

  using ReaderType = itk::ImageFileReader<FloatImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFilename);
  reader->Update();

  using VectorRescaleFilterType = itk::VectorRescaleIntensityImageFilter
↪<FloatImageType, UnsignedCharImageType>;
  VectorRescaleFilterType::Pointer rescaleFilter = VectorRescaleFilterType::New();
  rescaleFilter->SetInput(reader->GetOutput());
  rescaleFilter->SetOutputMaximumMagnitude(255);
  rescaleFilter->Update();

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFilename);
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class VectorRescaleIntensityImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TInputImage*, *TOutputImage*, Fun
Applies a linear transformation to the magnitude of pixel vectors in a vector Image.

VectorRescaleIntensityImageFilter applies pixel-wise a linear transformation to the intensity values of input image pixels. The linear transformation is defined by the user in terms of the maximum magnitude value of the vectors in the pixels that the output image should have.

All computations are performed in the precision of the input pixel's RealType. Before assigning the computed value to the output pixel.

**See** RescaleIntensityImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Transform Magnitude Of Vector Valued Image Pixels

See itk::VectorRescaleIntensityImageFilter for additional documentation.

### 3.4.18 ImageLabel

#### Extract Inner and Outer Boundaries of Blobs in Binary Image

#### Synopsis

Extract the inner and outer boundaries of blobs in a binary image.

#### Results

Fig. 267: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkInvertIntensityImageFilter.h"
#include "itkBinaryContourImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif
using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using binaryContourImageFilterType = itk::BinaryContourImageFilter<ImageType,␣
↪ImageType>;

  // Outer boundary
  binaryContourImageFilterType::Pointer binaryContourFilter =␣
↪binaryContourImageFilterType::New();
  binaryContourFilter->SetInput(image);
  binaryContourFilter->SetForegroundValue(0);
  binaryContourFilter->SetBackgroundValue(255);
  binaryContourFilter->Update();

  // Invert the result
  using InvertIntensityImageFilterType = itk::InvertIntensityImageFilter<ImageType>;

  InvertIntensityImageFilterType::Pointer invertIntensityFilter =␣
↪InvertIntensityImageFilterType::New();
  invertIntensityFilter->SetInput(binaryContourFilter->GetOutput());
  invertIntensityFilter->Update();

  ImageType::Pointer outerBoundary = ImageType::New();
  outerBoundary->Graft(invertIntensityFilter->GetOutput());

  // Inner boundary
  binaryContourFilter->SetForegroundValue(255);
  binaryContourFilter->SetBackgroundValue(0);
  binaryContourFilter->Update();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(image.GetPointer());
  viewer.AddImage(outerBoundary.GetPointer());
  viewer.AddImage(binaryContourFilter->GetOutput());
  viewer.Visualize();
```

(continues on next page)

(continued from previous page)

```
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 5; r < 10; r++)
  {
    for (unsigned int c = 5; c < 10; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryContourImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>, **protected** itk::Scar

Labels the pixels on the border of the objects in a binary image.

BinaryContourImageFilter takes a binary image as input, where the pixels in the objects are the pixels with a value equal to ForegroundValue. Only the pixels on the contours of the objects are kept. The pixels not on the border are changed to BackgroundValue.

The connectivity can be changed to minimum or maximum connectivity with SetFullyConnected(). Full connectivity produces thicker contours.

https://www.insight-journal.org/browse/publication/217

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelContourImageFilter BinaryErodeImageFilter SimpleContourExtractorImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Boundaries Of Connected Regions In Binary Image

- Extract Inner And Outer Boundaries Of Blobs In Binary Image

See itk::BinaryContourImageFilter for additional documentation.

### Extract Boundaries of Connected Regions in Binary Image

### Synopsis

Extract the boundaries of connected regions in a binary image.

### Results

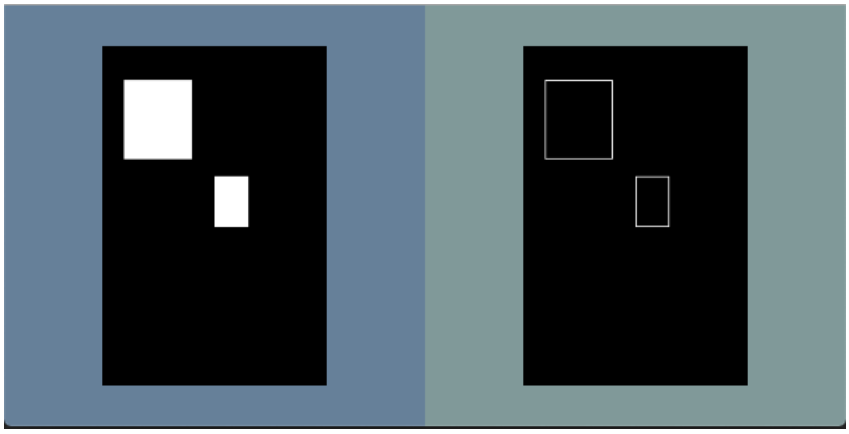


Fig. 268: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkBinaryContourImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);
```

(continues on next page)

(continued from previous page)

```
  using binaryContourImageFilterType = itk::BinaryContourImageFilter<ImageType,␣
↪ImageType>;

  binaryContourImageFilterType::Pointer binaryContourFilter =␣
↪binaryContourImageFilterType::New();
  binaryContourFilter->SetInput(image);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage<ImageType>(image);
  viewer.AddImage<ImageType>(binaryContourFilter->GetOutput());
  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
```

(continues on next page)

(continued from previous page)

```
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryContourImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>, **protected** itk::Scar
Labels the pixels on the border of the objects in a binary image.

BinaryContourImageFilter takes a binary image as input, where the pixels in the objects are the pixels with a value equal to ForegroundValue. Only the pixels on the contours of the objects are kept. The pixels not on the border are changed to BackgroundValue.

The connectivity can be changed to minimum or maximum connectivity with SetFullyConnected(). Full connectivity produces thicker contours.

https://www.insight-journal.org/browse/publication/217

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelContourImageFilter BinaryErodeImageFilter SimpleContourExtractorImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Extract Boundaries Of Connected Regions In Binary Image
- Extract Inner And Outer Boundaries Of Blobs In Binary Image

See itk::BinaryContourImageFilter for additional documentation.

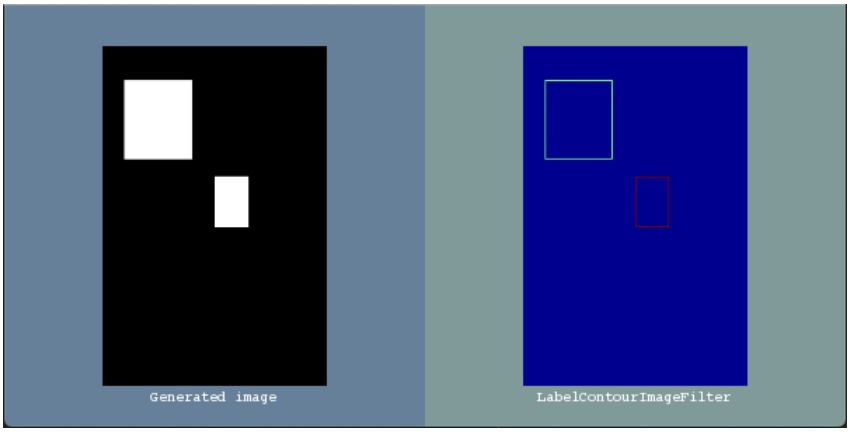
### Label Contours of Connect Components

### Synopsis

Label the contours of connected components.

### Results

Generated Output.




## Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkLabelContourImageFilter.h"
#include "itkConnectedComponentImageFilter.h"
#include "itkScalarToRGBColormapImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

namespace
{
using PixelType = unsigned char;
using RGBPixelType = itk::RGBPixel<unsigned char>;
using ImageType = itk::Image<PixelType, 2>;
using RGBImageType = itk::Image<RGBPixelType, 2>;
} // namespace

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  // Create or read an image
  ImageType::Pointer image;
  if (argc < 2)
  {
    image = ImageType::New();
    CreateImage(image.GetPointer());
  }
  else
  {
```

(continues on next page)

(continued from previous page)

```
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();

    image = reader->GetOutput();
  }

  // Generate connected components
  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<ImageType, ImageType>;
  ConnectedComponentImageFilterType::Pointer connectedComponentImageFilter =␣
↪ConnectedComponentImageFilterType::New();
  connectedComponentImageFilter->SetInput(image);

  // Generate contours for each component
  using LabelContourImageFilterType = itk::LabelContourImageFilter<ImageType,␣
↪ImageType>;
  LabelContourImageFilterType::Pointer labelContourImageFilter =␣
↪LabelContourImageFilterType::New();
  labelContourImageFilter->SetInput(connectedComponentImageFilter->GetOutput());

  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<ImageType, RGBImageType>;
  RGBFilterType::Pointer rgbFilter = RGBFilterType::New();
  rgbFilter->SetInput(labelContourImageFilter->GetOutput());
  rgbFilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::RGBColormapFilter::
↪Jet);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "LabelContourImageFilter";
  viewer.AddRGBImage(rgbFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;
```

(continues on next page)

(continued from previous page)

```
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LabelContourImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>, **protected** itk::Scanli

Labels the pixels on the border of the objects in a labeled image.

LabelContourImageFilter takes a labeled image as input, where the pixels in the objects are the pixels with a value different of the BackgroundValue. Only the pixels on the contours of the objects are kept. The pixels not on the border are changed to BackgroundValue. The labels of the object are the same in the input and in the output image.

The connectivity can be changed to minimum or maximum connectivity with SetFullyConnected(). Full connectivity produces thicker contours.

https://www.insight-journal.org/browse/publication/217

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** BinaryContourImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Label Contours Of Connect Components

See itk::LabelContourImageFilter for additional documentation.

### 3.4.19 ImageStatistics

#### Adaptive Histogram Equalization Image Filter

#### Synopsis

Apply a power law adaptive histogram equalization controlled by the parameters alpha and beta.

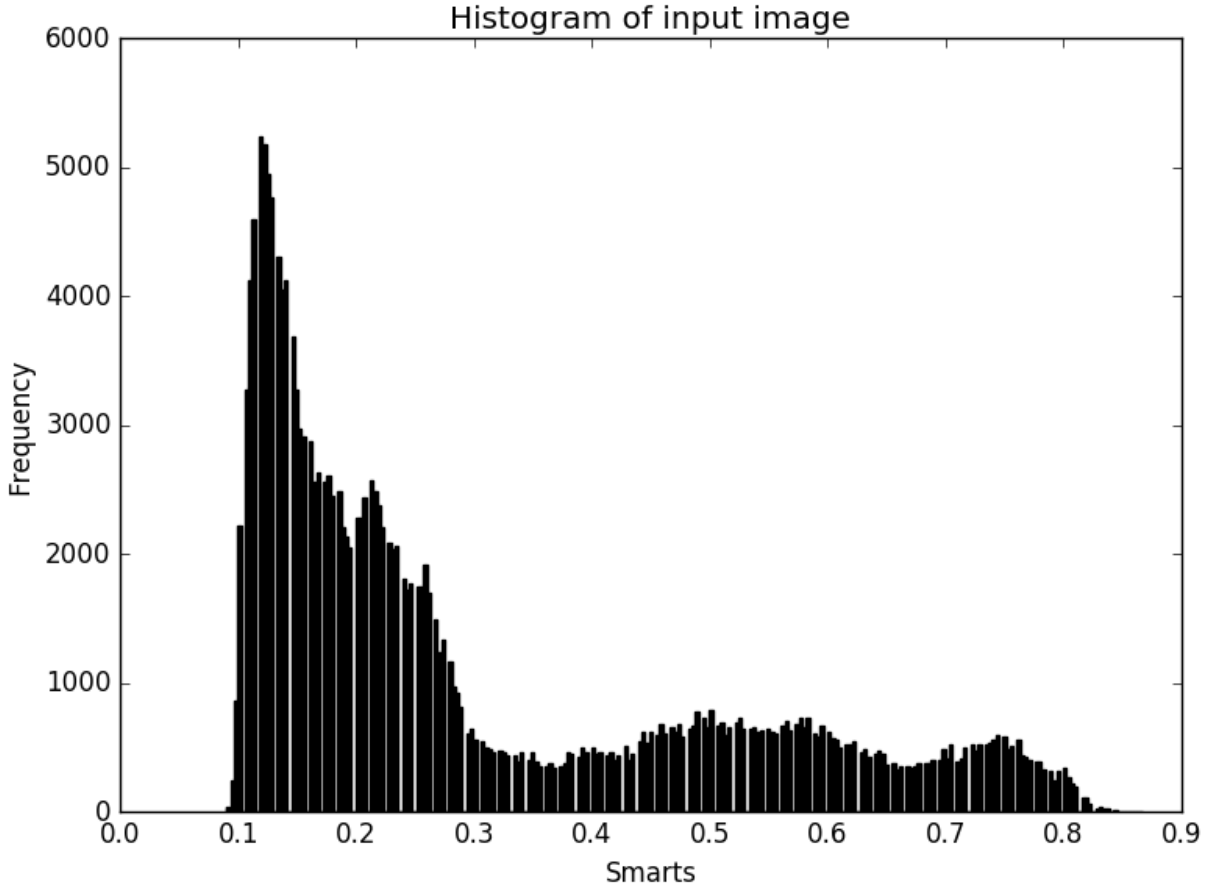
The parameter `alpha` controls how much the filter acts like the classical histogram equalization method (`alpha = 0`) to how much the filter acts like an unsharp mask (`alpha = 1`).

The parameter `beta` controls how much the filter acts like an unsharp mask (`beta = 0`) to much the filter acts like pass through (`beta = 1, with alpha = 1`).

The parameter window (or radius) controls the size of the region over which local statistics are calculated.

### Results

Input image.




Input image histogram.

### Code

### C++

```
#include "itkAdaptiveHistogramEqualizationImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc < 6)
  {
    std::cerr << "Missing parameters." << std::endl;
    std::cerr << "Usage: " << argv[0] << " inputImageFile outputImageFile alpha beta␣
↪radius" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using FileReaderType = itk::ImageFileReader<ImageType>;
  FileReaderType::Pointer reader = FileReaderType::New();
  reader->SetFileName(argv[1]);

  using AdaptiveHistogramEqualizationImageFilterType = itk::
↪AdaptiveHistogramEqualizationImageFilter<ImageType>;
  AdaptiveHistogramEqualizationImageFilterType::Pointer␣
↪adaptiveHistogramEqualizationImageFilter =
    AdaptiveHistogramEqualizationImageFilterType::New();

  float alpha = std::stod(argv[3]);
  adaptiveHistogramEqualizationImageFilter->SetAlpha(alpha);

  float beta = std::stod(argv[4]);
  adaptiveHistogramEqualizationImageFilter->SetBeta(beta);

  int                                                    radiusSize = std::
↪stoi(argv[5]);
  AdaptiveHistogramEqualizationImageFilterType::ImageSizeType radius;
  radius.Fill(radiusSize);
  adaptiveHistogramEqualizationImageFilter->SetRadius(radius);

  adaptiveHistogramEqualizationImageFilter->SetInput(reader->GetOutput());

  adaptiveHistogramEqualizationImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(adaptiveHistogramEqualizationImageFilter->GetOutput());

  writer->Update();
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Adaptive Histogram Equalization Image Filter."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("alpha", type=float)
parser.add_argument("beta", type=float)
parser.add_argument("radius", type=int)
args = parser.parse_args()

Dimension = 2

PixelType = itk.ctype("unsigned char")
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

histogramEqualization = itk.AdaptiveHistogramEqualizationImageFilter.New(reader)
histogramEqualization.SetAlpha(args.alpha)
histogramEqualization.SetBeta(args.beta)

radius = itk.Size[Dimension]()
radius.Fill(args.radius)
histogramEqualization.SetRadius(radius)

itk.imwrite(histogramEqualization, args.output_image)
```

### Classes demonstrated

template<typename **TImageType**, typename **TKernel** = Neighborhood<bool, *TImageType*::ImageDimension>>
**class AdaptiveHistogramEqualizationImageFilter** : **public** itk::MovingHistogramImageFilter<*TImageType*, *TImag*

Power Law Adaptive Histogram Equalization.

Histogram equalization modifies the contrast in an image. The AdaptiveHistogramEqualizationImageFilter is a superset of many contrast enhancing filters. By modifying its parameters (alpha, beta, and window), the AdaptiveHistogramEqualizationImageFilter can produce an adaptively equalized histogram or a version of unsharp mask (local mean subtraction). Instead of applying a strict histogram equalization in a window about a pixel, this filter prescribes a mapping function (power law) controlled by the parameters alpha and beta.

The parameter alpha controls how much the filter acts like the classical histogram equalization method (alpha=0) to how much the filter acts like an unsharp mask (alpha=1).

The parameter beta controls how much the filter acts like an unsharp mask (beta=0) to much the filter acts like pass through (beta=1, with alpha=1).

The parameter window controls the size of the region over which local statistics are calculated. The size of the window is controlled by SetRadius the default Radius is 5 in all directions.

By altering alpha, beta and window, a host of equalization and unsharp masking filters is available.

The boundary condition ignores the part of the neighborhood outside the image, and over-weights the valid part of the neighborhood.

For detail description, reference "Adaptive Image Contrast

Enhancement using Generalizations of Histogram Equalization." J.Alex Stark. IEEE Transactions on Image Processing, May 2000.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Adaptive Histogram Equalization Image Filter

See itk::AdaptiveHistogramEqualizationImageFilter for additional documentation.

### Apply Accumulate Image Filter

### Synopsis

Accumulate pixels of an image along a selected direction.

### Results

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkAccumulateImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <Dimension>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  auto         accumulateDimension = static_cast<unsigned int>(std::stoi(argv[3]));
```

(continues on next page)

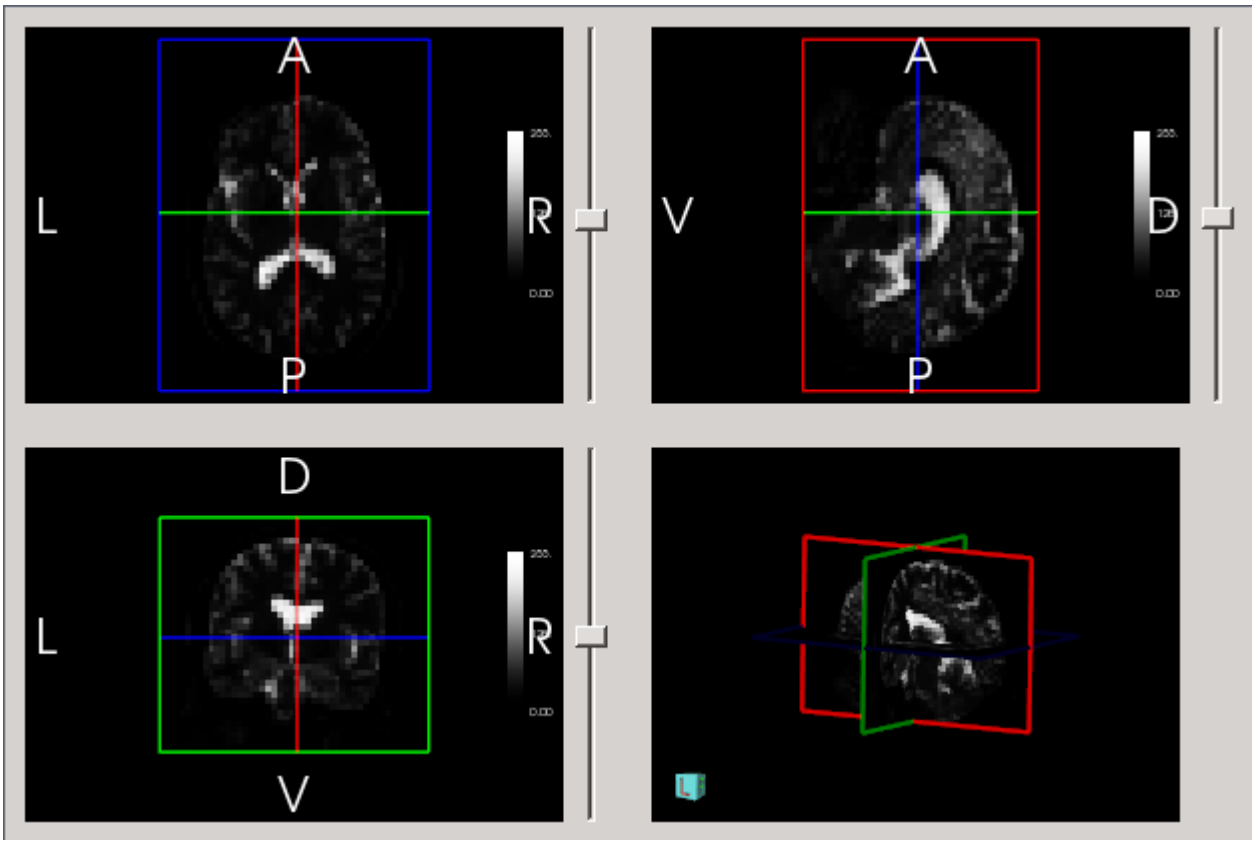



Fig. 269: Input image

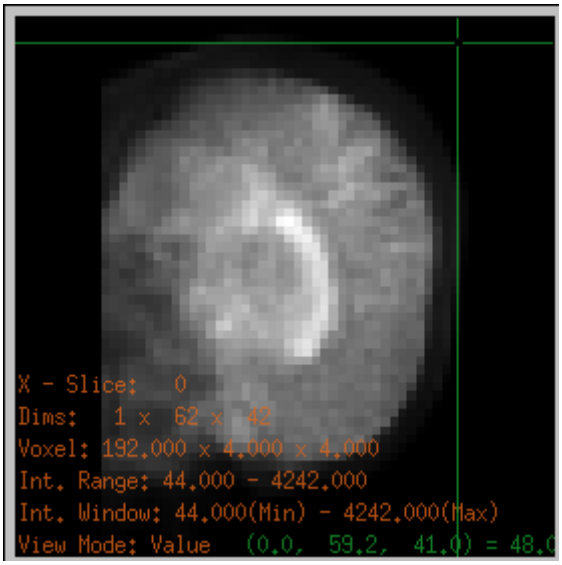



Fig. 270: Output image accumulated along direction 0

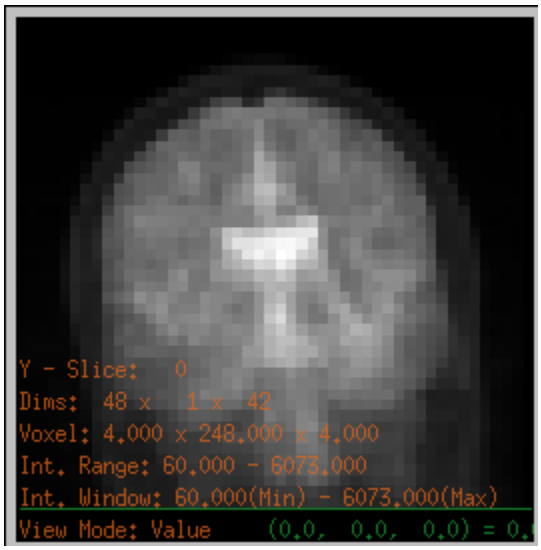



Fig. 271: Output image accumulated along direction 1

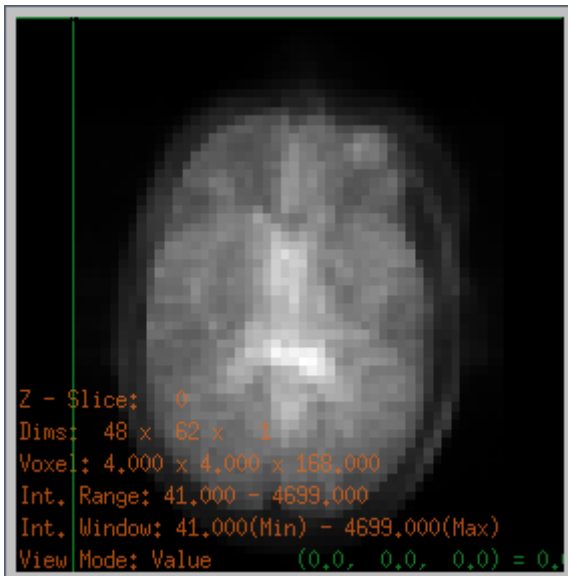



Fig. 272: Output image accumulated along direction 2

(continued from previous page)

```
  constexpr unsigned int Dimension = 3;

  using InputPixelType = unsigned char;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using OutputPixelType = double;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using FilterType = itk::AccumulateImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetAccumulateDimension(accumulateDimension);

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class AccumulateImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Implements an accumulation of an image along a selected direction.

This class accumulates an image along a dimension and reduce the size of this dimension to 1. The dimension being accumulated is set by AccumulateDimension.

Each pixel is the cumulative sum of the pixels along the collapsed dimension and reduce the size of the accumulated dimension to 1 (only on the accumulated).

The dimensions of the InputImage and the OutputImage must be the same.

This class is parameterized over the type of the input image and the type of the output image.

This filter was contributed by Emiliano Beronich

**Author** Emiliano Beronich

**See** GetAverageSliceImageFilter

Subclassed by itk::GetAverageSliceImageFilter< TInputImage, TOutputImage >

See itk::AccumulateImageFilter for additional documentation.

### Compute Min, Max, Variance and Mean of Image

### Synopsis

Compute min, max, variance and mean of an Image.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkStatisticsImageFilter.h"
#include "itkImageFileReader.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using StatisticsImageFilterType = itk::StatisticsImageFilter<ImageType>;
  StatisticsImageFilterType::Pointer statisticsImageFilter =␣
↪StatisticsImageFilterType::New();
  statisticsImageFilter->SetInput(image);
  statisticsImageFilter->Update();

  std::cout << "Mean: " << statisticsImageFilter->GetMean() << std::endl;
  std::cout << "Std.: " << statisticsImageFilter->GetSigma() << std::endl;
  std::cout << "Min: " << statisticsImageFilter->GetMinimum() << std::endl;
  std::cout << "Max: " << statisticsImageFilter->GetMaximum() << std::endl;

  return EXIT_SUCCESS;
}


void
CreateImage(ImageType::Pointer image)
{
```

(continues on next page)

(continued from previous page)

```
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class StatisticsImageFilter** : **public** itk::ImageSink<*TInputImage*>

Compute min, max, variance and mean of an Image.

StatisticsImageFilter computes the minimum, maximum, sum, sum of squares, mean, variance sigma of an image. The filter needs all of its input image. It behaves as a filter with an input and output. Thus it can be inserted in a pipline with other filters and the statistics will only be recomputed if a downstream filter changes.

This filter is automatically multi-threaded and can stream its input when NumberOfStreamDivisions is set to

more than one. Statistics are independently computed for each streamed and threaded region then merged.

Internally a compensated summation algorithm is used for the accumulation of intensities to improve accuracy for large images.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

\sphinxexample{Filtering/ImageStatistics/ComputeMinMaxVarianceMeanOfImage,Compute Min, Max, Variance And Mean Of Image}

See itk::StatisticsImageFilter for additional documentation.

### Compute PCA Shape From Training Sample

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Compute a PCA shape model from a training sample.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
/*
Author: Juan Cardelino <juan dot cardelino at gmail dot com>
*/

// ITK
#include "itkBinaryThresholdImageFilter.h"
#include "itkBoundedReciprocalImageFilter.h"
#include "itkChangeInformationImageFilter.h"
#include "itkCommand.h"
#include "itkCurvatureAnisotropicDiffusionImageFilter.h"
#include "itkEuler2DTransform.h"
#include "itkFastMarchingImageFilter.h"
#include "itkGeodesicActiveContourShapePriorLevelSetImageFilter.h"
#include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
```

(continues on next page)

(continued from previous page)

```
#include "itkImageFileWriter.h"
#include "itkImagePCAShapeModelEstimator.h"
#include "itkMultiplyImageFilter.h"
#include "itkNumericSeriesFileNames.h"
#include "itkNormalVariateGenerator.h"
#include "itkOnePlusOneEvolutionaryOptimizer.h"
#include "itkPCAShapeSignedDistanceFunction.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkShapePriorMAPCostFunction.h"
#include "itkSpatialFunctionImageEvaluatorFilter.h"

// VNL
#include "vnl/vnl_sample.h"

int
main(int argc, char * argv[])
{
  if (argc < 5)
  {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " nbTrain  trainFilePattern";
    std::cerr << " nbModes  modeFilePattern";
    std::cerr << std::endl;
    return 1;
  }

  for (int i = 0; i < argc; i++)
  {
    std::cout << "id: " << i << " arg: " << argv[i] << std::endl;
  }
  constexpr unsigned int Dimension = 2;
  using my_PixelType = float;
  using ImageType = itk::Image<my_PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  using WriterType = itk::ImageFileWriter<ImageType>;
  using ScaleType = itk::MultiplyImageFilter<ImageType, ImageType, ImageType>;

  unsigned int nb_train = atoi(argv[1]);

  itk::NumericSeriesFileNames::Pointer fileNamesCreator = itk::NumericSeriesFileNames:
↪:New();
  std::vector<ImageType::Pointer>      trainingImages(nb_train);

  fileNamesCreator->SetStartIndex(0);
  fileNamesCreator->SetEndIndex(nb_train - 1);
  fileNamesCreator->SetSeriesFormat(argv[2]);
  const std::vector<std::string> & shapeModeFileNames = fileNamesCreator->
↪GetFileNames();

  for (unsigned int k = 0; k < nb_train; k++)
  {
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(shapeModeFileNames[k].c_str());
    reader->Update();
    trainingImages[k] = reader->GetOutput();
```

(continues on next page)

(continued from previous page)

```
  }

  using my_Estimatortype = itk::ImagePCAShapeModelEstimator<ImageType, ImageType>;
  my_Estimatortype::Pointer filter = my_Estimatortype::New();
  filter->SetNumberOfTrainingImages(nb_train);
  filter->SetNumberOfPrincipalComponentsRequired(2);

  for (unsigned int k = 0; k < nb_train; k++)
  {
    filter->SetInput(k, trainingImages[k]);
  }

  unsigned int nb_modes = atoi(argv[3]);

  itk::NumericSeriesFileNames::Pointer fileNamesOutCreator = itk::
↪NumericSeriesFileNames::New();

  fileNamesOutCreator->SetStartIndex(0);
  fileNamesOutCreator->SetEndIndex(nb_modes - 1);
  fileNamesOutCreator->SetSeriesFormat(argv[4]);
  const std::vector<std::string> & outFileNames = fileNamesOutCreator->GetFileNames();

  ScaleType::Pointer scaler = ScaleType::New();

  filter->Update();
  my_Estimatortype::VectorOfDoubleType v = filter->GetEigenValues();
  double                               sv_mean = sqrt(v[0]);

  for (unsigned int k = 0; k < nb_modes; k++)
  {
    double sv = sqrt(v[k]);
    double sv_n = sv / sv_mean;
    // double sv_n=sv;
    std::cout << "writing: " << outFileNames[k] << std::endl;

    std::cout << "svd[" << k << "]: " << sv << " norm: " << sv_n << std::endl;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName(outFileNames[k].c_str());
    scaler->SetInput(filter->GetOutput(k));
    scaler->SetConstant(sv_n);
    writer->SetInput(scaler->GetOutput());
    writer->Update();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<double, *TInputImage*::ImageDimension>>
**class ImagePCAShapeModelEstimator** : **public** itk::ImageShapeModelEstimatorBase<*TInputImage*, *TOutputImage*>
Base class for ImagePCAShapeModelEstimator object.

itkImagePCAShapeModelEstimator performs a principal component analysis (PCA) on a set of images. The user specifies the number of training images and also the number of desired largest principal components needed. The ITK pipeline mechanism sets up the storage for both input and output images. The number of output images are the user specified number of desired largest principal components plus 1 (for the mean image).

The algorithm uses the VNL library to perform the eigen analysis. To speed the computation of the instead of performing the eigen analysis of the covariance vector A*A' where A is a matrix with p x t, p = number of pixels or voxels in each images and t = number of training images, we calculate the eigen vectors of the inner product matrix A'*A. The resulting eigen vectors (E) are then multiplied with the the matrix A to get the principal components. The covariance matrix has a dimension of p x p. Since number of pixels in any image being typically very high the eigen decomposition becomes computationally expensive. The inner product on the other hand has the dimension of t x t, where t is typically much smaller that p. Hence the eigen decomposition (most compute intensive part) is an orders of magnitude faster.

The Update() function enables the calculation of the various models, creates the membership function objects and populates them.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute PCA Shape From Training Sample

See itk::ImagePCAShapeModelEstimator for additional documentation.

### Statistical Properties of Labeled Regions

### Synopsis

Get statistical properties of labeled regions in an image.

### Results

Fig. 273: image.png

Output:

```
Number of labels: 2

min: 255
max: 255
median: 0
mean: 255
sigma: 0
variance: 0
```

(continues on next page)

(continued from previous page)

```
sum: 4080
count: 16
region: ImageRegion (0x7ffedfd04a28)
Dimension: 2
Index: [6, 6]
Size: [4, 4]

min: 0
max: 0
median: 0
mean: 0
sigma: 0
variance: 0
sum: 0
count: 384
region: ImageRegion (0x7ffedfd04a28)
Dimension: 2
Index: [0, 0]
Size: [20, 20]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelStatisticsImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  using LabelMapToLabelImageFilterType =
    itk::LabelMapToLabelImageFilter<BinaryImageToLabelMapFilterType::OutputImageType,␣
↪ImageType>;
  LabelMapToLabelImageFilterType::Pointer labelMapToLabelImageFilter =␣
↪LabelMapToLabelImageFilterType::New();
  labelMapToLabelImageFilter->SetInput(binaryImageToLabelMapFilter->GetOutput());
  labelMapToLabelImageFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  using LabelStatisticsImageFilterType = itk::LabelStatisticsImageFilter<ImageType,␣
↪ImageType>;
  LabelStatisticsImageFilterType::Pointer labelStatisticsImageFilter =␣
↪LabelStatisticsImageFilterType::New();
  labelStatisticsImageFilter->SetLabelInput(labelMapToLabelImageFilter->GetOutput());
  labelStatisticsImageFilter->SetInput(image);
  labelStatisticsImageFilter->Update();

  std::cout << "Number of labels: " << labelStatisticsImageFilter->
↪GetNumberOfLabels() << std::endl;
  std::cout << std::endl;

  using LabelPixelType = LabelStatisticsImageFilterType::LabelPixelType;

  for (auto vIt = labelStatisticsImageFilter->GetValidLabelValues().begin();
       vIt != labelStatisticsImageFilter->GetValidLabelValues().end();
       ++vIt)
  {
    if (labelStatisticsImageFilter->HasLabel(*vIt))
    {
      LabelPixelType labelValue = *vIt;
      std::cout << "min: " << labelStatisticsImageFilter->GetMinimum(labelValue) <<␣
↪std::endl;
      std::cout << "max: " << labelStatisticsImageFilter->GetMaximum(labelValue) <<␣
↪std::endl;
      std::cout << "median: " << labelStatisticsImageFilter->GetMedian(labelValue) <<␣
↪std::endl;
      std::cout << "mean: " << labelStatisticsImageFilter->GetMean(labelValue) << std:
↪:endl;
      std::cout << "sigma: " << labelStatisticsImageFilter->GetSigma(labelValue) <<␣
↪std::endl;
      std::cout << "variance: " << labelStatisticsImageFilter->
↪GetVariance(labelValue) << std::endl;
      std::cout << "sum: " << labelStatisticsImageFilter->GetSum(labelValue) << std::
↪endl;
      std::cout << "count: " << labelStatisticsImageFilter->GetCount(labelValue) <<␣
↪std::endl;
      // std::cout << "box: " << labelStatisticsImageFilter->GetBoundingBox(␣
↪labelValue ) << std::endl; // can't output
      // a box
      std::cout << "region: " << labelStatisticsImageFilter->GetRegion(labelValue) <<␣
↪std::endl;
      std::cout << std::endl << std::endl;
    }
  }

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);
```

(continues on next page)

(continued from previous page)

```
  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TLabelImage**>
**class LabelStatisticsImageFilter** : **public** itk::ImageSink<*TInputImage*>

Given an intensity image and a label map, compute min, max, variance and mean of the pixels associated with each label or segment.

LabelStatisticsImageFilter computes the minimum, maximum, sum, mean, median, variance and sigma of regions of an intensity image, where the regions are defined via a label map (a second input). The label image should be integral type. The filter needs all of its input image. It behaves as a filter with an input and output. Thus it can be inserted in a pipeline with other filters and the statistics will only be recomputed if a downstream filter changes.

Optionally, the filter also computes intensity histograms on each object. If histograms are enabled, a median intensity value can also be computed, although its accuracy is limited to the bin width of the histogram. If histograms are not enabled, the median returns zero.

This filter is automatically multi-threaded and can stream its input when NumberOfStreamDivisions is set to more than

a. Statistics are independently computed for each streamed and threaded region then merged.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Statistical Properties Of Labeled Regions

See itk::LabelStatisticsImageFilter for additional documentation.

### Calculate Image Moments

### Calculate Image Moments of an Ellipse

Image moments can be used to determine the center of gravity or average pixel location of a binary object as well as the orientation of an object according to its axes of variation.

The interactive plot below demonstrates how to calculate the image moments of an ellipse-defined mask.

[1]:
```
import itk
import matplotlib.pyplot as plt
import numpy as np
from ipywidgets import interact
```

[2]:
```
output_size = [100, 100] # size of binary image containing the ellipse

@interact(tx=(0,output_size[0],1), ty=(0,output_size[1],1), a=(5,50,1), b=(5,50,1),␣
↪theta=(0,2*np.pi,0.1))
def foo(tx=30, ty=50, a=5, b=10, theta=np.pi/4.0):
    '''
    Creates a binary image of an ellipse and calculates the image moments.  Major␣
↪(purple) and minor (green)
    principal axes are displayed.

    Parameters
    ==========
    tx, ty : translation x and y
    a, b : ellipse horizontal and vertical widths before rotation
    theta : angle of rotation (radians)
    '''

    # ellipse starts as unit circle, use world transform to define final ellipse
    ellipse = itk.EllipseSpatialObject[2].New()
    ellipse_transform = itk.AffineTransform[itk.D, 2].New()
    ellipse_transform.Scale([a, b])
    ellipse_transform.Rotate2D(theta)
    ellipse_transform.Translate([tx, ty])
    ellipse.SetObjectToWorldTransform(ellipse_transform)

    ellipse_img = itk.spatial_object_to_image_filter(input=ellipse, inside_value=1,␣
↪outside_value=0, size=output_size)

    momentor = itk.ImageMomentsCalculator.New(Image=ellipse_img)
    momentor.Compute()

    centroid = momentor.GetCenterOfGravity()
    prin_axes = itk.array_from_matrix(momentor.GetPrincipalAxes())
    minor_axes = prin_axes[0]
```

(continues on next page)

(continued from previous page)

```
    major_axes = prin_axes[1]

    fig, ax = plt.subplots(figsize=[8,8])
    plt.imshow(ellipse_img, cmap='gray')
    plt.scatter(centroid[0], centroid[1])

    minor_pt = centroid + minor_axes*5
    plt.plot([centroid[0], minor_pt[0]], [centroid[1], minor_pt[1]], color='green')

    major_pt = centroid + major_axes*5
    plt.plot([centroid[0], major_pt[0]], [centroid[1], major_pt[1]], color='purple')

    print(momentor)
```

```
interactive(children=(IntSlider(value=30, description='tx'), IntSlider(value=50,␣
↪description='ty'), IntSlider(...
```

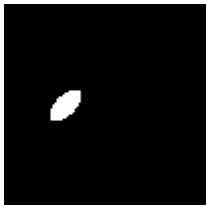
```
[ ]:
```

## Synopsis

Calculate image moments from binary images and an interactive ellipse.

## Results

Input Image



```
Output
 ImageMomentsCalculator (000002037138BE90)
   RTTI typeinfo:   class itk::ImageMomentsCalculator<class itk::Image<unsigned short,
↪2> >
   Reference Count: 1
   Modified Time: 249
   Debug: Off
   Object Name:
   Observers:
     none
   Image: 000002037138FDD0
   Valid: 1
   Zeroth Moment about origin: 153
   First Moment about origin: [30, 50]
   Second Moment about origin: 15.0458 -8.8366
 -8.8366 15.0458
```

(continues on next page)

(continued from previous page)

```
  Center of Gravity: [30, 50]
  Second central moments: 15.0458 -8.8366
-8.8366 15.0458

  Principal Moments: [950, 3654]
  Principal axes: 0.707107 0.707107
-0.707107 0.707107
```

### Jupyter Notebook

launch binder

### Code

### Python

```python
#!/usr/bin/env python

import sys
import itk
import argparse

parser = argparse.ArgumentParser(description="Calculate Image Moments.")
parser.add_argument("input_image")

args = parser.parse_args()
input_image = itk.imread(args.input_image)
momentor = itk.ImageMomentsCalculator.New(Image=input_image)
momentor.Compute()
print(momentor)
```

### C++

```cpp
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImageMomentsCalculator.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>" << std::endl;
    std::cerr << "Prints image moments from unsigned short binary image.";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  const char * inputFileName = argv[1];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned short;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  reader->Update();

  // clang-format off
  using FilterType = itk::ImageMomentsCalculator<ImageType>;
  // clang-format on
  FilterType::Pointer filter = FilterType::New();
  filter->SetImage(reader->GetOutput());
  filter->Compute();
  filter->Print(std::cout, 0);

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TImage**>
**class ImageMomentsCalculator** : **public** itk::*Object*

Compute moments of an n-dimensional image.

This class provides methods for computing the moments and related properties of a single-echo image. Computing the (non-central) moments of a large image can easily take a million times longer than computing the various other values derived from them, so we compute the moments only on explicit request, and save their values (in an ImageMomentsCalculator object) for later retrieval by the user.

The non-central moments computed by this class are not really intended for general use and are therefore in index coordinates; that is, we pretend that the index that selects a particular pixel also equals its physical coordinates. The center of gravity, central moments, principal moments and principal axes are all more generally useful and are computed in the physical coordinates defined by the Origin and Spacing parameters of the image.

The methods that return values return the values themselves rather than references because the cost is small compared to the cost of computing the moments and doing so simplifies memory management for the caller.

See itk::ImageMomentsCalculator for additional documentation.

### 3.4.20 LabelMap

#### Apply Morphological Closing on All Label Objects

#### Synopsis

Apply morphological closing operation on all LabelObjects of a given LabelMap.

In details:

- read image is converted to LabelMap

- apply morphological closing operation on all LabelObjects of the LabelMap
- make sure there is no overlapping LabelObject with using LabelUniqueLabelMapFilter
- convert the LabelMap to LabelImage
- write the corresponding LabelImage

### Results

Fig. 274: Input image

Fig. 275: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkObjectByObjectLabelMapFilter.h"
#include "itkLabelUniqueLabelMapFilter.h"
#include "itkFlatStructuringElement.h"
#include "itkBinaryMorphologicalClosingImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " <InputFileName> <OutputFileName> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char *       inputFileName = argv[1];
  const char *       outputFileName = argv[2];
  const unsigned int radiusValue = std::stoi(argv[3]);

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using LabelObjectType = itk::LabelObject<PixelType, Dimension>;
  using LabelMapType = itk::LabelMap<LabelObjectType>;

  using LabelImageToLabelMapFilterType = itk::LabelImageToLabelMapFilter<ImageType,␣
↪LabelMapType>;
  LabelImageToLabelMapFilterType::Pointer labelMapConverter =␣
↪LabelImageToLabelMapFilterType::New();
  labelMapConverter->SetInput(reader->GetOutput());
  labelMapConverter->SetBackgroundValue(itk::NumericTraits<PixelType>::Zero);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);

  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using MorphologicalFilterType =
    itk::BinaryMorphologicalClosingImageFilter<ImageType, ImageType,␣
↪StructuringElementType>;
  MorphologicalFilterType::Pointer closingFilter = MorphologicalFilterType::New();

  using ObjectByObjectLabelMapFilterType = itk::ObjectByObjectLabelMapFilter
↪<LabelMapType>;
  ObjectByObjectLabelMapFilterType::Pointer objectByObjectLabelMapFilter =␣
↪ObjectByObjectLabelMapFilterType::New();
  objectByObjectLabelMapFilter->SetInput(labelMapConverter->GetOutput());
  objectByObjectLabelMapFilter->SetBinaryInternalOutput(true);
  objectByObjectLabelMapFilter->SetFilter(closingFilter);

  using UniqueLabelMapFilterType = itk::LabelUniqueLabelMapFilter<LabelMapType>;
  UniqueLabelMapFilterType::Pointer unique = UniqueLabelMapFilterType::New();
  unique->SetInput(objectByObjectLabelMapFilter->GetOutput());

  using LabelMapToLabelImageFilterType = itk::LabelMapToLabelImageFilter<LabelMapType,
↪ ImageType>;
  LabelMapToLabelImageFilterType::Pointer labelImageConverter =␣
↪LabelMapToLabelImageFilterType::New();
  labelImageConverter->SetInput(unique->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(labelImageConverter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*, typename **TInputFilter** = *ImageToImageFilter*<
**class ObjectByObjectLabelMapFilter** : **public** itk::LabelMapFilter<*TInputImage*, *TOutputImage*>

ObjectByObjectLabelMapFilter applies an image pipeline to all the objects of a label map and produce a new label map.

The image pipeline can simply produce a modified object or produce several objects from the single input object. Several options are provided to handle the different cases.

KeepLabel, which defaults to true, makes the filter try to keep as much as possible the labels of the original objects. If an image pipeline produce several objects the label of the input object is assigned to the first output object. The other output objects get another label not present in the input image. When KeepLabel is set to false, all the objects are relabeled in the order of apparition during the filter process.

BinaryInternalOutput can be set to true if the image pipeline produce binary output image. In that case, the objects produced are identified with a connected component algorithm before being reinserted in the output label map. InternalForegroundValue can be set to a specific value which represent the foreground value in the binary image.

PadSize and ConstrainPaddingToImage can be used to extend the size of the image to process passed to the image pipeline. This is useful if the image pipeline is known to be able to enlarge the object. The padding can be constrained to the input label map region by setting ConstrainPaddingToImage to true - this parameter can make a difference for the algorithm with a different behavior on the border of the image. By default, the image is padded by 1 pixel and constrained to the image region.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Note** : When applying a single filter, input and output filters are the same; while applying a pipeline, input and output filters are different, may not even be of the same type. It is the responsibility of the user to connect the pipeline properly outside of this filter.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Morphological Closing On All Label Objects

See itk::ObjectByObjectLabelMapFilter for additional documentation.

template<typename **TImage**>
**class LabelUniqueLabelMapFilter** : **public** itk::AttributeUniqueLabelMapFilter<*TImage*, Functor::LabelLabelObjectAcc
Make sure that the objects are not overlapping.

AttributeUniqueLabelMapFilter search the overlapping zones in the overlapping objects and keeps only a single object on all the pixels of the image. The object to keep is selected according to their label.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** AttributeLabelObject

See itk::LabelUniqueLabelMapFilter for additional documentation.

### Apply Morphological Closing on Specific Label Object

### Synopsis

Apply BinaryMorphologicalClosingFilter on one LabelObject of given LabelMap.

In details:

- read one Image
- convert the LabelImage into one LabelMap
- extract the LabelObject of interest using LabelSelectionLabelMapFilter
- apply morphological closing operation on the LabelObject of interest
- merge the processed LabelObject with remaining LabelObjects from LabelSelectionLabelMapFilter
- make sure there is no overlapping LabelObject with using LabelUniqueLabelMapFilter
- convert the LabelMap to LabelImage
- write the corresponding LabelImage

### Results

Fig. 276: Input image

Fig. 277: Output image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelSelectionLabelMapFilter.h"
#include "itkMergeLabelMapFilter.h"
#include "itkObjectByObjectLabelMapFilter.h"
#include "itkLabelUniqueLabelMapFilter.h"
#include "itkFlatStructuringElement.h"
#include "itkBinaryMorphologicalClosingImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <label> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  const char *       inputFileName = argv[1];
  const char *       outputFileName = argv[2];
  const auto         label = static_cast<PixelType>(std::stoi(argv[3]));
  const unsigned int radiusValue = std::stoi(argv[4]);

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using LabelObjectType = itk::LabelObject<PixelType, Dimension>;
  using LabelMapType = itk::LabelMap<LabelObjectType>;
```

(continues on next page)

(continued from previous page)

```
  using LabelImageToLabelMapFilterType = itk::LabelImageToLabelMapFilter<ImageType,␣
↪LabelMapType>;
  LabelImageToLabelMapFilterType::Pointer labelMapConverter =␣
↪LabelImageToLabelMapFilterType::New();
  labelMapConverter->SetInput(reader->GetOutput());
  labelMapConverter->SetBackgroundValue(itk::NumericTraits<PixelType>::Zero);

  using SelectorType = itk::LabelSelectionLabelMapFilter<LabelMapType>;
  SelectorType::Pointer selector = SelectorType::New();
  selector->SetInput(labelMapConverter->GetOutput());
  selector->SetLabel(label);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);

  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using MorphologicalFilterType =
    itk::BinaryMorphologicalClosingImageFilter<ImageType, ImageType,␣
↪StructuringElementType>;
  MorphologicalFilterType::Pointer closingFilter = MorphologicalFilterType::New();

  using ObjectByObjectLabelMapFilterType = itk::ObjectByObjectLabelMapFilter
↪<LabelMapType>;
  ObjectByObjectLabelMapFilterType::Pointer objectByObjectLabelMapFilter =␣
↪ObjectByObjectLabelMapFilterType::New();
  objectByObjectLabelMapFilter->SetInput(selector->GetOutput(0));
  objectByObjectLabelMapFilter->SetBinaryInternalOutput(true);
  objectByObjectLabelMapFilter->SetFilter(closingFilter);

  using MergeLabelFilterType = itk::MergeLabelMapFilter<LabelMapType>;
  MergeLabelFilterType::Pointer merger = MergeLabelFilterType::New();
  merger->SetInput(0, objectByObjectLabelMapFilter->GetOutput(0));
  merger->SetInput(1, selector->GetOutput(1));
  merger->SetMethod(itk::MergeLabelMapFilterEnums::ChoiceMethod::KEEP);

  using UniqueLabelMapFilterType = itk::LabelUniqueLabelMapFilter<LabelMapType>;
  UniqueLabelMapFilterType::Pointer unique = UniqueLabelMapFilterType::New();
  unique->SetInput(merger->GetOutput());

  using LabelMapToLabelImageFilterType = itk::LabelMapToLabelImageFilter<LabelMapType,
↪ ImageType>;
  LabelMapToLabelImageFilterType::Pointer labelImageConverter =␣
↪LabelMapToLabelImageFilterType::New();
  labelImageConverter->SetInput(unique->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(labelImageConverter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*, typename **TInputFilter** = *ImageToImageFilter*<
**class ObjectByObjectLabelMapFilter** : **public** itk::LabelMapFilter<*TInputImage*, *TOutputImage*>

ObjectByObjectLabelMapFilter applies an image pipeline to all the objects of a label map and produce a new label map.

The image pipeline can simply produce a modified object or produce several objects from the single input object. Several options are provided to handle the different cases.

KeepLabel, which defaults to true, makes the filter try to keep as much as possible the labels of the original objects. If an image pipeline produce several objects the label of the input object is assigned to the first output object. The other output objects get another label not present in the input image. When KeepLabel is set to false, all the objects are relabeled in the order of apparition during the filter process.

BinaryInternalOutput can be set to true if the image pipeline produce binary output image. In that case, the objects produced are identified with a connected component algorithm before being reinserted in the output label map. InternalForegroundValue can be set to a specific value which represent the foreground value in the binary image.

PadSize and ConstrainPaddingToImage can be used to extend the size of the image to process passed to the image pipeline. This is useful if the image pipeline is known to be able to enlarge the object. The padding can be constrained to the input label map region by setting ConstrainPaddingToImage to true - this parameter can make a difference for the algorithm with a different behavior on the border of the image. By default, the image is padded by 1 pixel and constrained to the image region.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Note** : When applying a single filter, input and output filters are the same; while applying a pipeline, input and output filters are different, may not even be of the same type. It is the responsibility of the user to connect the pipeline properly outside of this filter.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Apply Morphological Closing On All Label Objects

See itk::ObjectByObjectLabelMapFilter for additional documentation.

### Convert itk::Image With Labels to Label Map

### Synopsis

Convert an itk::Image consisting of labeled regions to a LabelMap.

### Results

Output:

```
There are 2 objects.
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkConnectedComponentImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<ImageType, ImageType>;
  ConnectedComponentImageFilterType::Pointer connectedComponentImageFilter =␣
↪ConnectedComponentImageFilterType::New();
  connectedComponentImageFilter->SetInput(image);
  connectedComponentImageFilter->Update();

  using LabelImageToLabelMapFilterType = itk::LabelImageToLabelMapFilter<ImageType>;
  LabelImageToLabelMapFilterType::Pointer labelImageToLabelMapFilter =␣
↪LabelImageToLabelMapFilterType::New();
  labelImageToLabelMapFilter->SetInput(connectedComponentImageFilter->GetOutput());
  labelImageToLabelMapFilter->Update();

  // The output of this filter is an itk::LabelMap, which contains itk::LabelObject's
  std::cout << "There are " << labelImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects() << " objects."
            << std::endl;

  return EXIT_SUCCESS;
}
```

(continues on next page)

(continued from previous page)

```
void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *LabelMap*<LabelObject<**typename** *TInputImage*::PixelType, *TI*
**class LabelImageToLabelMapFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
convert a labeled image to a label collection image

LabelImageToLabelMapFilter converts a label image to a label collection image. The labels are the same in the input and the output image.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** BinaryImageToLabelMapFilter, LabelMapToLabelImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert itk::Image With Labels To Label Map

See itk::LabelImageToLabelMapFilter for additional documentation.

### Convert Image With Labeled Regions to ShapeLabelMap

### Synopsis

Convert an itk::Image consisting of labeled regions to a ShapeLabelMap.

### Results

Output:

```
There are 2 objects.
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkLabelImageToShapeLabelMapFilter.h"
#include "itkConnectedComponentImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
```

(continues on next page)

(continued from previous page)

```
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<ImageType, ImageType>;
  ConnectedComponentImageFilterType::Pointer connectedComponentImageFilter =␣
↪ConnectedComponentImageFilterType::New();
  connectedComponentImageFilter->SetInput(image);
  connectedComponentImageFilter->Update();

  using LabelImageToShapeLabelMapFilterType = itk::LabelImageToShapeLabelMapFilter
↪<ImageType>;
  LabelImageToShapeLabelMapFilterType::Pointer labelImageToShapeLabelMapFilter =
    LabelImageToShapeLabelMapFilterType::New();
  labelImageToShapeLabelMapFilter->SetInput(connectedComponentImageFilter->
↪GetOutput());
  labelImageToShapeLabelMapFilter->Update();

  // The output of this filter is an itk::LabelMap, which contains itk::LabelObject's
  std::cout << "There are " << labelImageToShapeLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects() << " objects."
            << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;
```

(continues on next page)

(continued from previous page)

```
      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *LabelMap*<ShapeLabelObject<**typename** *TInputImage*::PixelTy
**class LabelImageToShapeLabelMapFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Converts a label image to a label map and valuates the shape attributes.

A convenient class that converts a label image to a label map and valuates the shape attribute at once.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, LabelShapeOpeningImageFilter, LabelStatisticsOpeningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Shape Attributes For Binary Image
- Convert Image With Labeled Regions To ShapeLabelMap

See itk::LabelImageToShapeLabelMapFilter for additional documentation.

### Convert Label Map to Normal Image

### Synopsis

Convert a LabelMap to a normal image with different values representing each region

### Results

Fig. 278: image.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  using LabelMapToLabelImageFilterType =
    itk::LabelMapToLabelImageFilter<BinaryImageToLabelMapFilterType::OutputImageType,␣
↪ImageType>;
  LabelMapToLabelImageFilterType::Pointer labelMapToLabelImageFilter =␣
↪LabelMapToLabelImageFilterType::New();
  labelMapToLabelImageFilter->SetInput(binaryImageToLabelMapFilter->GetOutput());
  labelMapToLabelImageFilter->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
```

(continues on next page)

(continued from previous page)

```
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LabelMapToLabelImageFilter** : **public** itk::LabelMapFilter<*TInputImage*, *TOutputImage*>

Converts a LabelMap to a labeled image.

LabelMapToBinaryImageFilter to a label image.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelMapToBinaryImageFilter, LabelMapMaskImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert Label Map To Normal Image

See itk::LabelMapToLabelImageFilter for additional documentation.

### Extract Given Label Object

### Synopsis

Extract one given LabelObject from one LabelMap into a new LabelMap and remaining ones into another one.

### Results

Fig. 279: Input image

Fig. 280: first output (i.e. LabelMap with the LabelObject of interest)

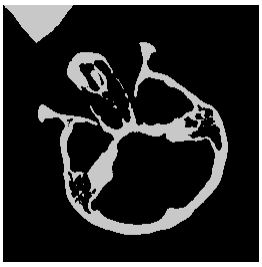


Fig. 281: second output (i.e. LabelMap with remaining LabelObjects)

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelObject.h"
#include "itkLabelMap.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkLabelSelectionLabelMapFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName1> <OutputFileName2> <Label>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  const char * inputFileName = argv[1];
  const char * outputFileName[2];
  outputFileName[0] = argv[2];
  outputFileName[1] = argv[3];
  const auto label = static_cast<PixelType>(std::stoi(argv[4]));

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using LabelObjectType = itk::LabelObject<PixelType, Dimension>;
  using LabelMapType = itk::LabelMap<LabelObjectType>;

  using LabelImageToLabelMapFilterType = itk::LabelImageToLabelMapFilter<ImageType,␣
↪LabelMapType>;
  LabelImageToLabelMapFilterType::Pointer labelMapConverter =␣
↪LabelImageToLabelMapFilterType::New();
  labelMapConverter->SetInput(reader->GetOutput());
  labelMapConverter->SetBackgroundValue(itk::NumericTraits<PixelType>::Zero);

  using SelectorType = itk::LabelSelectionLabelMapFilter<LabelMapType>;
  SelectorType::Pointer selector = SelectorType::New();
  selector->SetInput(labelMapConverter->GetOutput());
  selector->SetLabel(label);

  for (int i = 0; i < 2; i++)
  {
    using LabelMapToLabelImageFilterType = itk::LabelMapToLabelImageFilter
↪<LabelMapType, ImageType>;
```

(continues on next page)

(continued from previous page)

```
    LabelMapToLabelImageFilterType::Pointer labelImageConverter =␣
↪LabelMapToLabelImageFilterType::New();
    labelImageConverter->SetInput(selector->GetOutput(i));

    using WriterType = itk::ImageFileWriter<ImageType>;
    WriterType::Pointer writer = WriterType::New();
    writer->SetFileName(outputFileName[i]);
    writer->SetInput(labelImageConverter->GetOutput());
    try
    {
      writer->Update();
    }
    catch (itk::ExceptionObject & error)
    {
      std::cerr << "Error: " << error << std::endl;
      return EXIT_FAILURE;
    }
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class LabelSelectionLabelMapFilter** : **public** itk::AttributeSelectionLabelMapFilter<*TImage*, Functor::LabelLabelObj
remove the objects according to the value of their attribute

LabelSelectionLabelMapFilter removes the objects in a label collection image with an attribute value inside or outside a set of attribute values passed by the user. The attribute is provide by an attribute accessor given in template parameter. Contrary to the other filters made to remove some object of a LabelMap, no ordering relation for the attribute is needed in that filter.

This code was contributed in the Insight Journal paper: "Label object representation and manipulation with ITK" by Lehmann G. https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** AttributeLabelObject

See itk::LabelSelectionLabelMapFilter for additional documentation.

### Invert Image Using Binary Not Operation

### Synopsis

Invert an image using the Binary Not operation.

### Results

Fig. 282: input.png

Fig. 283: output.png

### Code

#### C++

```
#include "itkImage.h"
#if ITK_VERSION_MAJOR >= 4
#  include "itkBinaryNotImageFilter.h"
#endif
#include "itkImageRegionIterator.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
#if ITK_VERSION_MAJOR >= 4
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();

  using BinaryNotImageFilterType = itk::BinaryNotImageFilter<ImageType>;

  BinaryNotImageFilterType::Pointer binaryNotFilter = BinaryNotImageFilterType::New();
  binaryNotFilter->SetInput(image);
  binaryNotFilter->Update();

  writer->SetFileName("output.png");
  writer->SetInput(binaryNotFilter->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  writer->Update();
#endif

  return EXIT_SUCCESS;
}
void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 50)
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}
```

## Classes demonstrated

template<typename **TImage**>
**class BinaryNotImageFilter** : **public** itk::*UnaryFunctorImageFilter*<*TImage*, *TImage*, Functor::BinaryNot<*TImage*::Pixel

Implements the BinaryNot logical operator pixel-wise between two images.

This class is parameterized over the types of the two input images and the type of the output image. Numeric conversions (castings) are done by the C++ defaults.

The total operation over one pixel will be

output_pixel = static_cast<PixelType>( input1_pixel != input2_pixel )

Where "!=" is the equality operator in C++.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Invert Image Using Binary Not Operation

See itk::BinaryNotImageFilter for additional documentation.

### Keep Binary Regions That Meet Specific Properties

### Synopsis

Keep only regions that meet a specified threshold of a specified property.

### Results

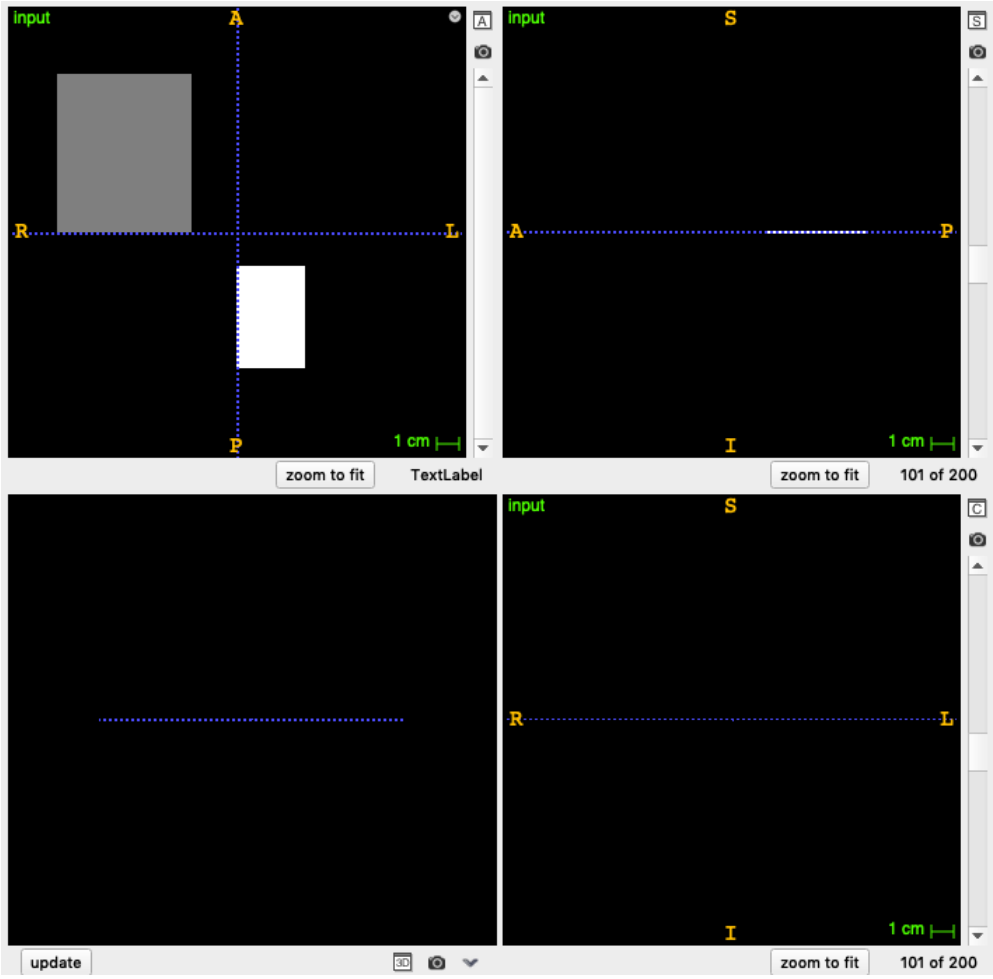



Fig. 284: input.mhd

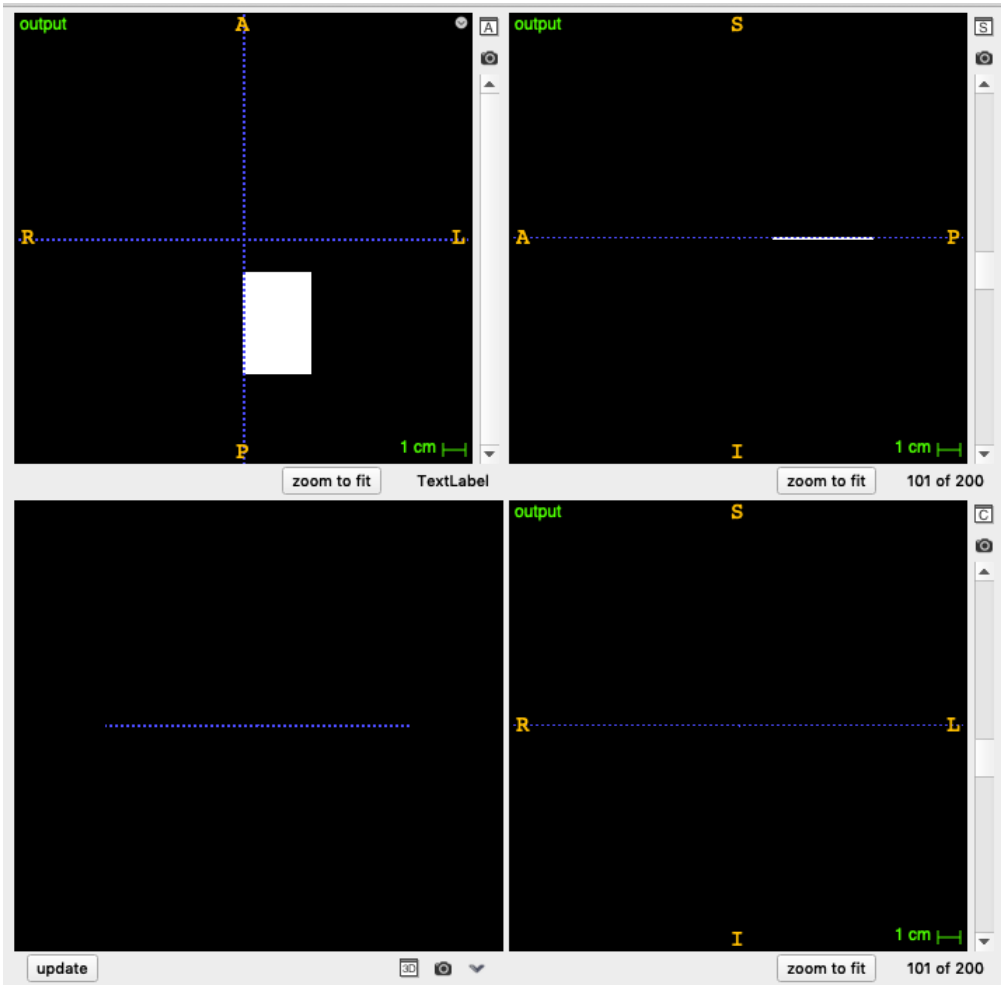



Fig. 285: output.mhd

### Code

### C++

```
#include "itkBinaryStatisticsOpeningImageFilter.h"
#include "itkImageFileWriter.h"

using ImageType = itk::Image<unsigned char, 2>;
void
CreateImage(ImageType::Pointer image1, ImageType::Pointer image2);

int
main(int, char *[])
{
  ImageType::Pointer binaryImage = ImageType::New();
  ImageType::Pointer featureImage = ImageType::New();

  CreateImage(binaryImage, featureImage);

  using BinaryOpeningType = itk::BinaryStatisticsOpeningImageFilter<ImageType,␣
↪ImageType>;
  BinaryOpeningType::Pointer opening = BinaryOpeningType::New();
  opening->SetInput(binaryImage);
  opening->SetFeatureImage(featureImage);
  opening->SetBackgroundValue(0);
  opening->SetForegroundValue(255);
  opening->SetLambda(150);
  opening->SetAttribute(BinaryOpeningType::LabelObjectType::MEAN);
  opening->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.mhd");
  writer->SetInput(featureImage);
  writer->Update();
  writer->SetFileName("output.mhd");
  writer->SetInput(opening->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image1, ImageType::Pointer image2)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 200;

  region.SetSize(size);
  region.SetIndex(start);
```

(continues on next page)

(continued from previous page)

```
  image1->SetRegions(region);
  image1->Allocate();
  image1->FillBuffer(0);

  image2->SetRegions(region);
  image2->Allocate();
  image2->FillBuffer(0);

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image1->SetPixel(pixelIndex, 255);
      image2->SetPixel(pixelIndex, 100);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image1->SetPixel(pixelIndex, 255);
      image2->SetPixel(pixelIndex, 200);
    }
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TFeatureImage**>
**class BinaryStatisticsOpeningImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TInputImage*>

Remove objects based on the value of their Statistics attribute.

The BinaryStatisticsOpeningImageFilter removes the objects in a binary image with an attribute value smaller or greater than a threshold called Lambda. The attributes are those of the StatisticsLabelObject.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** StatisticsLabelObject, LabelStatisticsOpeningImageFilter, BinaryStatisticsOpeningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Keep Binary Regions That Meet Specific Properties

See itk::BinaryStatisticsOpeningImageFilter for additional documentation.

### Keep Regions Above Certain Level

### Synopsis

Keep only regions that rank above a certain level of a particular property.

### Results

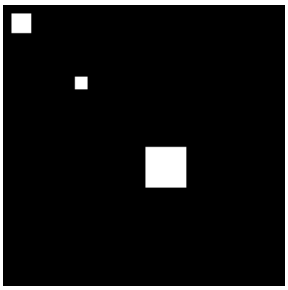


Fig. 286: input.png

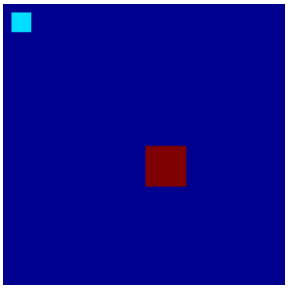


Fig. 287: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToLabelMapFilter.h"
#include "itkLabelShapeKeepNObjectsImageFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkScalarToRGBColormapImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
```

(continues on next page)

(continued from previous page)

```
using LabelImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  using LabelMapToLabelImageFilterType =
    itk::LabelMapToLabelImageFilter<BinaryImageToLabelMapFilterType::OutputImageType,␣
↪LabelImageType>;
  LabelMapToLabelImageFilterType::Pointer labelMapToLabelImageFilter =␣
↪LabelMapToLabelImageFilterType::New();
  labelMapToLabelImageFilter->SetInput(binaryImageToLabelMapFilter->GetOutput());
  labelMapToLabelImageFilter->Update();

  using LabelShapeKeepNObjectsImageFilterType = itk::LabelShapeKeepNObjectsImageFilter
↪<LabelImageType>;
  LabelShapeKeepNObjectsImageFilterType::Pointer labelShapeKeepNObjectsImageFilter =
    LabelShapeKeepNObjectsImageFilterType::New();
  labelShapeKeepNObjectsImageFilter->SetInput(labelMapToLabelImageFilter->
↪GetOutput());
  labelShapeKeepNObjectsImageFilter->SetBackgroundValue(0);
  labelShapeKeepNObjectsImageFilter->SetNumberOfObjects(2);
  // KeepNObjects->ReverseOrderingOn();
  // labelShapeKeepNObjectsImageFilter->SetAttribute( LabelObjectType::PERIMETER );
  labelShapeKeepNObjectsImageFilter->
↪SetAttribute(LabelShapeKeepNObjectsImageFilterType::LabelObjectType::PERIMETER);
  labelShapeKeepNObjectsImageFilter->Update();

  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, 2>;

  // Color each label/object a different color
  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<LabelImageType,␣
↪RGBImageType>;
  RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
  colormapImageFilter->SetInput(labelShapeKeepNObjectsImageFilter->GetOutput());
  colormapImageFilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::
↪RGBColormapFilter::Jet);
  colormapImageFilter->Update();

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(colormapImageFilter->GetOutput());
  writer->SetFileName("output.png");
  writer->Update();

  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with three white squares
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(200);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if (((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 20) &&
         (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 20)) ||
        ((imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 60) &&
         (imageIterator.GetIndex()[1] > 50 && imageIterator.GetIndex()[1] < 60)) ||
        ((imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 130) &&
         (imageIterator.GetIndex()[1] > 100 && imageIterator.GetIndex()[1] < 130)))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class LabelShapeKeepNObjectsImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TInputImage*>
keep N objects according to their shape attributes

LabelShapeKeepNObjectsImageFilter keep the N objects in a labeled image with the highest (or lowest) attribute value. The attributes are the ones of the ShapeLabelObject.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, BinaryShapeKeepNObjectsImageFilter, LabelStatisticsKeepNObjectsImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Keep Regions Above Certain Level

See itk::LabelShapeKeepNObjectsImageFilter for additional documentation.

### Keep Regions That Meet Specific Properties

### Synopsis

Keep only regions that meet a specified threshold of a specified property.

### Results

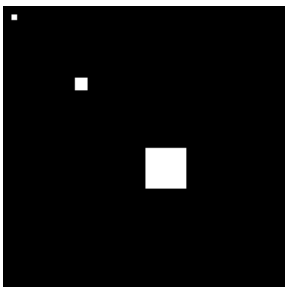


Fig. 288: input.png

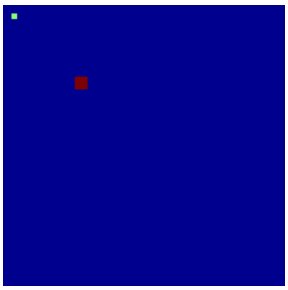


Fig. 289: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToShapeLabelMapFilter.h"
#include "itkShapeOpeningLabelMapFilter.h"
#include "itkLabelMapToLabelImageFilter.h"
#include "itkScalarToRGBColormapImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
using LabelImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  // Create a ShapeLabelMap from the image
  using BinaryImageToShapeLabelMapFilterType = itk::BinaryImageToShapeLabelMapFilter
↪<ImageType>;
  BinaryImageToShapeLabelMapFilterType::Pointer binaryImageToShapeLabelMapFilter =
    BinaryImageToShapeLabelMapFilterType::New();
  binaryImageToShapeLabelMapFilter->SetInput(image);
  binaryImageToShapeLabelMapFilter->Update();

  // Remove label objects that have PERIMETER greater than 50
  using ShapeOpeningLabelMapFilterType =
    itk::ShapeOpeningLabelMapFilter<BinaryImageToShapeLabelMapFilterType::
↪OutputImageType>;
  ShapeOpeningLabelMapFilterType::Pointer shapeOpeningLabelMapFilter =␣
↪ShapeOpeningLabelMapFilterType::New();
  shapeOpeningLabelMapFilter->SetInput(binaryImageToShapeLabelMapFilter->GetOutput());
  shapeOpeningLabelMapFilter->SetLambda(50);
  shapeOpeningLabelMapFilter->ReverseOrderingOn();
  shapeOpeningLabelMapFilter->SetAttribute(ShapeOpeningLabelMapFilterType::
↪LabelObjectType::PERIMETER);
  shapeOpeningLabelMapFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  // Create a label image
  using LabelMapToLabelImageFilterType =
    itk::LabelMapToLabelImageFilter<BinaryImageToShapeLabelMapFilterType::
↪OutputImageType, LabelImageType>;
  LabelMapToLabelImageFilterType::Pointer labelMapToLabelImageFilter =␣
↪LabelMapToLabelImageFilterType::New();
  labelMapToLabelImageFilter->SetInput(shapeOpeningLabelMapFilter->GetOutput());
  labelMapToLabelImageFilter->Update();

  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, 2>;

  // Color each label/object a different color
  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<LabelImageType,␣
↪RGBImageType>;
  RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
  colormapImageFilter->SetInput(labelMapToLabelImageFilter->GetOutput());
  colormapImageFilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::
↪RGBColormapFilter::Jet);
  colormapImageFilter->Update();

  // Write the output
  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(colormapImageFilter->GetOutput());
  writer->SetFileName("output.png");
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with three white squares
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(200);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if (((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
         (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10)) ||
        ((imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 60) &&
```

(continues on next page)

(continued from previous page)

```
          (imageIterator.GetIndex()[1] > 50 && imageIterator.GetIndex()[1] < 60)) ||
         ((imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 130) &&
          (imageIterator.GetIndex()[1] > 100 && imageIterator.GetIndex()[1] < 130)))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TImage**>
**class ShapeOpeningLabelMapFilter** : **public** itk::InPlaceLabelMapFilter<*TImage*>
Remove objects according to the value of their shape attribute.

ShapeOpeningLabelMapFilter removes objects in a label collection image with an attribute value smaller or greater than a threshold called Lambda. The attributes are those of the ShapeLabelObject.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, BinaryShapeOpeningImageFilter, LabelStatisticsOpeningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Keep Regions That Meet Specific Properties

Subclassed by itk::StatisticsOpeningLabelMapFilter< TImage >

See itk::ShapeOpeningLabelMapFilter for additional documentation.

### Label Binary Regions and Get Properties

### Synopsis

Label binary regions in an image and get their properties.

### Results

Fig. 290: image.png

Output:

```
There are 1 objects.
Object 0 has bounding box ImageRegion (0x7ff924a4e558)
Dimension: 2
Index: [6, 6]
Size: [4, 4]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToShapeLabelMapFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToShapeLabelMapFilterType = itk::BinaryImageToShapeLabelMapFilter
↪<ImageType>;
  BinaryImageToShapeLabelMapFilterType::Pointer binaryImageToShapeLabelMapFilter =
    BinaryImageToShapeLabelMapFilterType::New();
  binaryImageToShapeLabelMapFilter->SetInput(image);
  binaryImageToShapeLabelMapFilter->Update();

  // The output of this filter is an itk::ShapeLabelMap, which contains itk::
↪ShapeLabelObject's
  std::cout << "There are " << binaryImageToShapeLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects() << " objects."
            << std::endl;
```

(continues on next page)

(continued from previous page)

```
  // Loop over all of the blobs
  for (unsigned int i = 0; i < binaryImageToShapeLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects(); i++)
  {
    BinaryImageToShapeLabelMapFilterType::OutputImageType::LabelObjectType *␣
↪labelObject =
      binaryImageToShapeLabelMapFilter->GetOutput()->GetNthLabelObject(i);
    // Output the bounding box (an example of one possible property) of the ith region
#if ITK_VERSION_MAJOR >= 4
    std::cout << "Object " << i << " has bounding box " << labelObject->
↪GetBoundingBox() << std::endl;
#else
    std::cout << "Object " << i << " has region " << labelObject->GetRegion() << std::
↪endl;
#endif
  }

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
```

(continues on next page)

(continued from previous page)

```
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *LabelMap*<ShapeLabelObject<SizeValueType, *TInputImage*::Imag
**class BinaryImageToShapeLabelMapFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Converts a binary image to a label map and valuate the shape attributes.

A convenient class that converts a binary image to a label map and valuates the shape attributes at once.

The GetOutput() function returns an itk::ShapeLabelMap. A typical use would be to iterate over the ShapeLabelObjects in the map, using something like this:

```
for(unsigned int i = 0; i < filter->GetOutput()->GetNumberOfLabelObjects(); ++i)
  {
  FilterType::OutputImageType::LabelObjectType* shapeLabelObject =
    filter->GetOutput()->GetLabelObject(i);
  // Here you can get properties of the ShapeLabelObject
  std::cout << "Bounding box: " << shapeLabelObject->GetBoundingBox();
  }
```

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, LabelShapeOpeningImageFilter, BinaryStatisticsOpeningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Label Binary Regions And Get Properties

See itk::BinaryImageToShapeLabelMapFilter for additional documentation.

### Label Binary Regions in Image

### Synopsis

Label binary regions in an image.

### Results

Fig. 291: image.png

Output:

```
There are 1 objects.
Object 0 contains pixel [6, 6]
Object 0 contains pixel [7, 6]
Object 0 contains pixel [8, 6]
Object 0 contains pixel [9, 6]
Object 0 contains pixel [6, 7]
Object 0 contains pixel [7, 7]
Object 0 contains pixel [8, 7]
Object 0 contains pixel [9, 7]
Object 0 contains pixel [6, 8]
Object 0 contains pixel [7, 8]
Object 0 contains pixel [8, 8]
Object 0 contains pixel [9, 8]
Object 0 contains pixel [6, 9]
Object 0 contains pixel [7, 9]
Object 0 contains pixel [8, 9]
Object 0 contains pixel [9, 9]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToLabelMapFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  // The output of this filter is an itk::LabelMap, which contains itk::LabelObject's
  std::cout << "There are " << binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects() << " objects."
```

(continues on next page)

(continued from previous page)

```
            << std::endl;

  // Loop over each region
  for (unsigned int i = 0; i < binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects(); i++)
  {
    // Get the ith region
    BinaryImageToLabelMapFilterType::OutputImageType::LabelObjectType * labelObject =
      binaryImageToLabelMapFilter->GetOutput()->GetNthLabelObject(i);

    // Output the pixels composing the region
    for (unsigned int pixelId = 0; pixelId < labelObject->Size(); pixelId++)
    {
      std::cout << "Object " << i << " contains pixel " << labelObject->
↪GetIndex(pixelId) << std::endl;
    }
  }

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10))
    {
      imageIterator.Set(255);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
```

(continues on next page)

(continued from previous page)

```
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *LabelMap*<LabelObject<SizeValueType, *TInputImage*::ImageDim
**class BinaryImageToLabelMapFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>, **protected** itk:

Label the connected components in a binary image and produce a collection of label objects.

BinaryImageToLabelMapFilter labels the objects in a binary image. Each distinct object is assigned a unique label. The final object labels start with 1 and are consecutive. Objects that are reached earlier by a raster order scan have a lower label.

The GetOutput() function of this class returns an itk::LabelMap.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ConnectedComponentImageFilter, LabelImageToLabelMapFilter, LabelMap, LabelObject

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Label Binary Regions In Image

See itk::BinaryImageToLabelMapFilter for additional documentation.

### Mask One Image Given Label Map

### Synopsis

Mask the content of one input itk::Image according to one input itk::LabelMap

### Results

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkLabelObject.h"
#include "itkLabelMap.h"
#include "itkLabelImageToLabelMapFilter.h"
#include "itkLabelMapMaskImageFilter.h"
```

(continues on next page)

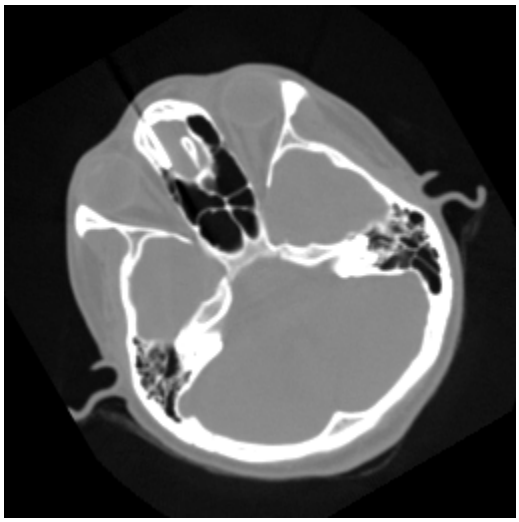


Fig. 292: Input image

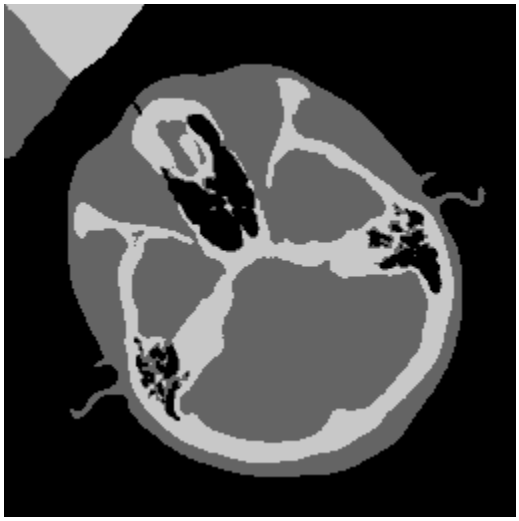


Fig. 293: Input label image

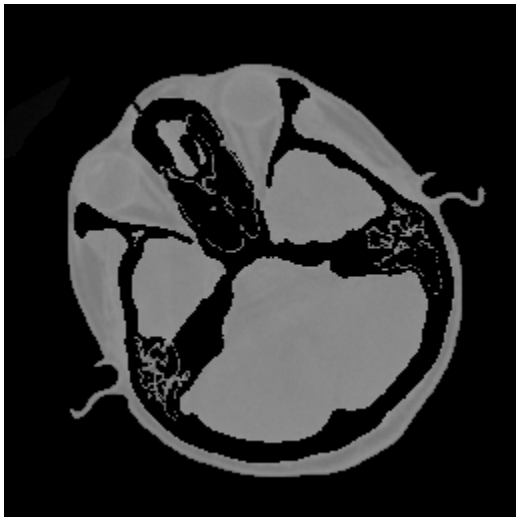


Fig. 294: Masked output with label 100, background 0.

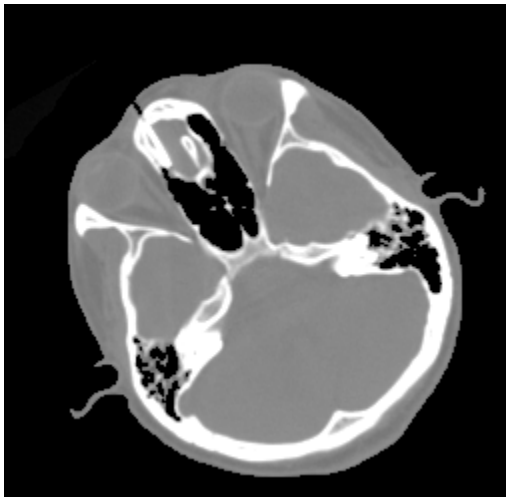


Fig. 295: Masked output with label 0, background 0, negated, cropped, and border size 10.

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if ((argc != 6) && (argc != 9))
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <LabelMapFileName> <OutputFileName> <Label>
↪<Background>";
    std::cerr << " [<negated (bool)> <crop (bool)> <border size>]";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  const char *              inputFileName = argv[1];
  const char *              labelMapFileName = argv[2];
  const char *              outputFileName = argv[3];
  const auto                label = static_cast<PixelType>(std::stoi(argv[4]));
  const auto                background = static_cast<PixelType>(std::stoi(argv[5]));
  bool                      negated = false;
  bool                      crop = false;
  ImageType::SizeValueType borderSize = 0;

  if (argc == 9)
  {
    negated = (std::stoi(argv[6]) == 1);
    crop = (std::stoi(argv[7]) == 1);
    borderSize = static_cast<ImageType::SizeValueType>(std::stoi(argv[8]));
  }

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader1 = ReaderType::New();
  reader1->SetFileName(inputFileName);

  ReaderType::Pointer reader2 = ReaderType::New();
  reader2->SetFileName(labelMapFileName);

  using LabelObjectType = itk::LabelObject<PixelType, Dimension>;
  using LabelMapType = itk::LabelMap<LabelObjectType>;

  // convert the label image into a LabelMap
  using LabelImage2LabelMapType = itk::LabelImageToLabelMapFilter<ImageType,␣
↪LabelMapType>;
  LabelImage2LabelMapType::Pointer convert = LabelImage2LabelMapType::New();
  convert->SetInput(reader2->GetOutput());

  using FilterType = itk::LabelMapMaskImageFilter<LabelMapType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(convert->GetOutput());
  filter->SetFeatureImage(reader1->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  // The label to be used to mask the image is passed via SetLabel
  filter->SetLabel(label);

  // The background in the output image (where the image is masked)
  // is passed via SetBackground
  filter->SetBackgroundValue(background);

  // The user can choose to mask the image outside the label object
  // (default behavior), or inside the label object with the chosen label,
  // by calling SetNegated().
  filter->SetNegated(negated);

  // Finally, the image can be cropped to the masked region, by calling
  // SetCrop( true ), or to a region padded by a border, by calling both
  // SetCrop() and SetCropBorder().
  // The crop border defaults to 0, and the image is not cropped by default.
  filter->SetCrop(crop);

  FilterType::SizeType border;
  border.Fill(borderSize);

  filter->SetCropBorder(border);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class LabelMapMaskImageFilter** : **public** itk::LabelMapFilter<*TInputImage*, *TOutputImage*>

Mask and image with a LabelMap.

LabelMapMaskImageFilter mask the content of an input image according to the content of the input LabelMap. The masked pixel of the input image are set to the BackgroundValue. LabelMapMaskImageFilter can keep the input image for one label only, with Negated = false (the default) or it can mask the input image for a single label, when Negated equals true. In Both cases, the label is set with SetLabel().

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** LabelMapToBinaryImageFilter, LabelMapToLabelImageFilter

See itk::LabelMapMaskImageFilter for additional documentation.

### Merge LabelMaps

### Synopsis

Merges several labelmaps.

### Results

Output:

```
number of objects: 4
number of expected objects: 4
```

### Code

### C++

```
#include "itkBinaryImageToShapeLabelMapFilter.h"
#include "itkMergeLabelMapFilter.h"

int
main(int, char *[])
{
  using ImageType = itk::Image<int, 3>;

  // Binary Image to Shape Label Map.
  using BI2SLMType = itk::BinaryImageToShapeLabelMapFilter<ImageType>;
  using LabelMapType = BI2SLMType::OutputImageType;
  using LabelObjectType = BI2SLMType::LabelObjectType;

  using MergerType = itk::MergeLabelMapFilter<LabelMapType>;
  MergerType::Pointer merger = MergerType::New();
  merger->SetMethod(itk::MergeLabelMapFilterEnums::ChoiceMethod::PACK);

  int noObjects = 4;

  for (int i = 1; i <= noObjects; i++)
  {
    LabelMapType::Pointer    labelMap = LabelMapType::New();
    LabelObjectType::Pointer labelObject = LabelObjectType::New();

    labelObject->SetLabel(1);
    labelMap->AddLabelObject(labelObject);
    labelMap->Update();

    merger->SetInput(i - 1, labelMap);
  }
```

(continues on next page)

(continued from previous page)

```
  merger->Update();
  std::cout << "number of objects:  " << merger->GetOutput()->
↪GetNumberOfLabelObjects() << "\n";
  std::cout << "number of expected objects:  " << noObjects << "\n";

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class MergeLabelMapFilter** : **public** itk::InPlaceLabelMapFilter<*TImage*>
Merges several Label Maps.

This filter takes one or more input Label Map and merges them.

SetMethod() can be used to change how the filter manage the labels from the different label maps. KEEP (0): MergeLabelMapFilter do its best to keep the label unchanged, but if a label is already used in a previous label map, a new label is assigned. AGGREGATE (1): If the same label is found several times in the label maps, the label objects with the same label are merged. PACK (2): MergeLabelMapFilter relabel all the label objects by order of processing. No conflict can occur. STRICT (3): MergeLabelMapFilter keeps the labels unchanged and raises an exception if the same label is found in several images.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, RelabelComponentImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Merge LabelMaps

See itk::MergeLabelMapFilter for additional documentation.

### Remove Holes Not Connected to Image Boundaries

### Synopsis

Remove holes in one binary image not connected to its boundary. In this case the foreground value for the image is taken to be zero (black), and everything is filled in within the black boundary.

### Results

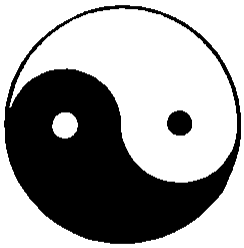


Fig. 296: Input image

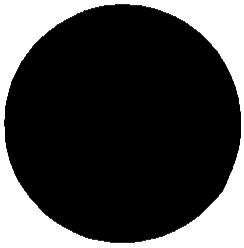


Fig. 297: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Remove Holes Not Connected To Image Boundaries."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

Dimension = 2

PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
```

(continues on next page)

(continued from previous page)

```
reader = ReaderType.New()
reader.SetFileName(args.input_image)

FilterType = itk.BinaryFillholeImageFilter[ImageType]
binaryfillholefilter = FilterType.New()
binaryfillholefilter.SetInput(reader.GetOutput())
binaryfillholefilter.SetForegroundValue(itk.NumericTraits[PixelType].min())

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(binaryfillholefilter.GetOutput())
writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkBinaryFillholeImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::BinaryFillholeImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetForegroundValue(itk::NumericTraits<PixelType>::min());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
```

(continues on next page)

(continued from previous page)

```
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class BinaryFillholeImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TInputImage*>
Remove holes not connected to the boundary of the image.

BinaryFillholeImageFilter fills holes in a binary image.

Geodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille's book "Morphological Image Analysis:

Principles and Applications", Second Edition, Springer, 2003.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** GrayscaleFillholeImageFilter

See itk::BinaryFillholeImageFilter for additional documentation.

### Remove Labels From Label Map

### Synopsis

Remove the labels from a LabelMap.

### Results

Output:

```
There are 10 objects.
There are originally 10 objects.
Removing 5 objects.
There are 5 objects remaining.
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkBinaryImageToLabelMapFilter.h"

using ImageType = itk::Image<unsigned char, 2>;
static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using BinaryImageToLabelMapFilterType = itk::BinaryImageToLabelMapFilter<ImageType>;
  BinaryImageToLabelMapFilterType::Pointer binaryImageToLabelMapFilter =␣
↪BinaryImageToLabelMapFilterType::New();
  binaryImageToLabelMapFilter->SetInput(image);
  binaryImageToLabelMapFilter->Update();

  // The output of this filter is an itk::LabelMap, which contains itk::LabelObject's
  std::cout << "There are " << binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects() << " objects."
            << std::endl;

  std::vector<unsigned long> labelsToRemove;

  std::cout << "There are originally " << binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects()
            << " objects." << std::endl;

  // Note: do NOT remove the labels inside the loop! The IDs are stored such that␣
↪they will change when one is deleted.
  // Instead, mark all of the labels to be removed first and then remove them all␣
↪together.
  for (unsigned int i = 0; i < binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects(); i++)
  {
    // Get the ith region
    BinaryImageToLabelMapFilterType::OutputImageType::LabelObjectType * labelObject =
      binaryImageToLabelMapFilter->GetOutput()->GetNthLabelObject(i);
    // std::cout << "Region " << i << " has " << labelObject->Size() << " pixels." <<␣
↪std::endl;

    // Mark every other label to be removed
    if (i % 2 == 0)
    {
      labelsToRemove.push_back(labelObject->GetLabel());
    }
  }

  std::cout << "Removing " << labelsToRemove.size() << " objects." << std::endl;
```

(continues on next page)

(continued from previous page)

```
  // Remove all regions that were marked for removal.
  for (unsigned long i : labelsToRemove)
  {
    binaryImageToLabelMapFilter->GetOutput()->RemoveLabel(i);
  }

  std::cout << "There are " << binaryImageToLabelMapFilter->GetOutput()->
↪GetNumberOfLabelObjects()
            << " objects remaining." << std::endl;


  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make line of non-touching diagonal pixels
  for (unsigned int i = 0; i < 20; i += 2)
  {
    itk::Index<2> pixel;
    pixel[0] = i;
    pixel[1] = i;
    image->SetPixel(pixel, 255);
  }
}
```

## Classes demonstrated

template<typename **TLabelObject**>
**class LabelMap** : **public** itk::ImageBase<*TLabelObject*::ImageDimension>

Templated n-dimensional image to store labeled objects.

LabelMap is an image class specialized in storing the labeled images. It represent the image in a different way than itk::Image. Instead of storing the content of the image in an array of pixels values, it store the a collection of labeled objects, and a background value. This way of storing the content of the image allow an easy and efficient manipulation of the objects in the image.

The LabelMap shares a lot of methods with the itk::Image class. it make it usable as input or output of the itk::ImageToImageFilter for example. However the methods don't have the same complexity in the 2 classes, because of the different way to store the data. GetPixel() is run in constant time for example in itk::Image, but have a worst case complexity of O(L), where L is the number of lines in the image (imageSize[1] * imageSize[2] for a 3D image).

To iterate over the LabelObjects in the map, use:

```
for(unsigned int i = 0; i < filter->GetOutput()->GetNumberOfLabelObjects(); ++i)
  {
  FilterType::OutputImageType::LabelObjectType* labelObject =
    filter->GetOutput()->GetNthLabelObject(i);
  }
```

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Remove Labels From Label Map

See itk::LabelMap for additional documentation.

### Shape Attributes for Binary Image

### Synopsis

Compute shape attributes for a binary image.

### Results

Output:

```
File "Generated image" has 2 labels.
Label: 1
BoundingBox: ImageRegion (0x7fab2f0003b8)
Dimension: 2
Index: [20, 30]
Size: [60, 70]

  NumberOfPixels: 4200
  PhysicalSize: 4200
  Centroid: [49.5, 64.5]
  NumberOfPixelsOnBorder: 0
  PerimeterOnBorder: 0
  FeretDiameter: 0
  PrincipalMoments: [299.917, 408.25]
  PrincipalAxes: 1 0
  0 1

  Elongation: 1.16671
  Perimeter: 245.385
  Roundness: 0.936229
  EquivalentSphericalRadius: 36.5637
  EquivalentSphericalPerimeter: 229.736
  EquivalentEllipsoidDiameter: [67.7015, 78.988]
  Flatness: 1.16671
```

(continues on next page)

(continued from previous page)

```
  PerimeterOnBorderRatio: 0
Label: 2
BoundingBox: ImageRegion (0x7fab2ef00308)
Dimension: 2
Index: [100, 115]
Size: [30, 45]

  NumberOfPixels: 1350
  PhysicalSize: 1350
  Centroid: [114.5, 137]
  NumberOfPixelsOnBorder: 0
  PerimeterOnBorder: 0
  FeretDiameter: 0
  PrincipalMoments: [74.9167, 168.667]
  PrincipalAxes: 1 0
  0 1

  Elongation: 1.50046
  Perimeter: 141.098
  Roundness: 0.923103
  EquivalentSphericalRadius: 20.7296
  EquivalentSphericalPerimeter: 130.248
  EquivalentEllipsoidDiameter: [33.8461, 50.7849]
  Flatness: 1.50046
  PerimeterOnBorderRatio: 0
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConnectedComponentImageFilter.h"
#include "itkLabelImageToShapeLabelMapFilter.h"

template <typename TImage>
static void
CreateImage(TImage * const image);

int
main(int argc, char * argv[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using LabelType = unsigned short;
  using InputImageType = itk::Image<PixelType, Dimension>;
  using OutputImageType = itk::Image<LabelType, Dimension>;
  using ShapeLabelObjectType = itk::ShapeLabelObject<LabelType, Dimension>;
  using LabelMapType = itk::LabelMap<ShapeLabelObjectType>;

  std::string             fileName;
  InputImageType::Pointer image;
  if (argc < 2)
  {
```

(continues on next page)

(continued from previous page)

```
    image = InputImageType::New();
    CreateImage(image.GetPointer());
    fileName = "Generated image";
  }
  else
  {
    fileName = argv[1];
    using ReaderType = itk::ImageFileReader<InputImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(fileName);
    reader->Update();

    image = reader->GetOutput();
  }


  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<InputImageType, OutputImageType>;
  using I2LType = itk::LabelImageToShapeLabelMapFilter<OutputImageType, LabelMapType>;

  ConnectedComponentImageFilterType::Pointer connected =␣
↪ConnectedComponentImageFilterType::New();
  connected->SetInput(image);
  connected->Update();

  using I2LType = itk::LabelImageToShapeLabelMapFilter<OutputImageType, LabelMapType>;
  I2LType::Pointer i2l = I2LType::New();
  i2l->SetInput(connected->GetOutput());
  i2l->SetComputePerimeter(true);
  i2l->Update();

  LabelMapType * labelMap = i2l->GetOutput();
  std::cout << "File "
            << "\"" << fileName << "\""
            << " has " << labelMap->GetNumberOfLabelObjects() << " labels." << std::
↪endl;

  // Retrieve all attributes
  for (unsigned int n = 0; n < labelMap->GetNumberOfLabelObjects(); ++n)
  {
    ShapeLabelObjectType * labelObject = labelMap->GetNthLabelObject(n);
    std::cout << "Label: " << itk::NumericTraits<LabelMapType::LabelType>::
↪PrintType(labelObject->GetLabel())
              << std::endl;
    std::cout << "    BoundingBox: " << labelObject->GetBoundingBox() << std::endl;
    std::cout << "    NumberOfPixels: " << labelObject->GetNumberOfPixels() << std::
↪endl;
    std::cout << "    PhysicalSize: " << labelObject->GetPhysicalSize() << std::endl;
    std::cout << "    Centroid: " << labelObject->GetCentroid() << std::endl;
    std::cout << "    NumberOfPixelsOnBorder: " << labelObject->
↪GetNumberOfPixelsOnBorder() << std::endl;
    std::cout << "    PerimeterOnBorder: " << labelObject->GetPerimeterOnBorder() <<␣
↪std::endl;
    std::cout << "    FeretDiameter: " << labelObject->GetFeretDiameter() << std::
↪endl;
    std::cout << "    PrincipalMoments: " << labelObject->GetPrincipalMoments() <<␣
↪std::endl;
```

(continues on next page)

(continued from previous page)

```
    std::cout << "    PrincipalAxes: " << labelObject->GetPrincipalAxes() << std::
↪endl;
    std::cout << "    Elongation: " << labelObject->GetElongation() << std::endl;
    std::cout << "    Perimeter: " << labelObject->GetPerimeter() << std::endl;
    std::cout << "    Roundness: " << labelObject->GetRoundness() << std::endl;
    std::cout << "    EquivalentSphericalRadius: " << labelObject->
↪GetEquivalentSphericalRadius() << std::endl;
    std::cout << "    EquivalentSphericalPerimeter: " << labelObject->
↪GetEquivalentSphericalPerimeter() << std::endl;
    std::cout << "    EquivalentEllipsoidDiameter: " << labelObject->
↪GetEquivalentEllipsoidDiameter() << std::endl;
    std::cout << "    Flatness: " << labelObject->GetFlatness() << std::endl;
    std::cout << "    PerimeterOnBorderRatio: " << labelObject->
↪GetPerimeterOnBorderRatio() << std::endl;
  }

  return EXIT_SUCCESS;
}

template <typename TImage>
void
CreateImage(TImage * const image)
{
  // Create an image with 2 objects
  typename TImage::IndexType start = { { 0, 0 } };
  start[0] = 0;
  start[1] = 0;

  typename TImage::SizeType size;
  unsigned int              NumRows = 200;
  unsigned int              NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;

  typename TImage::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (typename TImage::IndexValueType r = 20; r < 80; ++r)
  {
    for (typename TImage::IndexValueType c = 30; c < 100; ++c)
    {
      typename TImage::IndexType pixelIndex = { { r, c } };

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (typename TImage::IndexValueType r = 100; r < 130; ++r)
  {
    for (typename TImage::IndexValueType c = 115; c < 160; ++c)
    {
      typename TImage::IndexType pixelIndex = { { r, c } };
```

(continues on next page)

(continued from previous page)

```
      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *LabelMap*<ShapeLabelObject<**typename** *TInputImage*::PixelTy
**class LabelImageToShapeLabelMapFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Converts a label image to a label map and valuates the shape attributes.

A convenient class that converts a label image to a label map and valuates the shape attribute at once.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/176

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** ShapeLabelObject, LabelShapeOpeningImageFilter, LabelStatisticsOpeningImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Shape Attributes For Binary Image
- Convert Image With Labeled Regions To ShapeLabelMap

See itk::LabelImageToShapeLabelMapFilter for additional documentation.

### Note about this module:

This module contains the infrastructure to store the label images in an efficient and convenient way, including label images with overlapping objects. All the objects contained in the LabelMap can be associated to values called attributes. Some filters are available to compute the usual attributes values related to the shape of the objects or the pixels values in the objects and to manipulate the objects based on these values.

Implementations were taken from the Insight Journal paper.

## 3.4.21 MathematicalMorphology

### Create a Binary Ball Structuring Element

### Synopsis

Create an elliptical structuring element

### Results

### Code

### C++

```
#include "itkBinaryBallStructuringElement.h"

int
main(int, char *[])
{
  constexpr unsigned int Dimension = 3;
  using PixelType = float;

  using StructuringElementType = itk::BinaryBallStructuringElement<PixelType,␣
↪Dimension>;
  StructuringElementType structuringElement;
  structuringElement.SetRadius(5);
  structuringElement.CreateStructuringElement();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPixel**, unsigned int **VDimension** = 2, typename **TAllocator** = NeighborhoodAllocator<*TPixel*>>
**class BinaryBallStructuringElement** : **public** itk::Neighborhood<*TPixel*, *VDimension*, *TAllocator*>

A Neighborhood that represents a ball structuring element (ellipsoid) with binary elements.

This class defines a Neighborhood whose elements are either off or on depending on whether they are outside or inside an ellipsoid whose radii match the radii of the Neighborhood. This class can be used as a structuring element for the Morphology image filters.

A BinaryBallStructuringElement has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D BinaryBallStructuringElement object with a radius of 2x3 has sides of length 5x7.

BinaryBallStructuringElement objects always have an unambiguous center because their side lengths are always odd.

Internally, this class carries out all of its computations using the FlatStructuringElement. It is preferable to use that class instead of this one because FlatStructuringElement is more flexible.

**See** Neighborhood

**See** MorphologyImageFilter

**See** BinaryDilateImageFilter

**See** BinaryErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A Binary Ball Structuring Element

See itk::BinaryBallStructuringElement for additional documentation.

### Dilate a grayscale image

**See also:**

dilation; erosion

### Synopsis

Dilate regions by using a specified kernel, also known as a structuring element. In this example, the white regions are enlarged.

### Results

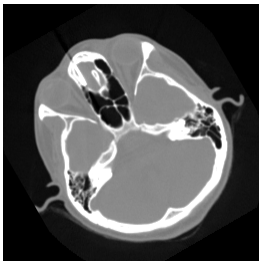


Fig. 298: Input grayscale image.

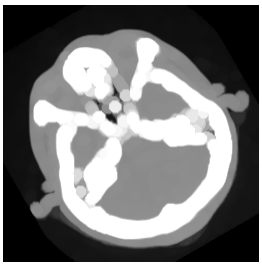


Fig. 299: Dilated output.

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

itk.auto_progress(2)
```

(continues on next page)

(continued from previous page)

```
parser = argparse.ArgumentParser(description="Dilate A Grayscale Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

StructuringElementType = itk.FlatStructuringElement[Dimension]
structuringElement = StructuringElementType.Ball(args.radius)

grayscaleFilter = itk.GrayscaleDilateImageFilter[
    ImageType, ImageType, StructuringElementType
].New()
grayscaleFilter.SetInput(reader.GetOutput())
grayscaleFilter.SetKernel(structuringElement)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(grayscaleFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkFlatStructuringElement.h"
#include "itkGrayscaleDilateImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <inputImage> <outputImage> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char *       inputImage = argv[1];
  const char *       outputImage = argv[2];
  const unsigned int radiusValue = std::stoi(argv[3]);

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);
  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using GrayscaleDilateImageFilterType = itk::GrayscaleDilateImageFilter<ImageType,␣
↪ImageType, StructuringElementType>;

  GrayscaleDilateImageFilterType::Pointer dilateFilter =␣
↪GrayscaleDilateImageFilterType::New();
  dilateFilter->SetInput(reader->GetOutput());
  dilateFilter->SetKernel(structuringElement);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(dilateFilter->GetOutput());
  writer->SetFileName(outputImage);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class GrayscaleDilateImageFilter** : **public** itk::KernelImageFilter<*TInputImage*, *TOutputImage*, *TKernel*>

Grayscale dilation of an image.

Dilate an image using grayscale morphology. Dilation takes the maximum of all the pixels identified by the structuring element.

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel.

**See** MorphologyImageFilter, GrayscaleFunctionDilateImageFilter, BinaryDilateImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Dilate A Grayscale Image

See itk::GrayscaleDilateImageFilter for additional documentation.

template<unsigned int **VDimension**>
**class FlatStructuringElement** : **public** itk::Neighborhood<bool, *VDimension*>

A class to support a variety of flat structuring elements, including versions created by decomposition of lines.

FlatStructuringElement provides several static methods, which can be used to create a structuring element with a particular shape, size, etc. Currently, those methods enable the creation of the following structuring elements: ball, box, cross, annulus, or polygon. Polygons are available as fast approximations of balls using line decompositions. Boxes also use line decompositions.

"Flat" refers to binary as opposed to grayscale structuring elements. Flat structuring elements can be used for both binary and grayscale images.

A Neighborhood has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D Neighborhood object with a radius of 2x3 has sides of length 5x7. However, in the case of balls and annuli, this definition is slightly different from the parametric definition of those objects. For example, an ellipse of radius 2x3 has a diameter of 4x6, not 5x7. To have a diameter of 5x7, the radius would need to increase by 0.5 in each dimension. Thus, the "radius" of the neighborhood and the "radius" of the object should be distinguished.

To accomplish this, the "ball" and "annulus" structuring elements have an optional flag called "radiusIsParametric" (off by default). Setting this flag to true will use the parametric definition of the object and will generate structuring elements with more accurate areas, which can be especially important when morphological operations are intended to remove or retain objects of particular sizes. When the mode is turned off (default), the radius is the same, but the object diameter is set to (radius*2)+1, which is the size of the neighborhood region. Thus, the original ball and annulus structuring elements have a systematic bias in the radius of +0.5 voxels in each dimension relative to the parametric definition of the radius. Thus, we recommend turning this mode on for more accurate structuring elements, but this mode is turned off by default for backward compatibility.

As an example, a 3D ball of radius 5 should have an area of 523. With this mode turned on, the number of "on" pixels is 515 (error 1.6%), but with it turned off, the area is 739 (error 41%). For a 3D annulus of radius 5 and thickness 2, the area should be 410. With this mode turned on, the area is 392 (error 4.5%), but when turned off it is 560 (error 36%). This same trend holds for balls and annuli of any radius or dimension. For more detailed experiments with this mode, please refer to the results of the test itkFlatStructuringElementTest.cxx or the wiki example.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Erode Binary Image Using Flat Structure Element

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Generate Structuring Elements With Accurate Area

See itk::FlatStructuringElement for additional documentation.

### Erode a Grayscale Image

**See also:**

erosion; dilation

### Synopsis

Erode regions by using a specified kernel, also known as a structuring element. In this example, the white regions shrink.

### Results

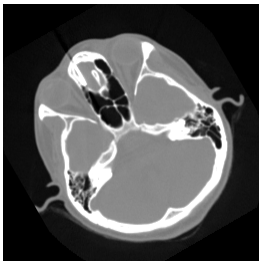


Fig. 300: Input grayscale image.

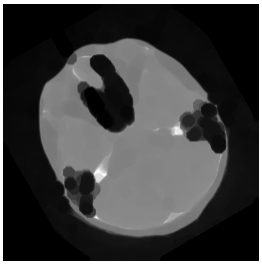


Fig. 301: Eroded output.

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

itk.auto_progress(2)

parser = argparse.ArgumentParser(description="Erode A Grayscale Image.")
parser.add_argument("input_image")
```

(continues on next page)

(continued from previous page)

```
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

StructuringElementType = itk.FlatStructuringElement[Dimension]
structuringElement = StructuringElementType.Ball(args.radius)

GrayscaleFilterType = itk.GrayscaleErodeImageFilter[
    ImageType, ImageType, StructuringElementType
]
grayscaleFilter = GrayscaleFilterType.New()
grayscaleFilter.SetInput(reader.GetOutput())
grayscaleFilter.SetKernel(structuringElement)

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(grayscaleFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGrayscaleErodeImageFilter.h"
#include "itkFlatStructuringElement.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <inputImage> <outputImage> <radius>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char *        inputImage = argv[1];
  const char *        outputImage = argv[2];
  const unsigned int radiusValue = std::stoi(argv[3]);

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImage);

  using StructuringElementType = itk::FlatStructuringElement<Dimension>;
  StructuringElementType::RadiusType radius;
  radius.Fill(radiusValue);
  StructuringElementType structuringElement = StructuringElementType::Ball(radius);

  using GrayscaleErodeImageFilterType = itk::GrayscaleErodeImageFilter<ImageType,␣
↪ImageType, StructuringElementType>;

  GrayscaleErodeImageFilterType::Pointer erodeFilter = GrayscaleErodeImageFilterType::
↪New();
  erodeFilter->SetInput(reader->GetOutput());
  erodeFilter->SetKernel(structuringElement);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(erodeFilter->GetOutput());
  writer->SetFileName(outputImage);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TKernel**>
**class GrayscaleErodeImageFilter** : **public** itk::KernelImageFilter<*TInputImage*, *TOutputImage*, *TKernel*>
Grayscale erosion of an image.

Erode an image using grayscale morphology. Erosion takes the maximum of all the pixels identified by the structuring element.

The structuring element is assumed to be composed of binary values (zero or one). Only elements of the structuring element having values > 0 are candidates for affecting the center pixel.

**See** MorphologyImageFilter, GrayscaleFunctionErodeImageFilter, BinaryErodeImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Erode A Grayscale Image

See itk::GrayscaleErodeImageFilter for additional documentation.

template<unsigned int **VDimension**>
**class FlatStructuringElement** : **public** itk::Neighborhood<bool, *VDimension*>

A class to support a variety of flat structuring elements, including versions created by decomposition of lines.

FlatStructuringElement provides several static methods, which can be used to create a structuring element with a particular shape, size, etc. Currently, those methods enable the creation of the following structuring elements: ball, box, cross, annulus, or polygon. Polygons are available as fast approximations of balls using line decompositions. Boxes also use line decompositions.

"Flat" refers to binary as opposed to grayscale structuring elements. Flat structuring elements can be used for both binary and grayscale images.

A Neighborhood has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D Neighborhood object with a radius of 2x3 has sides of length 5x7. However, in the case of balls and annuli, this definition is slightly different from the parametric definition of those objects. For example, an ellipse of radius 2x3 has a diameter of 4x6, not 5x7. To have a diameter of 5x7, the radius would need to increase by 0.5 in each dimension. Thus, the "radius" of the neighborhood and the "radius" of the object should be distinguished.

To accomplish this, the "ball" and "annulus" structuring elements have an optional flag called "radiusIsParametric" (off by default). Setting this flag to true will use the parametric definition of the object and will generate structuring elements with more accurate areas, which can be especially important when morphological operations are intended to remove or retain objects of particular sizes. When the mode is turned off (default), the radius is the same, but the object diameter is set to (radius*2)+1, which is the size of the neighborhood region. Thus, the original ball and annulus structuring elements have a systematic bias in the radius of +0.5 voxels in each dimension relative to the parametric definition of the radius. Thus, we recommend turning this mode on for more accurate structuring elements, but this mode is turned off by default for backward compatibility.

As an example, a 3D ball of radius 5 should have an area of 523. With this mode turned on, the number of "on" pixels is 515 (error 1.6%), but with it turned off, the area is 739 (error 41%). For a 3D annulus of radius 5 and thickness 2, the area should be 410. With this mode turned on, the area is 392 (error 4.5%), but when turned off it is 560 (error 36%). This same trend holds for balls and annuli of any radius or dimension. For more detailed experiments with this mode, please refer to the results of the test itkFlatStructuringElementTest.cxx or the wiki example.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Erode Binary Image Using Flat Structure Element

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Generate Structuring Elements With Accurate Area

See itk::FlatStructuringElement for additional documentation.

### Erode Binary Image Using Flat Structure Element

#### Synopsis

Erode a binary image using a flat (box) structuring element.

#### Results

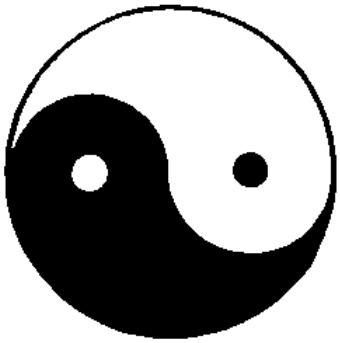


Fig. 302: Input image.

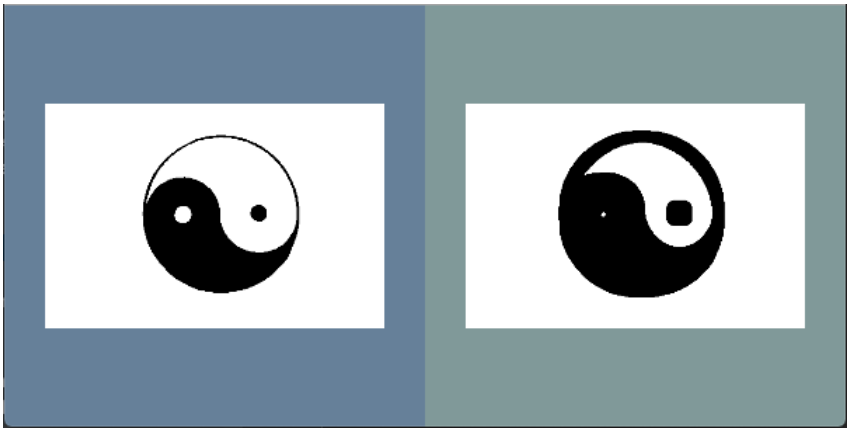


Fig. 303: Yinyang.png And Output.png When Radius = 7

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkFlatStructuringElement.h"
#include "itkBinaryErodeImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << argv[0] << " InputImageFile [radius]" << std::endl;
    return EXIT_FAILURE;
  }

  unsigned int radius = 2;
  if (argc > 2)
  {
    radius = std::stoi(argv[2]);
  }

  using ImageType = itk::Image<unsigned char, 2>;
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using StructuringElementType = itk::FlatStructuringElement<2>;
  StructuringElementType::RadiusType elementRadius;
  elementRadius.Fill(radius);

  StructuringElementType structuringElement = StructuringElementType::
↪Box(elementRadius);

  using BinaryErodeImageFilterType = itk::BinaryErodeImageFilter<ImageType, ImageType,
↪ StructuringElementType>;

  BinaryErodeImageFilterType::Pointer erodeFilter = BinaryErodeImageFilterType::New();
  erodeFilter->SetInput(reader->GetOutput());
  erodeFilter->SetKernel(structuringElement);
  erodeFilter->SetForegroundValue(255); // Intensity value to erode
  erodeFilter->SetBackgroundValue(0);   // Replacement value for eroded voxels

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput());
  viewer.AddImage(erodeFilter->GetOutput());
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VDimension**>
**class FlatStructuringElement** : **public** itk::Neighborhood<bool, *VDimension*>

A class to support a variety of flat structuring elements, including versions created by decomposition of lines.

FlatStructuringElement provides several static methods, which can be used to create a structuring element with a particular shape, size, etc. Currently, those methods enable the creation of the following structuring elements: ball, box, cross, annulus, or polygon. Polygons are available as fast approximations of balls using line decompositions. Boxes also use line decompositions.

"Flat" refers to binary as opposed to grayscale structuring elements. Flat structuring elements can be used for both binary and grayscale images.

A Neighborhood has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D Neighborhood object with a radius of 2x3 has sides of length 5x7. However, in the case of balls and annuli, this definition is slightly different from the parametric definition of those objects. For example, an ellipse of radius 2x3 has a diameter of 4x6, not 5x7. To have a diameter of 5x7, the radius would need to increase by 0.5 in each dimension. Thus, the "radius" of the neighborhood and the "radius" of the object should be distinguished.

To accomplish this, the "ball" and "annulus" structuring elements have an optional flag called "radiusIsParametric" (off by default). Setting this flag to true will use the parametric definition of the object and will generate structuring elements with more accurate areas, which can be especially important when morphological operations are intended to remove or retain objects of particular sizes. When the mode is turned off (default), the radius is the same, but the object diameter is set to (radius*2)+1, which is the size of the neighborhood region. Thus, the original ball and annulus structuring elements have a systematic bias in the radius of +0.5 voxels in each dimension relative to the parametric definition of the radius. Thus, we recommend turning this mode on for more accurate structuring elements, but this mode is turned off by default for backward compatibility.

As an example, a 3D ball of radius 5 should have an area of 523. With this mode turned on, the number of "on" pixels is 515 (error 1.6%), but with it turned off, the area is 739 (error 41%). For a 3D annulus of radius 5 and thickness 2, the area should be 410. With this mode turned on, the area is 392 (error 4.5%), but when turned off it is 560 (error 36%). This same trend holds for balls and annuli of any radius or dimension. For more detailed experiments with this mode, please refer to the results of the test itkFlatStructuringElementTest.cxx or the wiki example.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Erode Binary Image Using Flat Structure Element

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Generate Structuring Elements With Accurate Area

See itk::FlatStructuringElement for additional documentation.

### Generate Structuring Elements With Accurate Area

### Synopsis

Generate structuring elements with accurate area.

### Results

Output:

```
2D ball of radius 5 with radiusIsParametric mode off:
0       0       0       1       1       1       1       1       0       0       0
0       0       1       1       1       1       1       1       1       0       0
0       1       1       1       1       1       1       1       1       1       0
1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1
1       1       1       1   1   1       1       1       1       1       1
1       1       1       1       1       1       1       1       1       1       1
0       1       1       1       1       1       1       1       1       1       0
0       0       1       1   1   1       1       1       1       0       0
0       0       0       1       1       1       1       1       0       0       0
Expected foreground area: 78.5398
Computed foreground area: 97
Foreground area error: 23.5042%

2D ball of radius 5 with radiusIsParametric mode on:
0       0       0       0       0       1       0       0       0       0       0
0       0       1       1       1       1       1       1       1       0       0
0       1       1       1       1       1       1       1       1       1       0
0       1       1       1       1       1       1   1   1       1       0
0       1       1       1       1       1       1       1       1       1       0
1       1   1   1       1       1       1       1       1       1       1
0       1       1       1       1       1       1       1       1       1       0
0       1       1       1       1       1       1       1       1       1       0
0       1       1       1       1       1       1       1       1       1       0
0       0       1       1       1       1       1       1       1       0       0
0       0       0       0       0       1       0       0       0       0       0
Expected foreground area: 78.5398
Computed foreground area: 81
Foreground area error: 3.1324%

2D annulus of radius 5 and thickness 2 with radiusIsParametric mode off:
0       0       0       1       1       1       1       1       0       0       0
0       0       1       1       1       1       1       1       1       0       0
0   1   1       1   0   0       0       1       1       1       0
1       1       1       0       0       0       0       0       1       1       1
1       1       0       0       0       0       0       0       0       1       1
1       1       0       0       0       0       0       0       0       1       1
1       1       0       0   0   0       0       0       0       1       1
1       1       1       0       0       0       0       0       1       1       1
0       1       1       1       0       0       0       1       1       1       0
0       0       1       1       1       1       1       1       1       0       0
0   0   0       1       1       1       1       1       0       0       0
Expected foreground area: 50.2655
Computed foreground area: 60
```

(continues on next page)

(continued from previous page)

```
Foreground area error: 19.3662%

2D annulus of radius 5 and thickness 2 with radiusIsParametric mode on:
0   0   0       0       0       1       0       0       0       0       0
0       0       1       1       1   1   1       1       1       0       0
0       1       1       1       1       0       1       1       1       1       0
0       1       1   0   0       0       0       0       1   1   0
0       1       1       0       0       0       0       0       1       1       0
1       1       0       0       0       0       0       0   0   1       1
0       1       1       0       0       0       0       0       1       1       0
0       1       1       0       0       0       0       0       1       1       0
0       1       1       1       1       0       1       1       1       1       0
0       0       1       1       1       1       1       1       1       0       0
0       0       0       0       0       1       0       0       0       0       0
Expected foreground area: 50.2655
Computed foreground area: 52
Foreground area error: 3.45071%

3D ball of radius 5 with radiusIsParametric mode off:
Expected foreground area: 523.599
Computed foreground area: 739
Foreground area error: 41.1386%

3D ball of radius 5 with radiusIsParametric mode on:
Expected foreground area: 523.599
Computed foreground area: 515
Foreground area error: 1.64224%

3D annulus of radius 5 and thickness 2 with radiusIsParametric mode off:
Expected foreground area: 410.501
Computed foreground area: 560
Foreground area error: 36.4185%

3D annulus of radius 5 and thickness 2 with radiusIsParametric mode on:
Expected foreground area: 410.501
Computed foreground area: 392
Foreground area error: 4.50703%

4D ball of radius 5 with radiusIsParametric mode off:
Expected foreground area: 3084.25
Computed foreground area: 4785
Foreground area error: 55.143%

4D ball of radius 5 with radiusIsParametric mode on:
Expected foreground area: 3084.25
Computed foreground area: 2929
Foreground area error: 5.03368%

4D annulus of radius 5 and thickness 2 with radiusIsParametric mode off:
Expected foreground area: 2684.53
Computed foreground area: 4024
Foreground area error: 49.8957%

4D annulus of radius 5 and thickness 2 with radiusIsParametric mode on:
Expected foreground area: 2684.53
Computed foreground area: 2504
Foreground area error: 6.72491%
```

### Code

### C++

```
#include "itkFlatStructuringElement.h"

// Helper function
template <class SEType>
bool
ComputeAreaError(SEType k, unsigned int thickness = 0);

int
main(int, char *[])
{
  int  scalarRadius = 5;
  int  scalarThickness = 2;
  bool radiusIsParametric = true;

  using SE2Type = itk::FlatStructuringElement<2>;
  SE2Type::RadiusType r2;
  r2.Fill(scalarRadius);
  SE2Type k2;

  std::cout << "2D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪off:" << std::endl;
  k2 = SE2Type::Ball(r2);
  ComputeAreaError(k2);

  // Test the radiusIsParametric mode.
  std::cout << "2D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪on:" << std::endl;
  k2 = SE2Type::Ball(r2, radiusIsParametric);
  ComputeAreaError(k2);

  std::cout << "2D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode off:" << std::endl;
  k2 = SE2Type::Annulus(r2, scalarThickness, false);
  ComputeAreaError(k2, scalarThickness);

  // Test the radiusIsParametric mode.
  std::cout << "2D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode on:" << std::endl;
  k2 = SE2Type::Annulus(r2, scalarThickness, false, radiusIsParametric);
  ComputeAreaError(k2, scalarThickness);

  using SE3Type = itk::FlatStructuringElement<3>;
  SE3Type::RadiusType r3;
  r3.Fill(scalarRadius);
  SE3Type k3;

  std::cout << "3D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪off:" << std::endl;
  k3 = SE3Type::Ball(r3);
  ComputeAreaError(k3);
```

(continues on next page)

(continued from previous page)

```
  // Test the radiusIsParametric mode.
  std::cout << "3D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪on:" << std::endl;
  k3 = SE3Type::Ball(r3, radiusIsParametric);
  ComputeAreaError(k3);

  std::cout << "3D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode off:" << std::endl;
  k3 = SE3Type::Annulus(r3, scalarThickness, false);
  ComputeAreaError(k3, scalarThickness);

  // Test the radiusIsParametric mode.
  std::cout << "3D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode on:" << std::endl;
  k3 = SE3Type::Annulus(r3, scalarThickness, false, radiusIsParametric);
  ComputeAreaError(k3, scalarThickness);

  using SE4Type = itk::FlatStructuringElement<4>;
  SE4Type::RadiusType r4;
  r4.Fill(scalarRadius);
  SE4Type k4;

  std::cout << "4D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪off:" << std::endl;
  k4 = SE4Type::Ball(r4);
  ComputeAreaError(k4);

  // Test the radiusIsParametric mode.
  std::cout << "4D ball of radius " << scalarRadius << " with radiusIsParametric mode␣
↪on:" << std::endl;
  k4 = SE4Type::Ball(r4, radiusIsParametric);
  ComputeAreaError(k4);

  std::cout << "4D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode off:" << std::endl;
  k4 = SE4Type::Annulus(r4, scalarThickness, false);
  ComputeAreaError(k4, scalarThickness);

  // Test the radiusIsParametric mode.
  std::cout << "4D annulus of radius " << scalarRadius << " and thickness " <<␣
↪scalarThickness
            << " with radiusIsParametric mode on:" << std::endl;
  k4 = SE4Type::Annulus(r4, scalarThickness, false, radiusIsParametric);
  ComputeAreaError(k4, scalarThickness);

  return EXIT_SUCCESS;
}

template <class SEType>
bool
ComputeAreaError(SEType k, unsigned int thickness)
{
  float expectedOuterForegroundArea = 1;
  float expectedInnerForegroundArea;
```

(continues on next page)

(continued from previous page)

```
  if (thickness == 0)
  {
    // Circle/Ellipse has no inner area to subract.
    expectedInnerForegroundArea = 0;
  }
  else
  {
    // Annulus does have inner area to subract.
    expectedInnerForegroundArea = 1;
  }
  if (SEType::NeighborhoodDimension == 2)
  {
    expectedOuterForegroundArea *= itk::Math::pi;
    expectedInnerForegroundArea *= itk::Math::pi;
  }
  else if (SEType::NeighborhoodDimension == 3)
  {
    expectedOuterForegroundArea *= 4.0 / 3.0 * itk::Math::pi;
    expectedInnerForegroundArea *= 4.0 / 3.0 * itk::Math::pi;
  }
  else if (SEType::NeighborhoodDimension == 4)
  {
    expectedOuterForegroundArea *= 0.5 * itk::Math::pi * itk::Math::pi;
    expectedInnerForegroundArea *= 0.5 * itk::Math::pi * itk::Math::pi;
  }
  else
  {
    return EXIT_FAILURE;
  }
  for (unsigned int i = 0; i < SEType::NeighborhoodDimension; i++)
  {
    expectedOuterForegroundArea *= k.GetRadius()[i];
    expectedInnerForegroundArea *= (k.GetRadius()[i] - thickness);
  }

  float expectedForegroundArea = expectedOuterForegroundArea -␣
↪expectedInnerForegroundArea;

  // Show the neighborhood if it is 2D.
  typename SEType::Iterator SEIt;
  if (SEType::NeighborhoodDimension == 2)
  {
    for (SEIt = k.Begin(); SEIt != k.End(); ++SEIt)
    {
      std::cout << *SEIt << "\t";
      if ((SEIt - k.Begin() + 1) % k.GetSize()[0] == 0)
      {
        std::cout << std::endl;
      }
    }
  }

  // Compute the area/volume.
  float computedForegroundArea = 0;
  for (SEIt = k.Begin(); SEIt != k.End(); ++SEIt)
  {
    if (*SEIt)
```

(continues on next page)

(continued from previous page)

```
    {
      computedForegroundArea++;
    }
  }

  std::cout << "Expected foreground area: " << expectedForegroundArea << std::endl;
  std::cout << "Computed foreground area: " << computedForegroundArea << std::endl;
  std::cout << "Foreground area error: "
            << 100 * std::abs(expectedForegroundArea - computedForegroundArea) /␣
↪expectedForegroundArea << "%"
            << "\n\n";

  return EXIT_FAILURE;
}
```

## Classes demonstrated

template<unsigned int **VDimension**>
**class FlatStructuringElement** : **public** itk::Neighborhood<bool, *VDimension*>

A class to support a variety of flat structuring elements, including versions created by decomposition of lines.

FlatStructuringElement provides several static methods, which can be used to create a structuring element with a particular shape, size, etc. Currently, those methods enable the creation of the following structuring elements: ball, box, cross, annulus, or polygon. Polygons are available as fast approximations of balls using line decompositions. Boxes also use line decompositions.

"Flat" refers to binary as opposed to grayscale structuring elements. Flat structuring elements can be used for both binary and grayscale images.

A Neighborhood has an N-dimensional *radius*. The radius is defined separately for each dimension as the number of pixels that the neighborhood extends outward from the center pixel. For example, a 2D Neighborhood object with a radius of 2x3 has sides of length 5x7. However, in the case of balls and annuli, this definition is slightly different from the parametric definition of those objects. For example, an ellipse of radius 2x3 has a diameter of 4x6, not 5x7. To have a diameter of 5x7, the radius would need to increase by 0.5 in each dimension. Thus, the "radius" of the neighborhood and the "radius" of the object should be distinguished.

To accomplish this, the "ball" and "annulus" structuring elements have an optional flag called "radiusIsParametric" (off by default). Setting this flag to true will use the parametric definition of the object and will generate structuring elements with more accurate areas, which can be especially important when morphological operations are intended to remove or retain objects of particular sizes. When the mode is turned off (default), the radius is the same, but the object diameter is set to (radius*2)+1, which is the size of the neighborhood region. Thus, the original ball and annulus structuring elements have a systematic bias in the radius of +0.5 voxels in each dimension relative to the parametric definition of the radius. Thus, we recommend turning this mode on for more accurate structuring elements, but this mode is turned off by default for backward compatibility.

As an example, a 3D ball of radius 5 should have an area of 523. With this mode turned on, the number of "on" pixels is 515 (error 1.6%), but with it turned off, the area is 739 (error 41%). For a 3D annulus of radius 5 and thickness 2, the area should be 410. With this mode turned on, the area is 392 (error 4.5%), but when turned off it is 560 (error 36%). This same trend holds for balls and annuli of any radius or dimension. For more detailed experiments with this mode, please refer to the results of the test itkFlatStructuringElementTest.cxx or the wiki example.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Erode Binary Image Using Flat Structure Element

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Generate Structuring Elements With Accurate Area

See itk::FlatStructuringElement for additional documentation.

### Regional Maximal

### Synopsis

Regional maximal image filter.

### Results

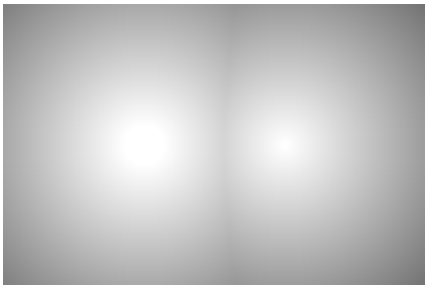


Fig. 304: intensityblobs.png

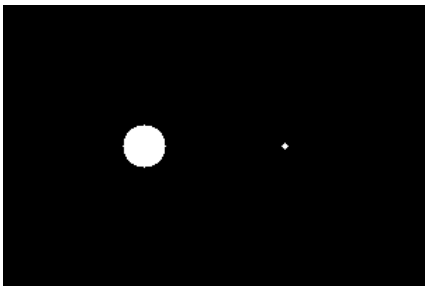


Fig. 305: maximal.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRegionalMaximaImageFilter.h"


using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using RegionalMaximaImageFilter = itk::RegionalMaximaImageFilter<ImageType,␣
↪ImageType>;

  RegionalMaximaImageFilter::Pointer filter = RegionalMaximaImageFilter::New();
  filter->SetInput(image);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName("intensityblobs.png");
  writer->SetInput(image);
  writer->Update();

  writer->SetFileName("maximal.png");
  writer->SetInput(filter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumCols;
  size[1] = NumRows;

  region.SetSize(size);
  region.SetIndex(start);
```

(continues on next page)

(continued from previous page)

```
  image->SetRegions(region);
  image->Allocate();

  // Make two intensity blobs
  for (unsigned int r = 0; r < NumRows; r++)
  {
    for (unsigned int c = 0; c < NumCols; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = c;
      pixelIndex[1] = r;

      double c1 = c - 100.0;
      double c2 = c - 200.0;

      double rr = r - 100.0;

      // purposely use 270,257 since it is > 255
      double v1 = 270.0 - std::sqrt(rr * rr + c1 * c1);
      double v2 = 257.0 - std::sqrt(rr * rr + c2 * c2);

      double maxv = v1;
      if (maxv < v2)
        maxv = v2;

      double val = maxv;

      if (val < 0.0)
        val = 0.0;
      if (val > 255.0)
        val = 255.0;

      image->SetPixel(pixelIndex, val);
    }
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RegionalMaximaImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Produce a binary image where foreground is the regional maxima of the input image.

Regional maxima are flat zones surrounded by pixels of lower value.

If the input image is constant, the entire image can be considered as a maxima or not. The desired behavior can be selected with the SetFlatIsMaxima() method.

This class was contributed to the Insight Journal by author Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France. The paper can be found at https://www.insight-journal.org/browse/publication/65

**Author** Gaetan Lehmann

**See** ValuedRegionalMaximaImageFilter

**See** HConvexImageFilter

**See** RegionalMinimaImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Regional Maximal

See itk::RegionalMaximaImageFilter for additional documentation.

### Regional Minimal

### Synopsis

Regional minimal image filter.

### Results

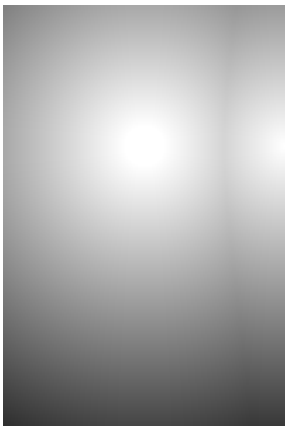


Fig. 306: input.png

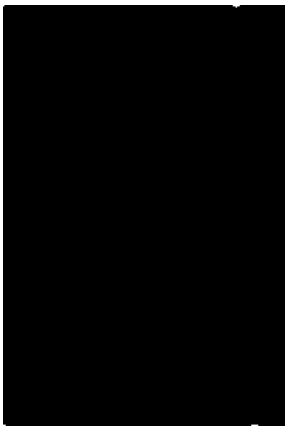


Fig. 307: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRegionalMinimaImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using RegionalMinimaImageFilterType = itk::RegionalMinimaImageFilter<ImageType,␣
↪ImageType>;
  RegionalMinimaImageFilterType::Pointer regionalMinimaImageFilter =␣
↪RegionalMinimaImageFilterType::New();
  regionalMinimaImageFilter->SetInput(image);
  regionalMinimaImageFilter->Update();

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();

  writer->SetFileName("output.png");
  writer->SetInput(regionalMinimaImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  ImageType::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();

  // Make two intensity blobs
  for (unsigned int r = 0; r < size[1]; r++)
  {
```

(continues on next page)

(continued from previous page)

```
    for (unsigned int c = 0; c < size[0]; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = c;
      pixelIndex[1] = r;

      double c1 = c - 100.0;
      double c2 = c - 200.0;

      double rr = r - 100.0;

      // purposely use 270,257 since it is > 255
      double v1 = 270.0 - std::sqrt(rr * rr + c1 * c1);
      double v2 = 257.0 - std::sqrt(rr * rr + c2 * c2);

      double maxv = v1;
      if (maxv < v2)
        maxv = v2;

      double val = maxv;

      // Clip
      if (val < 0.0)
        val = 0.0;
      if (val > 255.0)
        val = 255.0;

      image->SetPixel(pixelIndex, static_cast<unsigned char>(val));
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RegionalMinimaImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Produce a binary image where foreground is the regional minima of the input image.

Regional minima are flat zones surrounded by pixels of greater value.

If the input image is constant, the entire image can be considered as a minima or not. The SetFlatIsMinima() method let the user choose which behavior to use.

This class was contributed to the Insight Journal by

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France. https://www.insight-journal.org/browse/publication/65

**See** RegionalMaximaImageFilter

**See** ValuedRegionalMinimaImageFilter

**See** HConcaveImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Regional Minimal

See itk::RegionalMinimaImageFilter for additional documentation.

### Valued Regional Maxima Image

### Synopsis

Valued regional maximal of an image.

### Results

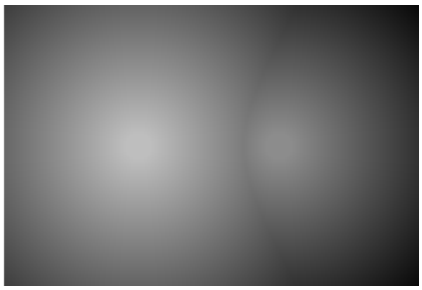


Fig. 308: intensityblobs.png

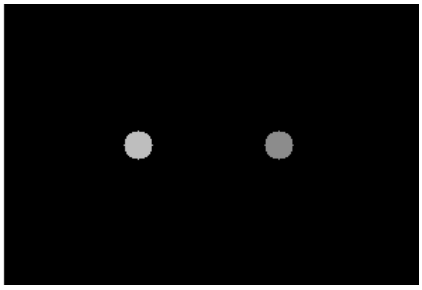


Fig. 309: maximal.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkValuedRegionalMaximaImageFilter.h"


using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);
```

(continues on next page)

(continued from previous page)

```
int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using ValuedRegionalMaximaImageFilter = itk::ValuedRegionalMaximaImageFilter
↪<ImageType, ImageType>;

  ValuedRegionalMaximaImageFilter::Pointer filter = ValuedRegionalMaximaImageFilter::
↪New();
  filter->SetInput(image);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName("intensityblobs.png");
  writer->SetInput(image);
  writer->Update();

  writer->SetFileName("maximal.png");
  writer->SetInput(filter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumCols;
  size[1] = NumRows;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make two intensity blobs
  for (unsigned int r = 0; r < NumRows; r++)
  {
    for (unsigned int c = 0; c < NumCols; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = c;
      pixelIndex[1] = r;
```

(continues on next page)

(continued from previous page)

```
      double c1 = c - 100.0;
      double c2 = c - 200.0;

      double rr = r - 100.0;

      double v1 = 200.0 - std::sqrt(rr * rr + c1 * c1);
      double v2 = 150.0 - std::sqrt(rr * rr + c2 * c2);

      if (v1 > 190.0)
        v1 = 190.0;
      if (v2 > 140.0)
        v2 = 140.0;

      double maxv = v1;
      if (maxv < v2)
        maxv = v2;

      double val = maxv;

      if (val < 0.0)
        val = 0.0;
      if (val > 255.0)
        val = 255.0;

      if (c < 5)
        val = 255.0;

      image->SetPixel(pixelIndex, val);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ValuedRegionalMaximaImageFilter** : **public** itk::ValuedRegionalExtremaImageFilter<*TInputImage*, *TOutputImag*

Transforms the image so that any pixel that is not a regional maxima is set to the minimum value for the pixel type. Pixels that are regional maxima retain their value.

Regional maxima are flat zones surrounded by pixels of lower value. A completely flat image will be marked as a regional maxima by this filter.

This code was contributed in the Insight Journal paper: "Finding regional extrema - methods and performance" by Beare R., Lehmann G. https://www.insight-journal.org/browse/publication/65

**Author** Richard Beare. Department of Medicine, Monash University, Melbourne, Australia.

**See** ValuedRegionalMinimaImageFilter

**See** ValuedRegionalExtremaImageFilter

**See** HMinimaImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Valued Regional Maxima Image

See itk::ValuedRegionalMaximaImageFilter for additional documentation.

### Valued Regional Minimal Image

### Synopsis

Valued regional minimal of an image.

### Results

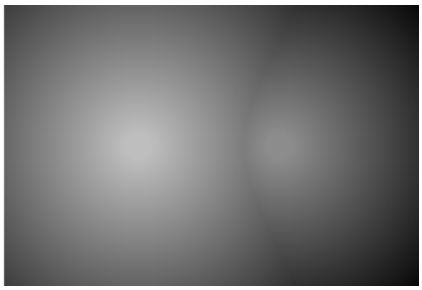


Fig. 310: input.png

Fig. 311: output.png

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkValuedRegionalMinimaImageFilter.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);
```

(continues on next page)

(continued from previous page)

```
int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using ValuedRegionalMinimaImageFilter = itk::ValuedRegionalMinimaImageFilter
↪<ImageType, ImageType>;

  ValuedRegionalMinimaImageFilter::Pointer filter = ValuedRegionalMinimaImageFilter::
↪New();
  filter->SetInput(image);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();

  writer->SetFileName("output.png");
  writer->SetInput(filter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  unsigned int        NumRows = 200;
  unsigned int        NumCols = 300;
  size[0] = NumCols;
  size[1] = NumRows;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make two intensity blobs
  for (unsigned int r = 0; r < NumRows; r++)
  {
    for (unsigned int c = 0; c < NumCols; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = c;
      pixelIndex[1] = r;
```

(continues on next page)

(continued from previous page)

```
      double c1 = c - 100.0;
      double c2 = c - 200.0;

      double rr = r - 100.0;

      double v1 = 200.0 - std::sqrt(rr * rr + c1 * c1);
      double v2 = 150.0 - std::sqrt(rr * rr + c2 * c2);

      if (v1 > 190.0)
        v1 = 190.0;
      if (v2 > 140.0)
        v2 = 140.0;

      double maxv = v1;
      if (maxv < v2)
        maxv = v2;

      double val = maxv;

      if (val < 0.0)
        val = 0.0;
      if (val > 255.0)
        val = 255.0;

      if (c < 5)
        val = 255.0;

      image->SetPixel(pixelIndex, static_cast<unsigned char>(val));
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ValuedRegionalMinimaImageFilter** : **public** itk::ValuedRegionalExtremaImageFilter<*TInputImage*, *TOutputImag*

Transforms the image so that any pixel that is not a regional minima is set to the maximum value for the pixel type. Pixels that are regional minima retain their value.

Regional minima are flat zones surrounded by pixels of higher value. A completely flat image will be marked as a regional minima by this filter.

This code was contributed in the Insight Journal paper: "Finding regional extrema - methods and performance" by Beare R., Lehmann G. https://www.insight-journal.org/browse/publication/65

**Author** Richard Beare. Department of Medicine, Monash University, Melbourne, Australia.

**See** ValuedRegionalMaximaImageFilter, ValuedRegionalExtremaImageFilter,

**See** HMinimaImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Valued Regional Minimal Image

See itk::ValuedRegionalMinimaImageFilter for additional documentation.

## 3.4.22 Path

### Data Structure for Piece-Wise Linear Curve

### Synopsis

A data structure for a piece-wise linear curve.

### Results

Output:

```
[0, 0]
[0, 1]
[0, 2]
[0, 3]
[0, 4]
[0, 5]
[0, 6]
[0, 7]
[0, 8]
[0, 9]
[0, 10]
[0, 11]
[0, 12]
[0, 13]
[0, 14]
[0, 15]
[0, 16]
[0, 17]
[0, 18]
[0, 19]
```

### Code

### C++

```
#include "itkPolyLineParametricPath.h"

int
main(int, char *[])
{
  using PathType = itk::PolyLineParametricPath<2>;

  PathType::Pointer path = PathType::New();
  path->Initialize();

  using ContinuousIndexType = PathType::ContinuousIndexType;

  // Create a line
```

(continues on next page)

(continued from previous page)

```
  for (unsigned int i = 0; i < 20; ++i)
  {
    ContinuousIndexType cindex;
    cindex[0] = 0;
    cindex[1] = i;
    path->AddVertex(cindex);
  }

  const PathType::VertexListType * vertexList = path->GetVertexList();

  for (unsigned int i = 0; i < vertexList->Size(); ++i)
  {
    std::cout << vertexList->GetElement(i) << std::endl;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<unsigned int **VDimension**>
**class PolyLineParametricPath** : **public** itk::ParametricPath<*VDimension*>

Represent a path of line segments through ND Space.

This class is intended to represent parametric paths through an image, where the paths are composed of line segments. Each line segment traverses one unit of input. A classic application of this class is the representation of contours in 2D images, especially when the contours only need to be approximately correct. Another use of a path is to guide the movement of an iterator through an image.

**See** EllipseParametricPath

**See** FourierSeriesPath

**See** OrthogonallyCorrectedParametricPath

**See** ParametricPath

**See** ChainCodePath

**See** Path

**See** ContinuousIndex

**See** Index

**See** Offset

**See** Vector

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Data Structure For Piece-Wise Linear Curve

See itk::PolyLineParametricPath for additional documentation.

### Extract Contours From Image

### Synopsis

Extract contours from an image.

### Results

Output:

```
There are 2 contours

Contour 0:
[10.5, 19]
[10.4603, 18]
[10.4603, 17]
[10.4603, 16]
[10.4603, 15]
[10.4603, 14]
[10.4603, 13]
[10.4603, 12]
[10.4603, 11]
[10.5, 10]
[10, 9.58579]
[9.5, 10]
[9.5397, 11]
[9.5397, 12]
[9.5397, 13]
[9.5397, 14]
[9.5397, 15]
[9.5397, 16]
[9.5397, 17]
[9.5397, 18]
[9.5, 19]
[10, 19.4142]
[10.5, 19]

Contour 1:
[59.4142, 59]
[59, 58.5]
[58.5, 58]
[58, 57.5]
[57.5, 57]
[57, 56.5]
[56.5, 56]
[56, 55.5]
[55.5, 55]
[55, 54.5]
[54.5, 54]
[54, 53.5]
[53.5, 53]
[53, 52.5]
[52.5, 52]
[52, 51.5]
[51.5, 51]
[51, 50.5]
```

(continues on next page)

(continued from previous page)

```
[50.5, 50]
[50, 49.5]
[49.5, 49]
[49, 48.5]
[48.5, 48]
[48, 47.5]
[47.5, 47]
[47, 46.5]
[46.5, 46]
[46, 45.5]
[45.5, 45]
[45, 44.5]
[44.5, 44]
[44, 43.5]
[43.5, 43]
[43, 42.5]
[42.5, 42]
[42, 41.5]
[41.5, 41]
[41, 40.5]
[40.5, 40]
[40, 39.5858]
[39.5858, 40]
[40, 40.5]
[40.5, 41]
[41, 41.5]
[41.5, 42]
[42, 42.5]
[42.5, 43]
[43, 43.5]
[43.5, 44]
[44, 44.5]
[44.5, 45]
[45, 45.5]
[45.5, 46]
[46, 46.5]
[46.5, 47]
[47, 47.5]
[47.5, 48]
[48, 48.5]
[48.5, 49]
[49, 49.5]
[49.5, 50]
[50, 50.5]
[50.5, 51]
[51, 51.5]
[51.5, 52]
[52, 52.5]
[52.5, 53]
[53, 53.5]
[53.5, 54]
[54, 54.5]
[54.5, 55]
[55, 55.5]
[55.5, 56]
[56, 56.5]
[56.5, 57]
```

(continues on next page)

(continued from previous page)

```
[57, 57.5]
[57.5, 58]
[58, 58.5]
[58.5, 59]
[59, 59.4142]
[59.4142, 59]
```

## Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkContourExtractor2DImageFilter.h"
#include "itkApproximateSignedDistanceMapImageFilter.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  using ApproximateSignedDistanceMapImageFilterType =
    itk::ApproximateSignedDistanceMapImageFilter<UnsignedCharImageType,␣
↪FloatImageType>;
  ApproximateSignedDistanceMapImageFilterType::Pointer␣
↪approximateSignedDistanceMapImageFilter =
    ApproximateSignedDistanceMapImageFilterType::New();
  approximateSignedDistanceMapImageFilter->SetInput(image);
  approximateSignedDistanceMapImageFilter->SetInsideValue(255);
  approximateSignedDistanceMapImageFilter->SetOutsideValue(0);
  approximateSignedDistanceMapImageFilter->Update();

  using ContourExtractor2DImageFilterType = itk::ContourExtractor2DImageFilter
↪<FloatImageType>;
  ContourExtractor2DImageFilterType::Pointer contourExtractor2DImageFilter =␣
↪ContourExtractor2DImageFilterType::New();
  contourExtractor2DImageFilter->SetInput(approximateSignedDistanceMapImageFilter->
↪GetOutput());
  contourExtractor2DImageFilter->SetContourValue(0);
  contourExtractor2DImageFilter->Update();

  std::cout << "There are " << contourExtractor2DImageFilter->GetNumberOfOutputs() <<
↪" contours" << std::endl;


  for (unsigned int i = 0; i < contourExtractor2DImageFilter->GetNumberOfOutputs();␣
↪++i)
```

(continues on next page)

(continued from previous page)

```
  {
    std::cout << "Contour " << i << ": " << std::endl;
    ContourExtractor2DImageFilterType::VertexListType::ConstIterator vertexIterator =
      contourExtractor2DImageFilter->GetOutput(i)->GetVertexList()->Begin();
    while (vertexIterator != contourExtractor2DImageFilter->GetOutput(i)->
↪GetVertexList()->End())
    {
      std::cout << vertexIterator->Value() << std::endl;
      ++vertexIterator;
    }
    std::cout << std::endl;
  }

  return EXIT_SUCCESS;
}

void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create an image
  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(100);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Create a line of white pixels
  for (unsigned int i = 40; i < 60; ++i)
  {
    itk::Index<2> pixel;
    pixel.Fill(i);
    image->SetPixel(pixel, 255);
  }

  // Create another line of pixels
  for (unsigned int i = 10; i < 20; ++i)
  {
    itk::Index<2> pixel;
    pixel[0] = 10;
    pixel[1] = i;
    image->SetPixel(pixel, 255);
  }

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("image.png");
  writer->SetInput(image);
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ContourExtractor2DImageFilter** : **public** itk::ImageToPathFilter<*TInputImage*, *PolyLineParametricPath*<2>>
Computes a list of PolyLineParametricPath objects from the contours in a 2D image.

Uses the "marching squares" method to compute a the iso-valued contours of the input 2D image for a given intensity value. Multiple outputs may be produced because an image can have multiple contours at a given level, so it is advised to call GetNumberOfIndexedOutputs() and GetOutput(n) to retrieve all of the contours. The contour value to be extracted can be set with SetContourValue(). Image intensities will be linearly interpolated to provide sub-pixel resolution for the output contours.

The marching squares algorithm is a special case of the marching cubes algorithm (Lorensen, William and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics (SIGGRAPH 87 Proceedings) 21(4) July 1987, p. 163-170). A simple explanation is available here: http://users.polytech.unice.fr/~lingrand/MarchingCubes/algo.html

There is an ambiguous case in the marching squares algorithm: if a given 2x2-pixel square has two high-valued and two low-valued pixels, each pair diagonally adjacent. (Note that high-valued and low-valued are with respect to the contour value sought when LabelContours is false. When LabelContours is true, high-valued means the label being traced and low-valued means any other label.) In this case, the default behavior is that the low-valued pixels are connected into the same contour via an isthmus that separates the high-valued pixels into separate contours. To reverse this, call VertexConnectHighPixelsOn(). Note that when LabelContours is true, the default behavior will leave all four pixels in separate contours. In this case, VertexConnectHighPixels equal to true can instead create contours that are crossing barbells.

Outputs are not guaranteed to be closed paths: contours which intersect the image edge will be left open. All other paths will be closed. (The closedness of a path can be tested by checking whether the beginning point is the same as the end point.)

Produced paths are oriented. Following the path from beginning to end, image intensity values lower than the contour value are to the left of the path and intensity values greater than the contour value are to the right. In other words, the image gradient at a path segment is (approximately) in the direct of that segment rotated right by 90 degrees, because the image intensity values increase from left-to-right across the segment. This means that the generated contours will circle clockwise around "hills" of above-contour-value intensity, and counter-clockwise around "depressions" of below-contour-value intensity. This convention can be reversed by calling ReverseContourOrientationOn().

By default values are interpreted as intensities relative to a contour value. First calling LabelContoursOn() changes this behavior to instead interpret each value as a label. Boundaries are computed for each label separately and all are returned. The value of LabelContours affects the interpretation of VertexConnectHighPixels; see above.

By default the input image's largest possible region will be processed; call SetRequestedRegion() to process a different region, or ClearRequestedRegion() to revert to the default value. Note that the requested regions are usually set on the output; however since paths have no notion of a "region", this must be set at the filter level.

This class was contributed to the Insight Journal by Zachary Pincus. https://www.insight-journal.org/browse/publication/72

**See** Image

**See** Path

**See** PolyLineParametricPath

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Extract Contours From Image

See itk::ContourExtractor2DImageFilter for additional documentation.

### 3.4.23 QuadEdgeMeshFiltering

#### Clean Quad Edge Mesh

#### Synopsis

Clean quad edge mesh

#### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message.

#### Code

#### C++

```
#include "itkQuadEdgeMesh.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"


#include "itkCleanQuadEdgeMeshFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cout << "Requires 3 argument: " << std::endl;
    std::cout << "1-Input file name " << std::endl;
    std::cout << "2-Relative Tolerance " << std::endl;
    std::cout << "3-Output file name " << std::endl;
    return EXIT_FAILURE;
  }

  using Coord = double;
  constexpr unsigned int Dimension = 3;

  using MeshType = itk::QuadEdgeMesh<Coord, Dimension>;
  using ReaderType = itk::MeshFileReader<MeshType>;
  using WriterType = itk::MeshFileWriter<MeshType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  MeshType::Pointer mesh = reader->GetOutput();
```

(continues on next page)

(continued from previous page)

```
  Coord             tol;
  std::stringstream ssout(argv[2]);
  ssout >> tol;

  using CleanFilterType = itk::CleanQuadEdgeMeshFilter<MeshType, MeshType>;
  CleanFilterType::Pointer filter = CleanFilterType::New();
  filter->SetInput(mesh);
  filter->SetRelativeTolerance(tol);

  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(filter->GetOutput());
  writer->SetFileName(argv[3]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  std::cout << filter;
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputMesh** = *TInputMesh*>
**class CleanQuadEdgeMeshFilter** : **public** itk::QuadEdgeMeshToQuadEdgeMeshFilter<*TInputMesh*, *TOutputMesh*>
TODO.

See itk::CleanQuadEdgeMeshFilter for additional documentation.

### Compute Normals of a Mesh

### Synopsis

Compute normals of a mesh

### Results

Output:

```
Index * Point * Normal
0 * [0.461602, -0.018446, -0.115989] * [0.964656, 0.0898966, -0.247704]
1 * [0.202784, 0.359481, -0.217858] * [0.398458, 0.53089, -0.747922]
2 * [-0.481417, -0.046787, 0.167623] * [-0.987132, 0.0342443, 0.1562]
3 * [-0.028388, -0.193602, -0.329748] * [-0.73488, -0.637902, -0.230287]
4 * [-0.087584, 0.116513, -0.399884] * [-0.396325, 0.718368, -0.571729]
```

(continues on next page)

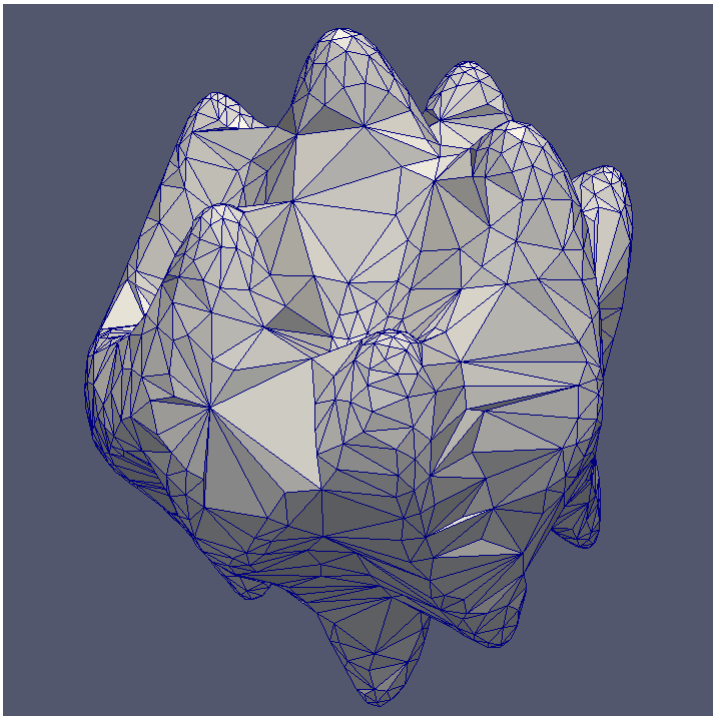


Fig. 312: Input mesh

(continued from previous page)

```
5 * [0.310978, -0.239208, -0.176146] * [-0.256754, 0.55162, -0.793595]
6 * [-0.303314, 0.043035, -0.515252] * [-0.26465, 0.684736, -0.679041]
7 * [-0.256717, -0.39897, 0.177429] * [0.0907732, -0.987892, 0.125819]
8 * [0.393919, -0.388691, -0.207638] * [0.290631, -0.955747, -0.0456223]
9 * [-0.225987, -0.368634, -0.367823] * [0.154093, 0.365541, -0.917952]
10 * [-0.142556, -0.187471, -0.613449] * [0.902758, -0.0825592, -0.422153]
11 * [0.405923, -0.12839, -0.433595] * [0.432278, -0.901416, 0.0241831]
12 * [0.107094, 0.05873, -0.53405] * [0.452665, 0.664453, -0.594639]
13 * [0.179737, -0.358912, 0.264171] * [0.622873, -0.779647, 0.0646554]
14 * [0.270291, 0.167529, -0.267412] * [0.282452, 0.630941, -0.722588]
15 * [0.190508, -0.139941, -0.30917] * [0.642572, -0.758423, 0.109063]
16 * [-0.405901, 0.088723, -0.347001] * [-0.736406, 0.557068, -0.383904]
17 * [0.150462, -0.314033, -0.695425] * [-0.171163, -0.38232, -0.908039]
18 * [-0.134383, -0.384493, 0.060186] * [-0.0256952, -0.994633, 0.100226]
19 * [-0.333505, -0.116041, -0.567548] * [-0.847824, -0.186998, -0.496212]
20 * [-0.133643, -0.267196, -0.238155] * [0.123997, 0.550013, -0.8259]
21 * [-0.274406, 0.185852, -0.366349] * [-0.0630695, 0.682696, -0.727976]
22 * [-0.34352, -0.372556, 0.17102] * [-0.565292, -0.779407, 0.270129]
23 * [-0.468522, 0.005617, -0.307273] * [-0.945988, 0.0866878, -0.312397]
24 * [0.012837, 0.243987, -0.417788] * [-0.288839, 0.714998, -0.636671]
25 * [0.004712, 0.118755, -0.486604] * [-0.451072, 0.542366, -0.708783]

[...]

1300 * [0.429046, 0.119465, 0.002968] * [0.838127, 0.413114, -0.356201]
1301 * [-0.346539, -0.163042, 0.316913] * [-0.323662, 0.716134, 0.618381]
1302 * [-0.111388, 0.551166, -0.050984] * [-0.558618, 0.686741, -0.465115]
1303 * [0.376285, 0.211351, 0.347429] * [0.754427, -0.0746173, 0.652129]
```

(continues on next page)

(continued from previous page)

```
1304 * [-0.026005, 0.583073, 0.081228] * [0.60426, 0.298334, 0.738828]
1305 * [-0.196168, -0.079437, 0.372088] * [-0.859283, 0.463078, 0.217237]
1306 * [-0.000569, 0.431162, 0.415514] * [0.683671, 0.70426, 0.191341]
```

### Code

### C++

```
#include "itkVector.h"
#include "itkQuadEdgeMesh.h"
#include "itkMeshFileReader.h"
#include "itkQuadEdgeMeshExtendedTraits.h"
#include "itkNormalQuadEdgeMeshFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << "<InputFileName> <WeightType>" << std::endl;
    std::cerr << "Weight type: " << std::endl;
    std::cerr << "  * 0:  GOURAUD" << std::endl;
    std::cerr << "  * 1:  THURMER" << std::endl;
    std::cerr << "  * 2:  AREA" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using InputMeshType = itk::QuadEdgeMesh<CoordType, Dimension>;

  using VectorType = itk::Vector<CoordType, Dimension>;

  using Traits =
    itk::QuadEdgeMeshExtendedTraits<VectorType, Dimension, 2, CoordType, CoordType,␣
↪VectorType, bool, bool>;

  using ReaderType = itk::MeshFileReader<InputMeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  using OutputMeshType = itk::QuadEdgeMesh<VectorType, Dimension, Traits>;

  using NormalFilterType = itk::NormalQuadEdgeMeshFilter<InputMeshType,␣
↪OutputMeshType>;
  NormalFilterType::WeightEnum weight_type;

  int param = std::stoi(argv[2]);

  if ((param < 0) || (param > 2))
  {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "Weight type must be either: " << std::endl;
    std::cerr << "   * 0:  GOURAUD" << std::endl;
    std::cerr << "   * 1:  THURMER" << std::endl;
    std::cerr << "   * 2:  AREA" << std::endl;
    return EXIT_FAILURE;
  }
  else
  {
    switch (param)
    {
      default:
      case 0:
        weight_type = itk::NormalQuadEdgeMeshFilterEnums::Weight::GOURAUD;
        break;
      case 1:
        weight_type = itk::NormalQuadEdgeMeshFilterEnums::Weight::THURMER;
        break;
      case 2:
        weight_type = itk::NormalQuadEdgeMeshFilterEnums::Weight::AREA;
        break;
    }
  }

  NormalFilterType::Pointer normals = NormalFilterType::New();
  normals->SetInput(reader->GetOutput());
  normals->SetWeight(weight_type);
  normals->Update();

  OutputMeshType::Pointer output = normals->GetOutput();

  OutputMeshType::PointsContainerPointer  points = output->GetPoints();
  OutputMeshType::PointsContainerIterator p_it = points->Begin();

  OutputMeshType::PointDataContainerPointer  container = output->GetPointData();
  OutputMeshType::PointDataContainerIterator d_it = container->Begin();

  std::cout << "Index * Point * Normal" << std::endl;

  while (p_it != points->End())
  {
    std::cout << p_it.Index() << " * ";
    std::cout << p_it.Value() << " * ";
    std::cout << d_it.Value() << std::endl;

    ++p_it;
    ++d_it;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputMesh**>
**class NormalQuadEdgeMeshFilter** : **public** itk::QuadEdgeMeshToQuadEdgeMeshFilter<*TInputMesh*, *TOutputMesh*>
Filter which computes normals to faces and vertices and store it in the output mesh. Normals to face are first computed, then normals to vertices are computed as linear combination of neighbor face normals, i.e.

$$n_v = \frac{\sum_{i=0}^{N_f} \omega_i \cdot n_i}{\| \sum_{k=0}^{N_f} \omega_x \cdot n_k \|}$$

The difference between each method relies in the definition of the weight $\omega_i$ that you can specify by the method SetWeight.

- GOURAUD $\omega_i = 1$ [1]
- THURMER $\omega_i =$ Angle of the considered triangle at the given vertex} [2]
- AREA $\omega_i = Area(t_i)$ [3]

These weights are defined in the literature:

- [1] Henri Gouraud. Continuous shading of curved surfaces. IEEE Transaction on Computers, 20(6):623-629, 1971
- [2] Shuangshuang Jin, Robert R. Lewis, and David West. A comparison of algorithms for vertex normal computation. The Visual Computer, 21(1-2):71-82, 2005.
- [3] Grit Thurmer and Charles A. Wuthrich. Computing vertex normals from polygonal facets. Journal of Graphic Tools, 3(1):43-46, 1998.

**Note** By default the weight is set to the TURMER weight.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Normals Of A Mesh

See itk::NormalQuadEdgeMeshFilter for additional documentation.

### Compute Planar Parameterization of a Mesh

### Synopsis

Compute planar parameterization of a surface mesh represented by one itk::QuadEdgeMesh.

### Results

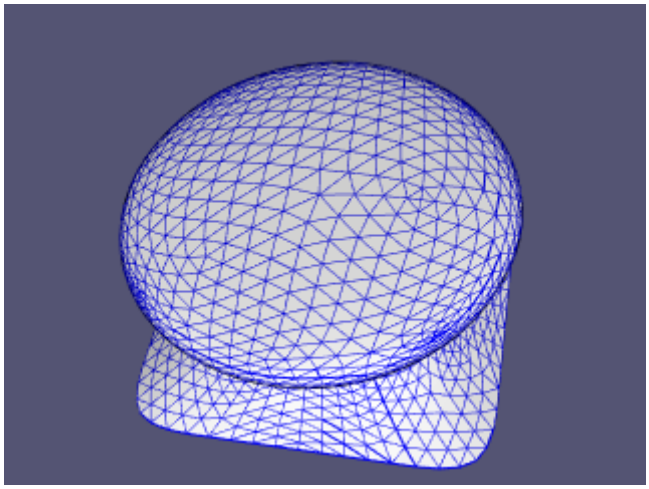


Fig. 313: Input mesh

### Code

### C++

```
#include "itkQuadEdgeMesh.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"
#include "VNLIterativeSparseSolverTraits.h"
#include "itkParameterizationQuadEdgeMeshFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cout << "Requires 4 arguments: " << std::endl;
    std::cout << "1-Input file name " << std::endl;
    std::cout << "2-Border Type" << std::endl;
    std::cout << "    * 0: SQUARE" << std::endl;
    std::cout << "    * 1: DISK" << std::endl;
    std::cout << "3-CoefficientType Type" << std::endl;
    std::cout << "    * 0: OnesMatrixCoefficients" << std::endl;
    std::cout << "    * 1: InverseEuclideanDistanceMatrixCoefficients" << std::endl;
    std::cout << "    * 2: ConformalMatrixCoefficients" << std::endl;
    std::cout << "    * 3: AuthalicMatrixCoefficients" << std::endl;
    std::cout << "    * 4: HarmonicMatrixCoefficients" << std::endl;
    std::cout << "4-Output file name " << std::endl;

    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[4];
```

(continues on next page)

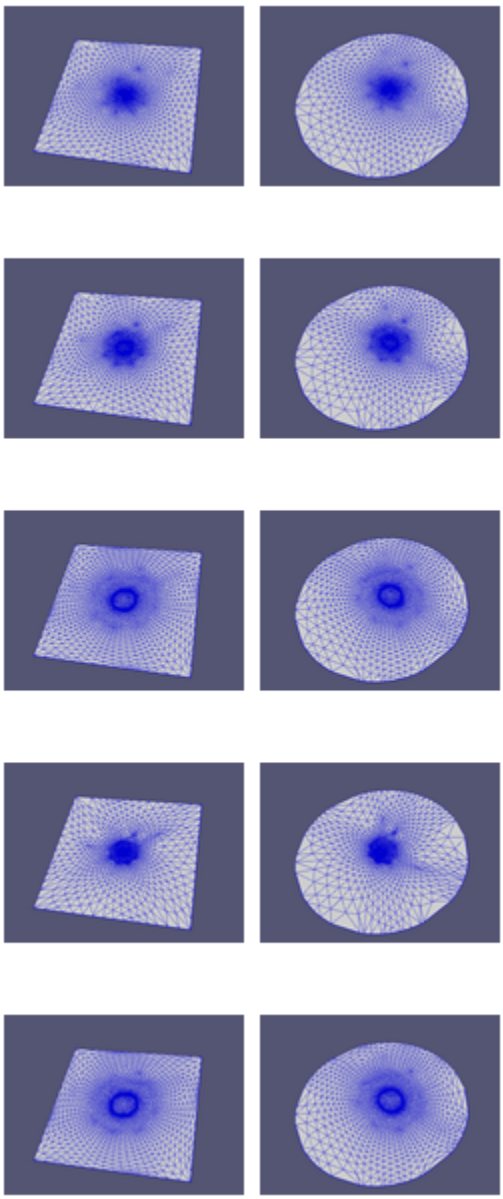


Fig. 314: Output planes with left column) SQUARE, and right column) DISK BorderType, and row 1) OnesMatrix, 2) InverseEuclideanMatrix 3) ConformalMatrix, 4) AuthalicMatrix, and 5) HarmonicMatrix CoefficientType.

(continued from previous page)

```
  using CoordType = double;
  constexpr unsigned int Dimension = 3;

  using MeshType = itk::QuadEdgeMesh<CoordType, Dimension>;
  using ReaderType = itk::MeshFileReader<MeshType>;
  using WriterType = itk::MeshFileWriter<MeshType>;
  using BorderTransformType = itk::BorderQuadEdgeMeshFilter<MeshType, MeshType>;
  using SolverTraits = VNLIterativeSparseSolverTraits<CoordType>;

  using ParametrizationType = itk::ParameterizationQuadEdgeMeshFilter<MeshType,␣
↪MeshType, SolverTraits>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  MeshType::Pointer mesh = reader->GetOutput();

  BorderTransformType::Pointer border_transform = BorderTransformType::New();
  border_transform->SetInput(reader->GetOutput());

  int               border;
  std::stringstream ssout(argv[2]);
  ssout >> border;
  switch (border) // choose border type
  {
    case 0: // square shaped domain
      border_transform->SetTransformType(itk::BorderQuadEdgeMeshFilterEnums::
↪BorderTransform::SQUARE_BORDER_TRANSFORM);
      break;
    case 1: // disk shaped domain
      border_transform->SetTransformType(itk::BorderQuadEdgeMeshFilterEnums::
↪BorderTransform::DISK_BORDER_TRANSFORM);
      break;
    default: // handle .... user ....
      std::cerr << "2nd argument must be " << std::endl;
      std::cerr << "0 for SQUARE BORDER TRANSFORM or "
                << "1 for DISK BORDER TRANSFORM" << std::endl;
      return EXIT_FAILURE;
  }

  std::cout << "Transform type is: " << border_transform->GetTransformType() << std::
↪endl;

  // ** CHOOSE AND SET BARYCENTRIC WEIGHTS **
  itk::OnesMatrixCoefficients<MeshType>                      coeff0;
  itk::InverseEuclideanDistanceMatrixCoefficients<MeshType> coeff1;
  itk::ConformalMatrixCoefficients<MeshType>                 coeff2;
  itk::AuthalicMatrixCoefficients<MeshType>                  coeff3;
  itk::HarmonicMatrixCoefficients<MeshType>                  coeff4;

  ParametrizationType::Pointer param = ParametrizationType::New();
  param->SetInput(mesh);
  param->SetBorderTransform(border_transform);

  int               param_type;
  std::stringstream ssout3(argv[3]);
```

(continues on next page)

(continued from previous page)

```
  ssout3 >> param_type;

  switch (param_type)
  {
    case 0:
      param->SetCoefficientsMethod(&coeff0);
      break;
    case 1:
      param->SetCoefficientsMethod(&coeff1);
      break;
    case 2:
      param->SetCoefficientsMethod(&coeff2);
      break;
    case 3:
      param->SetCoefficientsMethod(&coeff3);
      break;
    case 4:
      param->SetCoefficientsMethod(&coeff4);
      break;
    default:
      std::cerr << "3rd argument must be " << std::endl;
      std::cerr << "0, 1, 2, 3 or 4" << std::endl;
      std::cerr << "Here it is: " << param_type << std::endl;
      return EXIT_FAILURE;
  }

  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(param->GetOutput());
  writer->SetFileName(outputFileName);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputMesh**, typename **TSolverTraits**>
**class ParameterizationQuadEdgeMeshFilter** : **public** itk::QuadEdgeMeshToQuadEdgeMeshFilter<*TInputMesh*, *TOu*

Compute a planar parameterization of the input mesh.

This filter computes a mapping in between a planar parametric domain and one input mesh.

This filter is made for fixed boundary parameterization where the parametric domain shape is given by the means of the border transform. Then, the position of internal vertices (not on the boundary) is directly connected to m_CoefficientsComputation.

This filter internally creates and solves a sparse linear system: storage and computation can be set by the

means of TSolverTraits. Since for 3D meshes, this filter solves for similar sparse linear systems for the three dimensions, it is highly recommended to use one direct solver which would first decompose sparse matrix (e.g. VNLSparseLUSolverTraits).

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/202

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Planar Parameterization Of A Mesh

See itk::ParameterizationQuadEdgeMeshFilter for additional documentation.

### Delaunay Conform Edge Flipping

### Synopsis

Flip the edges of one itk::QuadEdgeMesh to satisfy Delaunay condition.

### Results

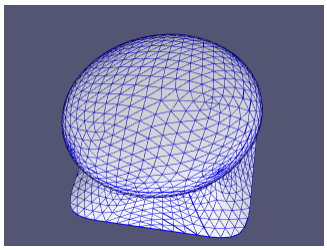


Fig. 315: Input mesh

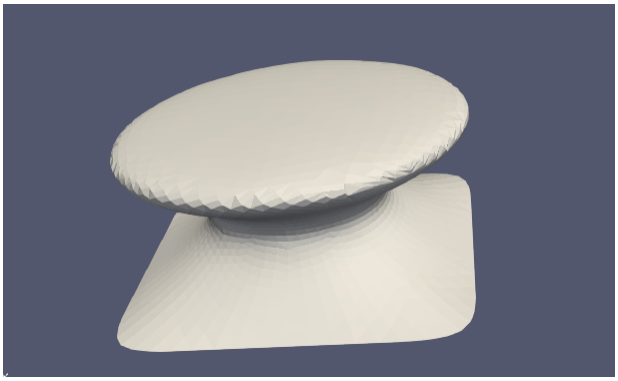


Fig. 316: Output mesh

### Code

### C++

```
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"
#include "itkQuadEdgeMesh.h"

#include "itkDelaunayConformingQuadEdgeMeshFilter.h"

int
main(int argc, char * argv[])
{
  // Error message and help.
  if (argc != 3)
  {
    std::cerr << "Usage:" << std::endl;
    std::cerr << argv[0] << " <InputFileName> <OutputFileName>" << std::endl;
    return EXIT_FAILURE;
  }

  // Basic types.
  constexpr unsigned int Dimension = 3;
  using CoordType = double;

  using MeshType = itk::QuadEdgeMesh<CoordType, Dimension>;

  // Read the file in.
  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  // Process the mesh.
  MeshType::Pointer mesh = reader->GetOutput();
  using DelaunayConformFilterType = itk::DelaunayConformingQuadEdgeMeshFilter
↪<MeshType, MeshType>;
  DelaunayConformFilterType::Pointer filter = DelaunayConformFilterType::New();
  filter->SetInput(mesh);

  // Write the output.
  using WriterType = itk::MeshFileWriter<MeshType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(filter->GetOutput());
  writer->SetFileName(argv[2]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  std::cout << "Number of Edge flipped performed: " << filter->GetNumberOfEdgeFlips()
↪<< std::endl;
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputMesh**, typename **TOutputMesh** = *TInputMesh*>
**class DelaunayConformingQuadEdgeMeshFilter** : **public** itk::QuadEdgeMeshToQuadEdgeMeshFilter<*TInputMesh*, *T*
FIXME Add documentation.

See itk::DelaunayConformingQuadEdgeMeshFilter for additional documentation.

### 3.4.24 Smoothing

#### Blurring an Image Using a Binomial Kernel

#### Synopsis

The BinomialBlurImageFilter computes a nearest neighbor average along each dimension. The process is repeated a number of times, as specified by the user. In principle, after a large number of iterations the result will approach the convolution with a Gaussian.

#### Results

#### Code

#### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Blurring An Image Using A Binomial Kernel."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_repetitions", type=int)
args = parser.parse_args()

InputPixelType = itk.F
OutputPixelType = itk.UC
Dimension = 2

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

reader = itk.ImageFileReader[InputImageType].New()
reader.SetFileName(args.input_image)
```

(continues on next page)

(continued from previous page)

```
binomialFilter = itk.BinomialBlurImageFilter.New(reader)
binomialFilter.SetRepetitions(args.number_of_repetitions)

rescaler = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType].New()
rescaler.SetInput(binomialFilter.GetOutput())
rescaler.SetOutputMinimum(0)
rescaler.SetOutputMaximum(255)

writer = itk.ImageFileWriter[OutputImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkBinomialBlurImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "  inputImageFile  outputImageFile  numberOfRepetitions" <
↪< std::endl;
    return EXIT_FAILURE;
  }

  using InputPixelType = float;
  using OutputPixelType = float;
  using InputImageType = itk::Image<InputPixelType, 2>;
  using OutputImageType = itk::Image<OutputPixelType, 2>;


  using FilterType = itk::BinomialBlurImageFilter<InputImageType, OutputImageType>;
  FilterType::Pointer filter = FilterType::New();

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();

  reader->SetFileName(argv[1]);
  const unsigned int repetitions = std::stoi(argv[3]);
  filter->SetInput(reader->GetOutput());
  filter->SetRepetitions(repetitions);
  filter->Update();

  using WritePixelType = unsigned char;
  using WriteImageType = itk::Image<WritePixelType, 2>;
```

(continues on next page)

(continued from previous page)

```
  using RescaleFilterType = itk::RescaleIntensityImageFilter<OutputImageType,␣
↪WriteImageType>;

  RescaleFilterType::Pointer rescaler = RescaleFilterType::New();
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(255);

  using WriterType = itk::ImageFileWriter<WriteImageType>;
  WriterType::Pointer writer = WriterType::New();

  writer->SetFileName(argv[2]);
  rescaler->SetInput(filter->GetOutput());
  writer->SetInput(rescaler->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinomialBlurImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Performs a separable blur on each dimension of an image.

The binomial blur consists of a nearest neighbor average along each image dimension. The net result after n-iterations approaches convolution with a gaussian.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Blurring An Image Using A Binomial Kernel

See itk::BinomialBlurImageFilter for additional documentation.

### Computes Smoothing With Gaussian Kernel

### Synopsis

Computes the smoothing of an image by convolution with Gaussian kernels.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
```

(continues on next page)

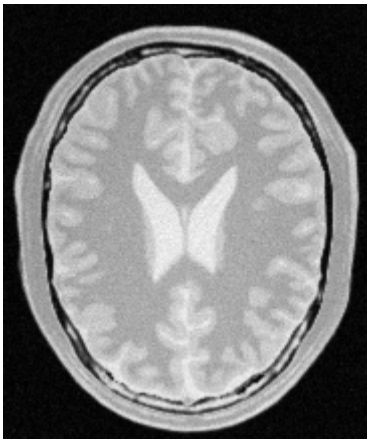


Fig. 317: Input image

Fig. 318: Output image

(continued from previous page)

```python
import argparse

parser = argparse.ArgumentParser(description="Computes Smoothing With Gaussian Kernel.
↪")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("sigma", type=float)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

smoothFilter = itk.SmoothingRecursiveGaussianImageFilter[ImageType, ImageType].New()
smoothFilter.SetInput(reader.GetOutput())
smoothFilter.SetSigma(args.sigma)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(smoothFilter.GetOutput())

writer.Update()
```

### C++

```cpp
#include "itkSmoothingRecursiveGaussianImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputImageFile> <OutputImageFile> <sigma>" << std::
↪endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const float  sigmaValue = std::stod(argv[3]);

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;
```

(continues on next page)

(continued from previous page)

```
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::SmoothingRecursiveGaussianImageFilter<ImageType, ImageType>;
  FilterType::Pointer smoothFilter = FilterType::New();

  smoothFilter->SetSigma(sigmaValue);
  smoothFilter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(smoothFilter->GetOutput());
  writer->SetFileName(outputFileName);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class SmoothingRecursiveGaussianImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>
Computes the smoothing of an image by convolution with the Gaussian kernels implemented as IIR filters.

This filter is implemented using the recursive gaussian filters. For multi-component images, the filter works on each component independently.

For this filter to be able to run in-place the input and output image types need to be the same and/or the same type as the RealImageType.

See itk::SmoothingRecursiveGaussianImageFilter for additional documentation.

### Find Higher Derivatives of Image

### Synopsis

Find higher derivatives of an image.

### Results

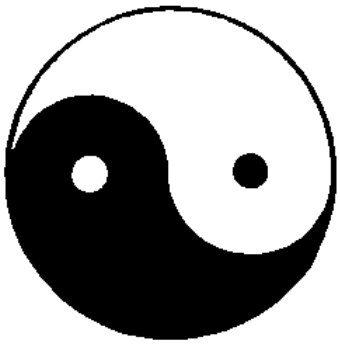


Fig. 319: Input image.

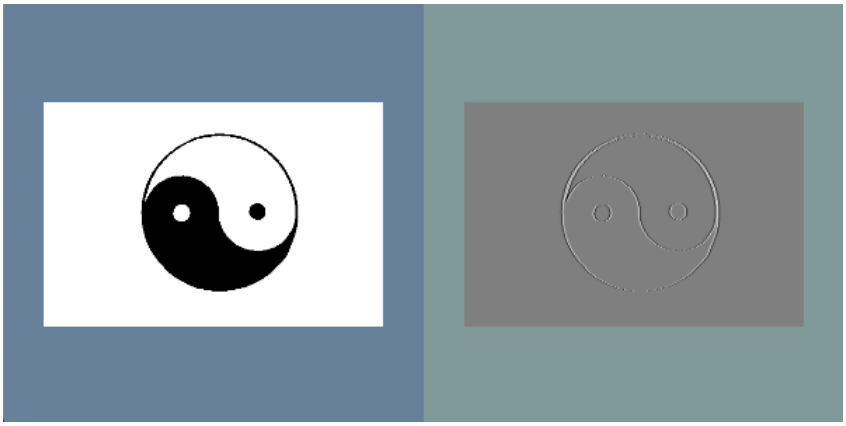


Fig. 320: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkRecursiveGaussianImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
```

(continues on next page)

(continued from previous page)

```
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << "inputImageFile" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line arguments
  std::string inputFilename = argv[1];

  // Setup types
  using FloatImageType = itk::Image<float, 2>;
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;

  using readerType = itk::ImageFileReader<UnsignedCharImageType>;

  using filterType = itk::RecursiveGaussianImageFilter<UnsignedCharImageType,␣
↪FloatImageType>;

  // Create and setup a reader
  readerType::Pointer reader = readerType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a gaussian filter
  filterType::Pointer gaussianFilter = filterType::New();
  gaussianFilter->SetInput(reader->GetOutput());
  gaussianFilter->SetDirection(0); // "x" axis
  gaussianFilter->SetSecondOrder();

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage<UnsignedCharImageType>(reader->GetOutput());
  viewer.AddImage<FloatImageType>(gaussianFilter->GetOutput());
  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class RecursiveGaussianImageFilter** : **public** itk::RecursiveSeparableImageFilter<*TInputImage*, *TOutputImage*>
Base class for computing IIR convolution with an approximation of a Gaussian kernel.

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

RecursiveGaussianImageFilter is the base class for recursive filters that approximate convolution with the Gaussian kernel. This class implements the recursive filtering method proposed by R.Deriche in IEEE-PAMI Vol.12, No.1, January 1990, pp 78-87, "Fast Algorithms for Low-Level Vision"

Details of the implementation are described in the technical report: R. Deriche, "Recursively Implementing The Gaussian and Its Derivatives", INRIA, 1993, ftp://ftp.inria.fr/INRIA/tech-reports/RR/RR-1893.ps.gz

Further improvements of the algorithm are described in: G. Farnebäck & C.-F. Westin, "Improving Deriche-style Recursive Gaussian

Filters". J Math Imaging Vis 26, 293–299 (2006). https://doi.org/10.1007/s10851-006-8464-z

As compared to itk::DiscreteGaussianImageFilter, this filter tends to be faster for large kernels, and it can take the derivative of the blurred image in one step. Also, note that we have itk::RecursiveGaussianImageFilter::SetSigma(), but itk::DiscreteGaussianImageFilter::SetVariance().

**See** DiscreteGaussianImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Find Higher Derivatives Of Image

See itk::RecursiveGaussianImageFilter for additional documentation.

### Mean Filtering of an Image

### Synopsis

Apply mean filtering on an image.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Mean Filtering Of An Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

meanFilter = itk.MeanImageFilter[ImageType, ImageType].New()
```

(continues on next page)

Fig. 321: Input image

Fig. 322: Output image

(continued from previous page)

```
meanFilter.SetInput(reader.GetOutput())
meanFilter.SetRadius(args.radius)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(meanFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkMeanImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputImageFile> <OutputImageFile> <radius>" << std::
↪endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const int    radiusValue = std::stoi(argv[3]);

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::MeanImageFilter<ImageType, ImageType>;
  FilterType::Pointer meanFilter = FilterType::New();

  FilterType::InputSizeType radius;
  radius.Fill(radiusValue);

  meanFilter->SetRadius(radius);
  meanFilter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(meanFilter->GetOutput());
  writer->SetFileName(outputFileName);
```

(continues on next page)

(continued from previous page)

```
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MeanImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>
Applies an averaging filter to an image.

Computes an image where a given pixel is the mean value of the the pixels in a neighborhood about the corresponding input pixel.

A mean filter is one of the family of linear filters.

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mean filter an image

Subclassed by itk::GPUImageToImageFilter< TInputImage, TOutputImage, MeanImageFilter< TInputImage, TOutputImage > >

See itk::MeanImageFilter for additional documentation.

### Median Filtering of an Image

### Synopsis

Apply median filtering on an image.

### Results

Fig. 323: Input image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Median Filtering Of An Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("radius", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2
```

(continues on next page)

Fig. 324: Output image

(continued from previous page)

```
ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

medianFilter = itk.MedianImageFilter[ImageType, ImageType].New()
medianFilter.SetInput(reader.GetOutput())
medianFilter.SetRadius(args.radius)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(medianFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkMedianImageFilter.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputImageFile> <OutputImageFile> <radius>" << std::
↪endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const int    radiusValue = std::stoi(argv[3]);

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::MedianImageFilter<ImageType, ImageType>;
  FilterType::Pointer medianFilter = FilterType::New();

  FilterType::InputSizeType radius;
  radius.Fill(radiusValue);

  medianFilter->SetRadius(radius);
```

(continues on next page)

(continued from previous page)

```
  medianFilter->SetInput(reader->GetOutput());

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(medianFilter->GetOutput());
  writer->SetFileName(outputFileName);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MedianImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>

Applies a median filter to an image.

Computes an image where a given pixel is the median value of the the pixels in a neighborhood about the corresponding input pixel.

A median filter is one of the family of nonlinear filters. It is used to smooth an image without being biased by outliers or shot noise.

This filter requires that the input pixel type provides an operator<() (LessThan Comparable).

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Median Filter Of An Image
- Median Filter Of An RGB Image

See itk::MedianImageFilter for additional documentation.

### Median Filtering of an RGB Image

### Synopsis

Apply median filtering on a RGB Image. The sorting of pixel is performed on their luminance.

### Results

Fig. 325: Input image

### Code

### C++

```
#include "itkMedianImageFilter.h"

#include "itkRGBPixel.h"
#include "itkImage.h"
#include "itkImageFileReader.h"
```

(continues on next page)

Fig. 326: Output image

(continued from previous page)

```
#include "itkImageFileWriter.h"

namespace itk
{
/** \class myRGBPixel
 * \brief Extends RGBPixel with operator <=
 *
 * This class overrides the <= and < operators to use Luminance as a sorting
 * value.
 */
template <typename TComponent = unsigned short>
class myRGBPixel : public RGBPixel<TComponent>
{
public:
  using Self = myRGBPixel;
  using Superclass = RGBPixel<TComponent>;

  using RGBPixel<TComponent>::operator=;

  bool
  operator<=(const Self & r) const
  {
    return (this->GetLuminance() <= r.GetLuminance());
  }
  bool
  operator<(const Self & r) const
  {
    return (this->GetLuminance() < r.GetLuminance());
  }
};
} // namespace itk

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc != 4)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " <InputImageFile> <OutputImageFile> <radius>" << std::
↪endl;
    return EXIT_FAILURE;
  }

  // Setup types
  constexpr unsigned int Dimension = 2;

  using MyPixelType = itk::myRGBPixel<unsigned char>;
  using MyImageType = itk::Image<MyPixelType, Dimension>;

  // Create and setup a reader
  using ReaderType = itk::ImageFileReader<MyImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  // Create and setup a median filter
  using FilterType = itk::MedianImageFilter<MyImageType, MyImageType>;
```

(continues on next page)

(continued from previous page)

```
  FilterType::Pointer medianFilter = FilterType::New();

  FilterType::InputSizeType radius;
  radius.Fill(std::stoi(argv[3]));

  medianFilter->SetRadius(radius);
  medianFilter->SetInput(reader->GetOutput());

  // Cast the custom myRBGPixel's to RGBPixel's
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, Dimension>;
  using CastType = itk::CastImageFilter<MyImageType, RGBImageType>;
  CastType::Pointer cast = CastType::New();
  cast->SetInput(medianFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<RGBImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(cast->GetOutput());
  writer->SetFileName(argv[2]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MedianImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>

Applies a median filter to an image.

Computes an image where a given pixel is the median value of the the pixels in a neighborhood about the corresponding input pixel.

A median filter is one of the family of nonlinear filters. It is used to smooth an image without being biased by outliers or shot noise.

This filter requires that the input pixel type provides an operator<() (LessThan Comparable).

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Median Filter Of An Image
- Median Filter Of An RGB Image

See itk::MedianImageFilter for additional documentation.

### Smooth Image With Discrete Gaussian Filter

### Synopsis

Smooth an image with a discrete Gaussian filter.

### Results

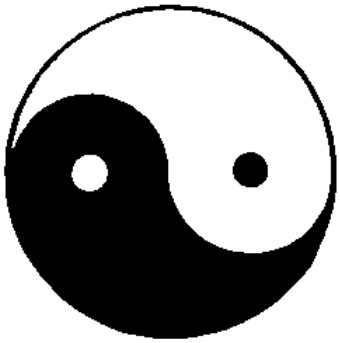


Fig. 327: Input image.

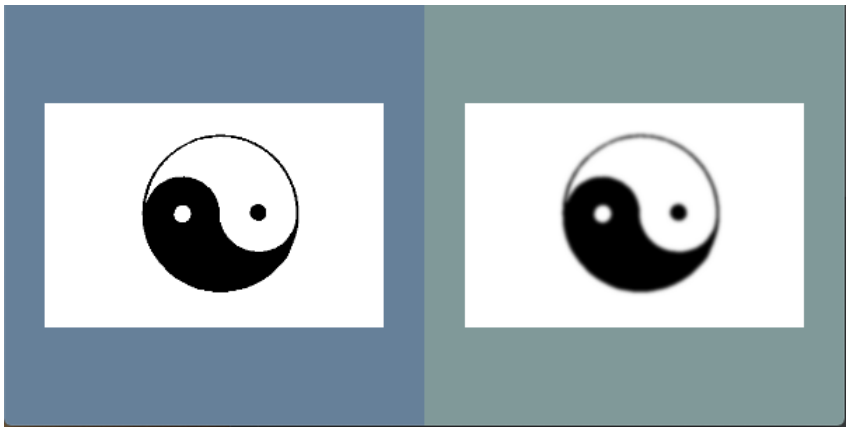


Fig. 328: Yinyang.png And Output.png With Variance = 8

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkDiscreteGaussianImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  // Verify command line arguments
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " inputImageFile [variance]" << std::endl;
    return EXIT_FAILURE;
  }

  // Parse command line argumentsa
  std::string inputFilename = argv[1];

  double variance = 4.0;
  if (argc > 2)
  {
    variance = std::stod(argv[2]);
  }

  // Setup types
  using UnsignedCharImageType = itk::Image<unsigned char, 2>;
  using FloatImageType = itk::Image<float, 2>;

  using readerType = itk::ImageFileReader<UnsignedCharImageType>;

  using filterType = itk::DiscreteGaussianImageFilter<UnsignedCharImageType,␣
↪FloatImageType>;

  // Create and setup a reader
  readerType::Pointer reader = readerType::New();
  reader->SetFileName(inputFilename.c_str());

  // Create and setup a Gaussian filter
  filterType::Pointer gaussianFilter = filterType::New();
  gaussianFilter->SetInput(reader->GetOutput());
  gaussianFilter->SetVariance(variance);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage<UnsignedCharImageType>(reader->GetOutput());
  viewer.AddImage<FloatImageType>(gaussianFilter->GetOutput());
  viewer.Visualize();
#endif
```

(continues on next page)

(continued from previous page)

```
  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *TInputImage*>
**class DiscreteGaussianImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Blurs an image by separable convolution with discrete gaussian kernels. This filter performs Gaussian blurring by separable convolution of an image and a discrete Gaussian operator (kernel).

The Gaussian operator used here was described by Tony Lindeberg (Discrete Scale-Space Theory and the Scale-Space Primal Sketch. Dissertation. Royal Institute of Technology, Stockholm, Sweden. May 1991.) The Gaussian kernel used here was designed so that smoothing and derivative operations commute after discretization.

The variance or standard deviation (sigma) will be evaluated as pixel units if SetUseImageSpacing is off (false) or as physical units if SetUseImageSpacing is on (true, default). The variance can be set independently in each dimension.

When the Gaussian kernel is small, this filter tends to run faster than itk::RecursiveGaussianImageFilter.

**See** GaussianOperator

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** RecursiveGaussianImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Computes the smoothing with Gaussian kernel

Subclassed by itk::GPUImageToImageFilter< TInputImage, TOutputImage, DiscreteGaussianImageFilter< TInputImage, TOutputImage > >

See itk::DiscreteGaussianImageFilter for additional documentation.

### 3.4.25 Thresholding

#### Demonstrate Available Threshold Algorithms

#### Synopsis

Demonstrate available threshold algorithms.

### Results

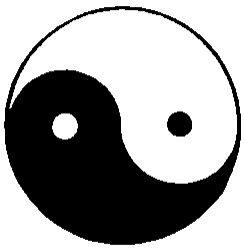


Fig. 329: Input Image

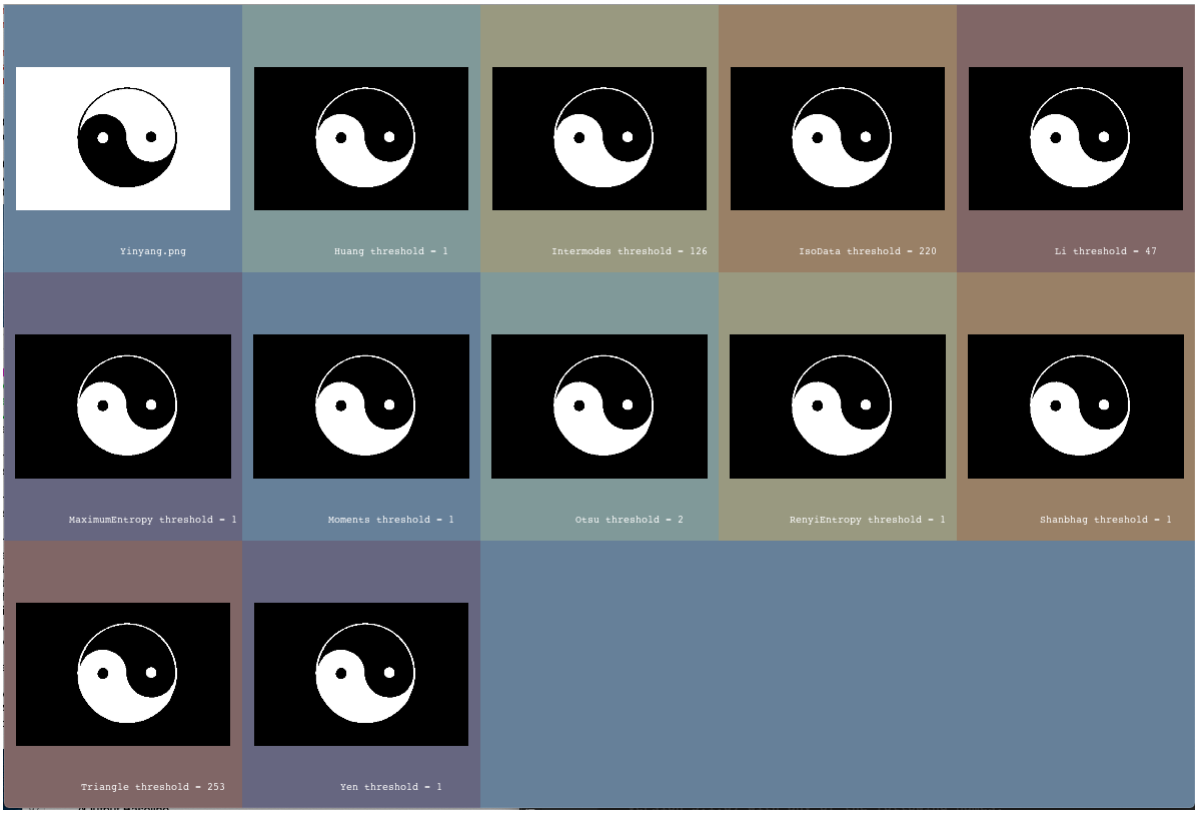



Fig. 330: Output In QuickView

### Code

### C++

```
#include "itkLiThresholdImageFilter.h"
#include "itkHuangThresholdImageFilter.h"
#include "itkIntermodesThresholdImageFilter.h"
#include "itkIsoDataThresholdImageFilter.h"
#include "itkKittlerIllingworthThresholdImageFilter.h"
#include "itkMaximumEntropyThresholdImageFilter.h"
#include "itkMomentsThresholdImageFilter.h"
#include "itkOtsuThresholdImageFilter.h"
#include "itkRenyiEntropyThresholdImageFilter.h"
#include "itkShanbhagThresholdImageFilter.h"
#include "itkTriangleThresholdImageFilter.h"
#include "itkYenThresholdImageFilter.h"

#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>
#include <map>
#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cout << "Usage: " << argv[0];
    std::cout << " inputImageFile";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  using InputPixelType = short;
  using OutputPixelType = unsigned char;

  using InputImageType = itk::Image<InputPixelType, 2>;
  using OutputImageType = itk::Image<OutputPixelType, 2>;

  using LiFilterType = itk::LiThresholdImageFilter<InputImageType, OutputImageType>;
  using HuangFilterType = itk::HuangThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using IntermodesFilterType = itk::IntermodesThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using IsoDataFilterType = itk::IsoDataThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  // TODO: Fix KittlerIllingworth Filter
  // using KittlerIllingworthFilterType = itk::KittlerIllingworthThresholdImageFilter
↪<InputImageType, OutputImageType>;
  using LiFilterType = itk::LiThresholdImageFilter<InputImageType, OutputImageType>;
  using MaximumEntropyFilterType = itk::MaximumEntropyThresholdImageFilter
↪<InputImageType, OutputImageType>;
```

(continues on next page)

(continued from previous page)

```
  using MomentsFilterType = itk::MomentsThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using OtsuFilterType = itk::OtsuThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using RenyiEntropyFilterType = itk::RenyiEntropyThresholdImageFilter<InputImageType,
↪ OutputImageType>;
  using ShanbhagFilterType = itk::ShanbhagThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using TriangleFilterType = itk::TriangleThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using YenFilterType = itk::YenThresholdImageFilter<InputImageType, OutputImageType>;

  using ReaderType = itk::ImageFileReader<InputImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  using FilterContainerType =
    std::map<std::string, itk::HistogramThresholdImageFilter<InputImageType,␣
↪OutputImageType>::Pointer>;
  FilterContainerType filterContainer;

  filterContainer["Huang"] = HuangFilterType::New();
  filterContainer["Intermodes"] = IntermodesFilterType::New();
  filterContainer["IsoData"] = IsoDataFilterType::New();
  // filterContainer["KittlerIllingworth"] = KittlerIllingworthFilterType::New();
  filterContainer["Li"] = LiFilterType::New();
  filterContainer["MaximumEntropy"] = MaximumEntropyFilterType::New();
  filterContainer["Moments"] = MomentsFilterType::New();
  filterContainer["Otsu"] = OtsuFilterType::New();
  filterContainer["RenyiEntropy"] = RenyiEntropyFilterType::New();
  filterContainer["Shanbhag"] = ShanbhagFilterType::New();
  filterContainer["Triangle"] = TriangleFilterType::New();
  filterContainer["Yen"] = YenFilterType::New();

  auto it = filterContainer.begin();
  for (it = filterContainer.begin(); it != filterContainer.end(); ++it)
  {
    (*it).second->SetInsideValue(255);
    (*it).second->SetOutsideValue(0);
    (*it).second->SetInput(reader->GetOutput());
    (*it).second->SetNumberOfHistogramBins(100);
    (*it).second->Update();
    std::stringstream desc;
    desc << (*it).first << " threshold = " << (*it).second->GetThreshold();
    viewer.AddImage((*it).second->GetOutput(), true, desc.str());
  }


  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
```

(continues on next page)

(continued from previous page)

```
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TOutputImage*>
**class LiThresholdImageFilter** : **public** itk::HistogramThresholdImageFilter<*TInputImage*, *TOutputImage*, *TMaskImage*>

Threshold an image using the Li Threshold.

This filter creates a binary thresholded image that separates an image into foreground and background components. The filter computes the threshold using the LiThresholdCalculator and applies that threshold to the input image using the BinaryThresholdImageFilter.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/811

**Author** Richard Beare. Department of Medicine, Monash University, Melbourne, Australia.

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** HistogramThresholdImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Demonstrate Available Threshold Algorithms

See itk::LiThresholdImageFilter for additional documentation.

### Separate Foreround and Background Using Otsu Method

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Separate foreground and background using Otsu's method.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkOtsuThresholdImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageRegionIterator.h"
#include "itkNumericTraits.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

namespace
{
using PixelType = unsigned char;
using ImageType = itk::Image<PixelType, 2>;
} // namespace

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image;
  if (argc < 2)
  {
    image = ImageType::New();
    CreateImage(image.GetPointer());
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();

    image = reader->GetOutput();
  }

  using FilterType = itk::OtsuThresholdImageFilter<ImageType, ImageType>;
  FilterType::Pointer otsuFilter = FilterType::New();
  otsuFilter->SetInput(image);
  otsuFilter->Update(); // To compute threshold

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
```

(continues on next page)

(continued from previous page)

```
  desc << "Otsu Threshold: " << itk::NumericTraits<FilterType::InputPixelType>::
↪PrintType(otsuFilter->GetThreshold());
  viewer.AddImage(otsuFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make the whole image white
  itk::ImageRegionIterator<ImageType> iterator(image, image->
↪GetLargestPossibleRegion());

  /*
   //Create a square
  while(!iterator.IsAtEnd())
    {
    iterator.Set(255);
    ++iterator;
    }
  */
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TOutputImage*>
**class OtsuThresholdImageFilter** : **public** itk::HistogramThresholdImageFilter<*TInputImage*, *TOutputImage*, *TMaskImag*

Threshold an image using the Otsu Threshold.

This filter creates a binary thresholded image that separates an image into foreground and background components. The filter computes the threshold using the OtsuThresholdCalculator and applies that threshold to the input image using the BinaryThresholdImageFilter.

This implementation was taken from the Insight Journal paper: https://www.insight-journal.org/browse/publication/811

**Author** Richard Beare

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** HistogramThresholdImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Separate Foreround And Background Using Otsu Method

See itk::OtsuThresholdImageFilter for additional documentation.

### Threshold an Image

### Synopsis

Threshold an image using itk::ThresholdImageFilter

### Results

Fig. 331: Input image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Threshold An Image.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("lower_threshold", type=int)
parser.add_argument("upper_threshold", type=int)
```

(continues on next page)

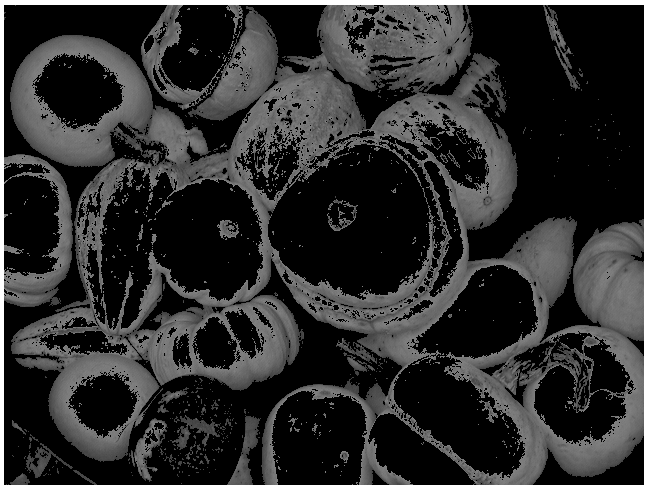


Fig. 332: Output image

(continued from previous page)

```
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

thresholdFilter = itk.ThresholdImageFilter[ImageType].New()

thresholdFilter.SetInput(reader.GetOutput())
thresholdFilter.ThresholdOutside(args.lower_threshold, args.upper_threshold)
thresholdFilter.SetOutsideValue(0)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(thresholdFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkThresholdImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 5)
  {
    std::cerr << "Usage: " << std::endl;
```

(continues on next page)

(continued from previous page)

```
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <Lower Threshold> <Upper␣
↪Threshold>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  unsigned char lowerThreshold = std::stoi(argv[3]);
  unsigned char upperThreshold = std::stoi(argv[4]);

  using FilterType = itk::ThresholdImageFilter<ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->ThresholdOutside(lowerThreshold, upperThreshold);
  filter->SetOutsideValue(0);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ThresholdImageFilter** : **public** itk::*InPlaceImageFilter<TImage, TImage>*

Set image values to a user-specified value if they are below, above, or between simple threshold values.

ThresholdImageFilter sets image values to a user-specified "outside" value (by default, "black") if the image values are below, above, or between simple threshold values.

The available methods are:

ThresholdAbove(): The values greater than the threshold value are set to OutsideValue

ThresholdBelow(): The values less than the threshold value are set to OutsideValue

ThresholdOutside(): The values outside the threshold range (less than lower or greater than upper) are set to OutsideValue

Note that these definitions indicate that pixels equal to the threshold value are not set to OutsideValue in any of these methods

The pixels must support the operators >= and <=.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Threshold An Image

See itk::ThresholdImageFilter for additional documentation.

### Threshold an Image Using Binary Thresholding

### Synopsis

Binarize an input image by thresholding.

### Results

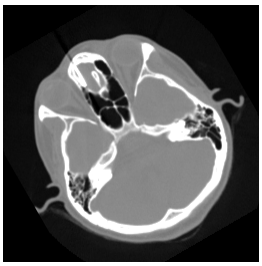


Fig. 333: Input image

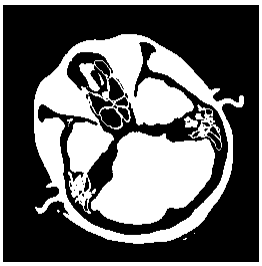


Fig. 334: Output image

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Threshold An Image Using Binary.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("lower_threshold", type=int)
parser.add_argument("upper_threshold", type=int)
parser.add_argument("outside_value", type=int)
parser.add_argument("inside_value", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

thresholdFilter = itk.BinaryThresholdImageFilter[ImageType, ImageType].New()
thresholdFilter.SetInput(reader.GetOutput())

thresholdFilter.SetLowerThreshold(args.lower_threshold)
thresholdFilter.SetUpperThreshold(args.upper_threshold)
thresholdFilter.SetOutsideValue(args.outside_value)
thresholdFilter.SetInsideValue(args.inside_value)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(thresholdFilter.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkBinaryThresholdImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 7)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << std::endl;
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " <InputImage> <OutputImage> <LowerThreshold>";
    std::cerr << " <UpperThreshold> <OutsideValue> <InsideValue>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  const char * InputImage = argv[1];
  const char * OutputImage = argv[2];

  const auto LowerThreshold = static_cast<PixelType>(atoi(argv[3]));
  const auto UpperThreshold = static_cast<PixelType>(atoi(argv[4]));
  const auto OutsideValue = static_cast<PixelType>(atoi(argv[5]));
  const auto InsideValue = static_cast<PixelType>(atoi(argv[6]));

  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(InputImage);

  using FilterType = itk::BinaryThresholdImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetLowerThreshold(LowerThreshold);
  filter->SetUpperThreshold(UpperThreshold);
  filter->SetOutsideValue(OutsideValue);
  filter->SetInsideValue(InsideValue);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(OutputImage);
  writer->SetInput(filter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class BinaryThresholdImageFilter** : **public** itk::*UnaryFunctorImageFilter<TInputImage*, *TOutputImage*, Functor::Binar
Binarize an input image by thresholding.

This filter produces an output image whose pixels are either one of two values ( OutsideValue or InsideValue ), depending on whether the corresponding input image pixels lie between the two thresholds ( LowerThreshold and UpperThreshold ). Values equal to either threshold is considered to be between the thresholds.

More precisely

$$Output(x_i) = \begin{cases} InsideValue & \text{if } LowerThreshold \leq x_i \leq UpperThreshold \\ OutsideValue & \text{otherwise} \end{cases}$$

This filter is templated over the input image type and the output image type.

The filter expect both images to have the same number of dimensions.

The default values for LowerThreshold and UpperThreshold are: LowerThreshold = NumericTraits<TInput>::NonpositiveMin(); UpperThreshold = NumericTraits<TInput>::max(); Therefore, generally only one of these needs to be set, depending on whether the user wants to threshold above or below the desired threshold.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Threshold An Image Using Binary Thresholding

Subclassed by itk::GPUImageToImageFilter< TInputImage, TOutputImage, BinaryThresholdImageFilter< TInputImage, TOutputImage > >

See itk::BinaryThresholdImageFilter for additional documentation.

### Threshold an Image Using Otsu

### Synopsis

Threshold an Image using Otsu's method.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Threshold An Image Using Otsu.")
```

(continues on next page)

Fig. 335: Input image

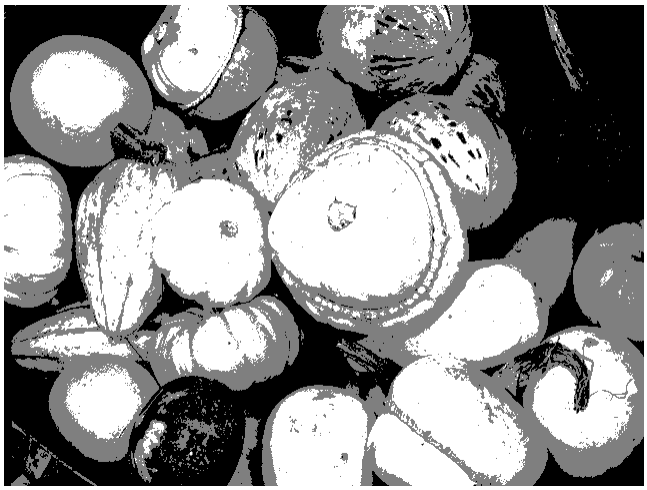


Fig. 336: Output image

(continued from previous page)

```
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("number_of_histogram_bins", type=int)
parser.add_argument("number_of_thresholds", type=int)
parser.add_argument("label_offset", type=int)
args = parser.parse_args()

PixelType = itk.UC
Dimension = 2

ImageType = itk.Image[PixelType, Dimension]

reader = itk.ImageFileReader[ImageType].New()
reader.SetFileName(args.input_image)

thresholdFilter = itk.OtsuMultipleThresholdsImageFilter[ImageType, ImageType].New()
thresholdFilter.SetInput(reader.GetOutput())

thresholdFilter.SetNumberOfHistogramBins(args.number_of_histogram_bins)
thresholdFilter.SetNumberOfThresholds(args.number_of_thresholds)
thresholdFilter.SetLabelOffset(args.label_offset)

rescaler = itk.RescaleIntensityImageFilter[ImageType, ImageType].New()
rescaler.SetInput(thresholdFilter.GetOutput())
rescaler.SetOutputMinimum(0)
rescaler.SetOutputMaximum(255)

writer = itk.ImageFileWriter[ImageType].New()
writer.SetFileName(args.output_image)
writer.SetInput(rescaler.GetOutput())

writer.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkOtsuMultipleThresholdsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc < 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << std::endl;
    std::cerr << " <InputImage> <OutputImage> <NumberOfBins>";
    std::cerr << " <NumberOfThresholds> <LabelOffset>" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
```

(continues on next page)

(continued from previous page)

```
  using SizeType = itk::SizeValueType;

  const char * InputImage = argv[1];
  const char * OutputImage = argv[2];

  const auto NumberOfHistogramBins = static_cast<SizeType>(atoi(argv[3]));
  const auto NumberOfThresholds = static_cast<SizeType>(atoi(argv[4]));
  const auto LabelOffset = static_cast<PixelType>(atoi(argv[5]));

  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(InputImage);

  using FilterType = itk::OtsuMultipleThresholdsImageFilter<ImageType, ImageType>;
  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetNumberOfHistogramBins(NumberOfHistogramBins);
  filter->SetNumberOfThresholds(NumberOfThresholds);
  filter->SetLabelOffset(LabelOffset);

  FilterType::ThresholdVectorType thresholds = filter->GetThresholds();

  std::cout << "Thresholds:" << std::endl;

  for (double threshold : thresholds)
  {
    std::cout << threshold << std::endl;
  }

  std::cout << std::endl;

  using RescaleType = itk::RescaleIntensityImageFilter<ImageType, ImageType>;
  RescaleType::Pointer rescaler = RescaleType::New();
  rescaler->SetInput(filter->GetOutput());
  rescaler->SetOutputMinimum(0);
  rescaler->SetOutputMaximum(255);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(OutputImage);
  writer->SetInput(rescaler->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << "Error: " << e << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class OtsuMultipleThresholdsImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Threshold an image using multiple Otsu Thresholds.

This filter creates a labeled image that separates the input image into various classes. The filter computes the thresholds using the OtsuMultipleThresholdsCalculator and applies those thresholds to the input image using the ThresholdLabelerImageFilter. The NumberOfHistogramBins and NumberOfThresholds can be set for the Calculator. The LabelOffset can be set for the ThresholdLabelerImageFilter.

This filter also includes an option to use the valley emphasis algorithm from H.F. Ng, "Automatic thresholding for defect detection", Pattern Recognition Letters, (27): 1644-1649, 2006. The valley emphasis algorithm is particularly effective when the object to be thresholded is small. See the following tests for examples: itkOtsuMultipleThresholdsImageFilterTest3 and itkOtsuMultipleThresholdsImageFilterTest4 To use this algorithm, simple call the setter: SetValleyEmphasis(true) It is turned off by default.

**See** ScalarImageToHistogramGenerator

**See** OtsuMultipleThresholdsCalculator

**See** ThresholdLabelerImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Threshold An Image Using Otsu

See itk::OtsuMultipleThresholdsImageFilter for additional documentation.

## 3.5 GPU

## 3.6 IO

### 3.6.1 GDCM

#### Read and Print DICOM Tags

#### Synopsis

Example to read, search and print DICOM image metadata tags

### Results

```
0008|0000 = 406
0008|0005 = ISO_IR 100
0008|0008 = ORIGINAL\PRIMARY\OTHER
0008|0016 = 1.2.840.10008.5.1.4.1.1.4
0008|0018 = 1.2.840.113619.2.133.1762890640.1886.1055165016.76
0008|0020 = 20030625
0008|0021 = 20030625
0008|0022 = 20030625
0008|0023 = 20030625
0008|0030 = 152734
0008|0031 = 153128
0008|0032 = 153128
0008|0033 = 153128
0008|0050 =
0008|0060 = MR
0008|0070 = GE MEDICAL SYSTEMS
0008|0080 = crd
0008|0090 =
0008|1010 = sig3ow0
0008|1030 = Tech_1
0008|103e = head scan
0008|1070 = RPM
0008|1090 = GENESIS_SIGNA
0010|0000 = 82
0010|0010 = Wes Turner
0010|0020 = 1111
0010|0030 =
0010|0040 = O
0010|1010 = 000Y
0010|1030 = 68.039
0010|21b0 =
0018|0000 = 428
0018|0020 = GR
0018|0021 = SS\SP
0018|0022 = FC\VB_GEMS\VASCTOF_GEMS\PFF
0018|0023 = 3D
0018|0025 = N
0018|0050 =  1.6
0018|0080 = 23
0018|0081 = 10
0018|0083 =  1
0018|0084 =  6.38883e+08
0018|0085 = H1
0018|0086 =  1
0018|0087 = 15000
0018|0088 = 1.6
0018|0091 = 0
0018|0093 =  100
0018|0094 = 75
0018|0095 =  97.6562
0018|1000 = 0000018387rdmr3
0018|1020 = 09
0018|1050 =  1.14585
0018|1088 = 0
0018|1090 = 0
```

(continues on next page)

(continued from previous page)

```
0018|1094 = 0
0018|1100 =  220
0018|1250 = HEAD
0018|1251 = HEAD
0018|1310 = 0\256\192\0
0018|1312 = ROW
0018|1314 = 25
0018|1315 = N
0018|1316 = 0.004169
0018|5100 = HFS
0020|0000 = 340
0020|000d = 1.2.840.113619.2.133.1762890640.1886.1055165015.961
0020|000e = 1.2.840.113619.2.133.1762890640.1886.1055165015.999
0020|0010 = 361
0020|0011 =  3
0020|0012 =  0
0020|0013 = 77
0020|0032 = -112\-18.8578\ 128.388
0020|0037 =   1\ 0\ 0\ 0\0.466651\ -0.884442
0020|0052 = 1.2.840.113619.2.133.1762890640.1910.1055164903.673
0020|0060 =
0020|0110 =  0
0020|1040 =
0020|1041 = -32.4746742249
0028|0000 = 148
0028|0002 = 1
0028|0004 = MONOCHROME2
0028|0010 = 256
0028|0011 = 256
0028|0030 =   0.85939\0.859375
0028|0100 = 16
0028|0101 = 16
0028|0102 = 15
0028|0103 = 1
0028|0120 = 0
0028|1050 = 86
0028|1051 =  173
7fe0|0000 = 131084
Patient's Name (0010|0010)  is: Wes Turner
```

### Code

### C++

```
//  This example illustrates how to read a DICOM series into a volume and then
//  print most of the DICOM header information. The binary fields are skipped.

#include "itkImageSeriesReader.h"
#include "itkGDCMImageIO.h"
#include "itkGDCMSeriesFileNames.h"

int
main(int argc, char * argv[])
{
```

(continues on next page)

(continued from previous page)

```
  if (argc < 2)
  {
    std::cerr << "Usage: " << argv[0] << " DicomDirectory " << std::endl;
    return EXIT_FAILURE;
  }

  //  Next, we instantiate the type to be used for storing the image once it is
  //  read into memory.
  using PixelType = signed short;
  constexpr unsigned int Dimension = 3;

  using ImageType = itk::Image<PixelType, Dimension>;

  // We use the image type for instantiating the series reader type and then we
  // construct one object of this class.
  using ReaderType = itk::ImageSeriesReader<ImageType>;

  ReaderType::Pointer reader = ReaderType::New();

  // A GDCMImageIO object is created and assigned to the reader.
  using ImageIOType = itk::GDCMImageIO;

  ImageIOType::Pointer dicomIO = ImageIOType::New();

  reader->SetImageIO(dicomIO);

  // A GDCMSeriesFileNames is declared in order to generate the names of DICOM
  // slices. We specify the directory with the SetInputDirectory() method
  // and, in this case, take the directory name from the command line arguments.
  // You could have obtained the directory name from a file dialog in a GUI.

  using NamesGeneratorType = itk::GDCMSeriesFileNames;

  NamesGeneratorType::Pointer nameGenerator = NamesGeneratorType::New();

  nameGenerator->SetInputDirectory(argv[1]);

  // The list of files to read is obtained from the name generator by invoking
  // the GetInputFileNames() method and receiving the results in a
  // container of strings. The list of filenames is passed to the reader using
  // the SetFileNames() method.
  using FileNamesContainer = std::vector<std::string>;
  FileNamesContainer fileNames = nameGenerator->GetInputFileNames();

  reader->SetFileNames(fileNames);

  // We trigger the reader by invoking the Update() method. This
  // invocation should normally be done inside a try/catch block given
  // that it may eventually throw exceptions.
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & ex)
  {
    std::cout << ex << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }

  // ITK internally queries GDCM and obtains all the DICOM tags from the file
  // headers. The tag values are stored in the MetaDataDictionary
  // which is a general-purpose container for \{key,value\} pairs. The Metadata
  // dictionary can be recovered from any ImageIO class by invoking the
  // GetMetaDataDictionary() method.
  using DictionaryType = itk::MetaDataDictionary;

  const DictionaryType & dictionary = dicomIO->GetMetaDataDictionary();

  // In this example, we are only interested in the DICOM tags that can be
  // represented as strings. Therefore, we declare a MetaDataObject of
  // string type in order to receive those particular values.
  using MetaDataStringType = itk::MetaDataObject<std::string>;

  // The metadata dictionary is organized as a container with its corresponding
  // iterators. We can therefore visit all its entries by first getting access to
  // its Begin() and End() methods.
  auto itr = dictionary.Begin();
  auto end = dictionary.End();

  // We are now ready for walking through the list of DICOM tags. For this
  // purpose we use the iterators that we just declared. At every entry we
  // attempt to convert it into a string entry by using the dynamic_cast
  // based on RTTI information. The
  // dictionary is organized like a std::map structure, so we should use
  // the first and second members of every entry in order
  // to get access to the \{key,value\} pairs.
  while (itr != end)
  {
    itk::MetaDataObjectBase::Pointer entry = itr->second;

    MetaDataStringType::Pointer entryvalue = dynamic_cast<MetaDataStringType *>(entry.
↪GetPointer());

    if (entryvalue)
    {
      std::string tagkey = itr->first;
      std::string tagvalue = entryvalue->GetMetaDataObjectValue();
      std::cout << tagkey << " = " << tagvalue << std::endl;
    }

    ++itr;
  }

  //  It is also possible to query for specific entries instead of reading all of
  //  them as we did above. In this case, the user must provide the tag
  //  identifier using the standard DICOM encoding. The identifier is stored in a
  //  string and used as key in the dictionary.
  std::string entryId = "0010|0010";

  auto tagItr = dictionary.Find(entryId);

  if (tagItr == end)
  {
    std::cerr << "Tag " << entryId;
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " not found in the DICOM header" << std::endl;
    return EXIT_FAILURE;
  }

  // Since the entry may or may not be of string type we must again use a
  // dynamic_cast in order to attempt to convert it to a string dictionary
  // entry. If the conversion is successful, we can then print out its content.
  MetaDataStringType::ConstPointer entryvalue = dynamic_cast<const MetaDataStringType␣
↪*>(tagItr->second.GetPointer());

  if (entryvalue)
  {
    std::string tagvalue = entryvalue->GetMetaDataObjectValue();
    std::cout << "Patient's Name (" << entryId << ") ";
    std::cout << " is: " << tagvalue << std::endl;
  }
  else
  {
    std::cerr << "Entry was not of string type" << std::endl;
    return EXIT_FAILURE;
  }

  // This type of functionality will probably be more useful when provided
  // through a graphical user interface. For a full description of the DICOM
  // dictionary please look at the following file.

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

parser = argparse.ArgumentParser(description="Read And Print DICOM Tags.")
parser.add_argument(
    "dicom_directory",
    nargs="?",
    help="If DicomDirectory is not specified, current directory is used",
)
args = parser.parse_args()

# current directory by default
dirName = "."
if args.dicom_directory:
    dirName = args.dicom_directory

# Setup the image readers with their type
PixelType = itk.ctype("signed short")
Dimension = 3
```

(continues on next page)

(continued from previous page)

```
ImageType = itk.Image[PixelType, Dimension]

# Using GDCMSeriesFileNames to generate the names of
# DICOM files.
namesGenerator = itk.GDCMSeriesFileNames.New()
namesGenerator.SetUseSeriesDetails(True)
namesGenerator.SetDirectory(dirName)

# Get the names of files
fileNames = namesGenerator.GetInputFileNames()

# Setup the image series reader using GDCMImageIO
reader = itk.ImageSeriesReader[ImageType].New()
dicomIO = itk.GDCMImageIO.New()
dicomIO.LoadPrivateTagsOn()
reader.SetImageIO(dicomIO)
reader.SetFileNames(fileNames)

# Attempt to read the series, exit if unable to.
try:
    reader.Update()
except:
    print("Error occured while reading DICOMs in: " + dirName)
    sys.exit(1)

# ITK internally queries GDCM and obtains all the DICOM tags from the file
# headers. The tag values are stored in the MetaDataDictionary
# which is a general-purpose container for \{key,value\} pairs. The Metadata
# dictionary can be recovered from any ImageIO class by invoking the
# GetMetaDataDictionary() method.
metadata = dicomIO.GetMetaDataDictionary()

# Print the key value pairs from the metadadictionary
tagkeys = metadata.GetKeys()

for tagkey in tagkeys:
    # Note the [] operator for the key
    try:
        tagvalue = metadata[tagkey]
        print(tagkey + "=" + str(tagvalue))
    except RuntimeError:
        # Cannot pass specialized values into metadata dictionary.
        print("Cannot pass specialized value" + tagkey + "into metadadictionary")


# Illustrating use of getting a label given a tag here
entryID = "0010|0010"
if not metadata.HasKey(entryID):
    print("tag: " + entryID + " not found in series")
else:
    # The second parameter is mandatory in python to get the
    # string label value
    label = itk.GDCMImageIO.GetLabelFromTag(entryID, "")
    tagvalue = metadata[entryID]
    print(label[1] + " (" + entryID + ") is: " + str(tagvalue))
```

### Classes demonstrated

**class GDCMImageIO** : **public** itk::*ImageIOBase*

ImageIO class for reading and writing DICOM V3.0 and ACR/NEMA 1&2 uncompressed images. This class is only an adaptor to the GDCM library.

GDCM can be found at: http://sourceforge.net/projects/gdcm

To learn more about the revision shipped with ITK, call

git log Modules/ThirdParty/GDCM/src/

from an ITK Git checkout.

The compressors supported include "JPEG2000" (default), and "JPEG". The compression level parameter is not supported.

**Warning** There are several restrictions to this current writer:

- Even though during the writing process you pass in a DICOM file as input The output file may not contains ALL DICOM field from the input file. In particular:
  - The SeQuence DICOM field (SQ).
  - Fields from Private Dictionary.
- Some very long (>0xfff) binary fields are not loaded (typically 0029|0010), you need to explicitly set the maximum length of elements to load to be bigger (see Get/SetMaxSizeLoadEntry).
- In DICOM some fields are stored directly using their binary representation. When loaded into the MetaDataDictionary some fields are converted to ASCII (only VR: OB/OW/OF and UN are encoded as mime64).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Resample DICOM Series
- Read DICOM Series and Write 3D Image

See itk::GDCMImageIO for additional documentation.

### Read DICOM Series and Write 3D Image

### Synopsis

This example reads all the DICOM series in a given folder argv[1] and writes them in the same folder with following file pattern: seriesIdentifier.nrrd

if output file name argv[2] and series name argv[3] are given, then it behaves like DicomSeriesReadImageWrite2.cxx (writing just the requested series with the specified name).

Based on DicomSeriesReadImageWrite2.cxx

Author: Dženan Zukić <dzenan.zukic@kitware.com>

### Code

### Python

```
#!/usr/bin/env python

import sys
import os
import itk
import argparse

parser = argparse.ArgumentParser(description="Read DICOM Series And Write 3D Image.")
parser.add_argument(
    "dicom_directory",
    nargs="?",
    help="If DicomDirectory is not specified, current directory is used",
)
parser.add_argument("output_image", nargs="?")
parser.add_argument("series_name", nargs="?")
args = parser.parse_args()

# current directory by default
dirName = "."
if args.dicom_directory:
    dirName = args.dicom_directory

PixelType = itk.ctype("signed short")
Dimension = 3

ImageType = itk.Image[PixelType, Dimension]

namesGenerator = itk.GDCMSeriesFileNames.New()
namesGenerator.SetUseSeriesDetails(True)
namesGenerator.AddSeriesRestriction("0008|0021")
namesGenerator.SetGlobalWarningDisplay(False)
namesGenerator.SetDirectory(dirName)

seriesUID = namesGenerator.GetSeriesUIDs()

if len(seriesUID) < 1:
    print("No DICOMs in: " + dirName)
    sys.exit(1)

print("The directory: " + dirName)
print("Contains the following DICOM Series: ")
for uid in seriesUID:
    print(uid)

seriesFound = False
for uid in seriesUID:
    seriesIdentifier = uid
    if args.series_name:
        seriesIdentifier = args.series_name
        seriesFound = True
    print("Reading: " + seriesIdentifier)
    fileNames = namesGenerator.GetFileNames(seriesIdentifier)
```

(continues on next page)

(continued from previous page)

```
    reader = itk.ImageSeriesReader[ImageType].New()
    dicomIO = itk.GDCMImageIO.New()
    reader.SetImageIO(dicomIO)
    reader.SetFileNames(fileNames)
    reader.ForceOrthogonalDirectionOff()

    writer = itk.ImageFileWriter[ImageType].New()
    outFileName = os.path.join(dirName, seriesIdentifier + ".nrrd")
    if args.output_image:
        outFileName = args.output_image
    writer.SetFileName(outFileName)
    writer.UseCompressionOn()
    writer.SetInput(reader.GetOutput())
    print("Writing: " + outFileName)
    writer.Update()

    if seriesFound:
        break
```

### C++

```
#include "itkImage.h"
#include "itkGDCMImageIO.h"
#include "itkGDCMSeriesFileNames.h"
#include "itkImageSeriesReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " [DicomDirectory  [outputFileName  [seriesName]]]";
    std::cerr << "\nIf DicomDirectory is not specified, current directory is used\n";
  }
  std::string dirName = "."; // current directory by default
  if (argc > 1)
  {
    dirName = argv[1];
  }

  using PixelType = signed short;
  constexpr unsigned int Dimension = 3;
  using ImageType = itk::Image<PixelType, Dimension>;

  using NamesGeneratorType = itk::GDCMSeriesFileNames;
  NamesGeneratorType::Pointer nameGenerator = NamesGeneratorType::New();

  nameGenerator->SetUseSeriesDetails(true);
  nameGenerator->AddSeriesRestriction("0008|0021");
  nameGenerator->SetGlobalWarningDisplay(false);
  nameGenerator->SetDirectory(dirName);
```

(continues on next page)

(continued from previous page)

```
  try
  {
    using SeriesIdContainer = std::vector<std::string>;
    const SeriesIdContainer & seriesUID = nameGenerator->GetSeriesUIDs();
    auto                      seriesItr = seriesUID.begin();
    auto                      seriesEnd = seriesUID.end();

    if (seriesItr != seriesEnd)
    {
      std::cout << "The directory: ";
      std::cout << dirName << std::endl;
      std::cout << "Contains the following DICOM Series: ";
      std::cout << std::endl;
    }
    else
    {
      std::cout << "No DICOMs in: " << dirName << std::endl;
      return EXIT_SUCCESS;
    }

    while (seriesItr != seriesEnd)
    {
      std::cout << seriesItr->c_str() << std::endl;
      ++seriesItr;
    }

    seriesItr = seriesUID.begin();
    while (seriesItr != seriesUID.end())
    {
      std::string seriesIdentifier;
      if (argc > 3) // If seriesIdentifier given convert only that
      {
        seriesIdentifier = argv[3];
        seriesItr = seriesUID.end();
      }
      else // otherwise convert everything
      {
        seriesIdentifier = seriesItr->c_str();
        seriesItr++;
      }
      std::cout << "\nReading: ";
      std::cout << seriesIdentifier << std::endl;
      using FileNamesContainer = std::vector<std::string>;
      FileNamesContainer fileNames = nameGenerator->GetFileNames(seriesIdentifier);

      using ReaderType = itk::ImageSeriesReader<ImageType>;
      ReaderType::Pointer reader = ReaderType::New();
      using ImageIOType = itk::GDCMImageIO;
      ImageIOType::Pointer dicomIO = ImageIOType::New();
      reader->SetImageIO(dicomIO);
      reader->SetFileNames(fileNames);
      reader->ForceOrthogonalDirectionOff(); // properly read CTs with gantry tilt

      using WriterType = itk::ImageFileWriter<ImageType>;
      WriterType::Pointer writer = WriterType::New();
      std::string         outFileName;
      if (argc > 2)
```

(continues on next page)

(continued from previous page)

```
      {
        outFileName = argv[2];
      }
      else
      {
        outFileName = dirName + std::string("/") + seriesIdentifier + ".nrrd";
      }
      writer->SetFileName(outFileName);
      writer->UseCompressionOn();
      writer->SetInput(reader->GetOutput());
      std::cout << "Writing: " << outFileName << std::endl;
      try
      {
        writer->Update();
      }
      catch (itk::ExceptionObject & ex)
      {
        std::cout << ex << std::endl;
        continue;
      }
    }
  }
  catch (itk::ExceptionObject & ex)
  {
    std::cout << ex << std::endl;
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class GDCMSeriesFileNames** : **public** itk::*ProcessObject*

Generate a sequence of filenames from a DICOM series.

This class generates a sequence of files whose filenames point to a DICOM file. The ordering is based on the following strategy: Read all images in the directory (assuming there is only one study/series)

a. Extract Image Orientation & Image Position from DICOM images, and then calculate the ordering based on the 3D coordinate of the slice.

b. If for some reason this information is not found or failed, another strategy is used: the ordering is based on 'Instance Number'.

c. If this strategy also failed, then the filenames are ordered by lexicographical order.

If multiple volumes are being grouped as a single series for your DICOM objects, you may want to try calling SetUseSeriesDetails(true) prior to calling SetDirectory().

See itk::GDCMSeriesFileNames for additional documentation.

**class GDCMImageIO** : **public** itk::*ImageIOBase*

ImageIO class for reading and writing DICOM V3.0 and ACR/NEMA 1&2 uncompressed images. This class is only an adaptor to the GDCM library.

GDCM can be found at: http://sourceforge.net/projects/gdcm

To learn more about the revision shipped with ITK, call

git log Modules/ThirdParty/GDCM/src/

from an ITK Git checkout.

The compressors supported include "JPEG2000" (default), and "JPEG". The compression level parameter is not supported.

**Warning** There are several restrictions to this current writer:

- Even though during the writing process you pass in a DICOM file as input The output file may not contains ALL DICOM field from the input file. In particular:
  - The SeQuence DICOM field (SQ).
  - Fields from Private Dictionary.
- Some very long (>0xfff) binary fields are not loaded (typically 0029|0010), you need to explicitly set the maximum length of elements to load to be bigger (see Get/SetMaxSizeLoadEntry).
- In DICOM some fields are stored directly using their binary representation. When loaded into the MetaDataDictionary some fields are converted to ASCII (only VR: OB/OW/OF and UN are encoded as mime64).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Resample DICOM Series
- Read DICOM Series and Write 3D Image

See itk::GDCMImageIO for additional documentation.

template<typename **TOutputImage**>
**class ImageSeriesReader** : **public** itk::*ImageSource*<*TOutputImage*>

Data source that reads image data from a series of disk files.

This class builds an n-dimension image from multiple n-1 dimension image files. The files stored in a vector of strings are read using the ImageFileReader. File format may vary between the files, but the image data must have the same Size for all dimensions.

**See** GDCMSeriesFileNames

**See** NumericSeriesFileNames

See itk::ImageSeriesReader for additional documentation.

### Resample DICOM Series

### Synopsis

Resample a DICOM series.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
// The program progresses as follows:
// 1) Read the input series
// 2) Resample the series according to the user specified x-y-z
//    spacing.
// 3) Create a MetaDataDictionary for each slice.
// 4) Shift data to undo the effect of a rescale intercept by the
//    DICOM reader (only for ITK < 4.6)
// 5) Write the new DICOM series
//

#include "itkVersion.h"

#include "itkImage.h"
#include "itkMinimumMaximumImageFilter.h"

#include "itkGDCMImageIO.h"
#include "itkGDCMSeriesFileNames.h"
#include "itkNumericSeriesFileNames.h"

#include "itkImageSeriesReader.h"
#include "itkImageSeriesWriter.h"

#include "itkResampleImageFilter.h"

#if ((ITK_VERSION_MAJOR == 4) && (ITK_VERSION_MINOR < 6))
#  include "itkShiftScaleImageFilter.h"
#endif

#include "itkIdentityTransform.h"
#include "itkLinearInterpolateImageFunction.h"

#include <itksys/SystemTools.hxx>

#if ITK_VERSION_MAJOR >= 4
#  include "gdcmUIDGenerator.h"
#else
#  include "gdcm/src/gdcmFile.h"
#  include "gdcm/src/gdcmUtil.h"
#endif

#include <string>
#include <sstream>
```

(continues on next page)

(continued from previous page)

```
static void
CopyDictionary(itk::MetaDataDictionary & fromDict, itk::MetaDataDictionary & toDict);


int
main(int argc, char * argv[])
{
  // Validate input parameters
  if (argc < 4)
  {
    std::cerr << "Usage: " << argv[0] << " InputDicomDirectory OutputDicomDirectory␣
↪spacing_x spacing_y spacing_z"
              << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int InputDimension = 3;
  constexpr unsigned int OutputDimension = 2;

  using PixelType = signed short;

  using InputImageType = itk::Image<PixelType, InputDimension>;
  using OutputImageType = itk::Image<PixelType, OutputDimension>;
  using ReaderType = itk::ImageSeriesReader<InputImageType>;
  using ImageIOType = itk::GDCMImageIO;
  using InputNamesGeneratorType = itk::GDCMSeriesFileNames;
  using OutputNamesGeneratorType = itk::NumericSeriesFileNames;
  using TransformType = itk::IdentityTransform<double, InputDimension>;
  using InterpolatorType = itk::LinearInterpolateImageFunction<InputImageType, double>
↪;
  using ResampleFilterType = itk::ResampleImageFilter<InputImageType, InputImageType>;
#if ((ITK_VERSION_MAJOR == 4) && (ITK_VERSION_MINOR < 6))
  using ShiftScaleType = itk::ShiftScaleImageFilter<InputImageType, InputImageType>;
#endif
  using SeriesWriterType = itk::ImageSeriesWriter<InputImageType, OutputImageType>;

  ////////////////////////////////////////////////
  // 1) Read the input series

  ImageIOType::Pointer             gdcmIO = ImageIOType::New();
  InputNamesGeneratorType::Pointer inputNames = InputNamesGeneratorType::New();
  inputNames->SetInputDirectory(argv[1]);

  const ReaderType::FileNamesContainer & filenames = inputNames->GetInputFileNames();

  ReaderType::Pointer reader = ReaderType::New();

  reader->SetImageIO(gdcmIO);
  reader->SetFileNames(filenames);
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & excp)
  {
    std::cerr << "Exception thrown while reading the series" << std::endl;
    std::cerr << excp << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  ////////////////////////////////////////////////
  // 2) Resample the series
  InterpolatorType::Pointer interpolator = InterpolatorType::New();

  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  const InputImageType::SpacingType & inputSpacing = reader->GetOutput()->
↪GetSpacing();
  const InputImageType::RegionType &  inputRegion = reader->GetOutput()->
↪GetLargestPossibleRegion();
  const InputImageType::SizeType &    inputSize = inputRegion.GetSize();

  std::cout << "The input series in directory " << argv[1] << " has " << filenames.
↪size() << " files with spacing "
            << inputSpacing << std::endl;

  // Compute the size of the output. The user specifies a spacing on
  // the command line. If the spacing is 0, the input spacing will be
  // used. The size (# of pixels) in the output is recomputed using
  // the ratio of the input and output sizes.
  InputImageType::SpacingType outputSpacing;
  outputSpacing[0] = std::stod(argv[3]);
  outputSpacing[1] = std::stod(argv[4]);
  outputSpacing[2] = std::stod(argv[5]);

  bool changeInSpacing = false;
  for (unsigned int i = 0; i < 3; i++)
  {
    if (outputSpacing[i] == 0.0)
    {
      outputSpacing[i] = inputSpacing[i];
    }
    else
    {
      changeInSpacing = true;
    }
  }
  InputImageType::SizeType outputSize;
  using SizeValueType = InputImageType::SizeType::SizeValueType;
  outputSize[0] = static_cast<SizeValueType>(inputSize[0] * inputSpacing[0] /␣
↪outputSpacing[0] + .5);
  outputSize[1] = static_cast<SizeValueType>(inputSize[1] * inputSpacing[1] /␣
↪outputSpacing[1] + .5);
  outputSize[2] = static_cast<SizeValueType>(inputSize[2] * inputSpacing[2] /␣
↪outputSpacing[2] + .5);

  ResampleFilterType::Pointer resampler = ResampleFilterType::New();
  resampler->SetInput(reader->GetOutput());
  resampler->SetTransform(transform);
  resampler->SetInterpolator(interpolator);
  resampler->SetOutputOrigin(reader->GetOutput()->GetOrigin());
  resampler->SetOutputSpacing(outputSpacing);
  resampler->SetOutputDirection(reader->GetOutput()->GetDirection());
```

(continues on next page)

(continued from previous page)

```
  resampler->SetSize(outputSize);
  resampler->Update();


  ////////////////////////////////////////////////
  // 3) Create a MetaDataDictionary for each slice.

  // Copy the dictionary from the first image and override slice
  // specific fields
  ReaderType::DictionaryRawPointer inputDict = (*(reader->
↪GetMetaDataDictionaryArray()))[0];
  ReaderType::DictionaryArrayType  outputArray;

  // To keep the new series in the same study as the original we need
  // to keep the same study UID. But we need new series and frame of
  // reference UID's.
#if ITK_VERSION_MAJOR >= 4
  gdcm::UIDGenerator suid;
  std::string        seriesUID = suid.Generate();
  gdcm::UIDGenerator fuid;
  std::string        frameOfReferenceUID = fuid.Generate();
#else
  std::string seriesUID = gdcm::Util::CreateUniqueUID(gdcmIO->GetUIDPrefix());
  std::string frameOfReferenceUID = gdcm::Util::CreateUniqueUID(gdcmIO->
↪GetUIDPrefix());
#endif
  std::string studyUID;
  std::string sopClassUID;
  itk::ExposeMetaData<std::string>(*inputDict, "0020|000d", studyUID);
  itk::ExposeMetaData<std::string>(*inputDict, "0008|0016", sopClassUID);
  gdcmIO->KeepOriginalUIDOn();

  for (unsigned int f = 0; f < outputSize[2]; f++)
  {
    // Create a new dictionary for this slice
    auto dict = new ReaderType::DictionaryType;

    // Copy the dictionary from the first slice
    CopyDictionary(*inputDict, *dict);

    // Set the UID's for the study, series, SOP  and frame of reference
    itk::EncapsulateMetaData<std::string>(*dict, "0020|000d", studyUID);
    itk::EncapsulateMetaData<std::string>(*dict, "0020|000e", seriesUID);
    itk::EncapsulateMetaData<std::string>(*dict, "0020|0052", frameOfReferenceUID);

#if ITK_VERSION_MAJOR >= 4
    gdcm::UIDGenerator sopuid;
    std::string        sopInstanceUID = sopuid.Generate();
#else
    std::string sopInstanceUID = gdcm::Util::CreateUniqueUID(gdcmIO->GetUIDPrefix());
#endif
    itk::EncapsulateMetaData<std::string>(*dict, "0008|0018", sopInstanceUID);
    itk::EncapsulateMetaData<std::string>(*dict, "0002|0003", sopInstanceUID);

    // Change fields that are slice specific
    std::ostringstream value;
    value.str("");
```

(continues on next page)

(continued from previous page)

```
    value << f + 1;

    // Image Number
    itk::EncapsulateMetaData<std::string>(*dict, "0020|0013", value.str());

    // Series Description - Append new description to current series
    // description
    std::string oldSeriesDesc;
    itk::ExposeMetaData<std::string>(*inputDict, "0008|103e", oldSeriesDesc);

    value.str("");
    value << oldSeriesDesc << ": Resampled with pixel spacing " << outputSpacing[0] <
↪< ", " << outputSpacing[1] << ", "
          << outputSpacing[2];
    // This is an long string and there is a 64 character limit in the
    // standard
    unsigned lengthDesc = value.str().length();

    std::string seriesDesc(value.str(), 0, lengthDesc > 64 ? 64 : lengthDesc);
    itk::EncapsulateMetaData<std::string>(*dict, "0008|103e", seriesDesc);

    // Series Number
    value.str("");
    value << 1001;
    itk::EncapsulateMetaData<std::string>(*dict, "0020|0011", value.str());

    // Derivation Description - How this image was derived
    value.str("");
    for (int i = 0; i < argc; i++)
    {
      value << argv[i] << " ";
    }
    value << ": " << ITK_SOURCE_VERSION;

    lengthDesc = value.str().length();
    std::string derivationDesc(value.str(), 0, lengthDesc > 1024 ? 1024 : lengthDesc);
    itk::EncapsulateMetaData<std::string>(*dict, "0008|2111", derivationDesc);

    // Image Position Patient: This is calculated by computing the
    // physical coordinate of the first pixel in each slice.
    InputImageType::PointType position;
    InputImageType::IndexType index;
    index[0] = 0;
    index[1] = 0;
    index[2] = f;
    resampler->GetOutput()->TransformIndexToPhysicalPoint(index, position);

    value.str("");
    value << position[0] << "\\" << position[1] << "\\" << position[2];
    itk::EncapsulateMetaData<std::string>(*dict, "0020|0032", value.str());
    // Slice Location: For now, we store the z component of the Image
    // Position Patient.
    value.str("");
    value << position[2];
    itk::EncapsulateMetaData<std::string>(*dict, "0020|1041", value.str());

    if (changeInSpacing)
```

(continues on next page)

(continued from previous page)

```
    {
      // Slice Thickness: For now, we store the z spacing
      value.str("");
      value << outputSpacing[2];
      itk::EncapsulateMetaData<std::string>(*dict, "0018|0050", value.str());
      // Spacing Between Slices
      itk::EncapsulateMetaData<std::string>(*dict, "0018|0088", value.str());
    }

    // Save the dictionary
    outputArray.push_back(dict);
  }

#if ((ITK_VERSION_MAJOR == 4) && (ITK_VERSION_MINOR < 6))
  ////////////////////////////////////////////////
  // 4) Shift data to undo the effect of a rescale intercept by the
  //    DICOM reader
  std::string interceptTag("0028|1052");
  using MetaDataStringType = itk::MetaDataObject<std::string>;
  itk::MetaDataObjectBase::Pointer entry = (*inputDict)[interceptTag];

  MetaDataStringType::ConstPointer interceptValue = dynamic_cast<const␣
↪MetaDataStringType *>(entry.GetPointer());

  int interceptShift = 0;
  if (interceptValue)
  {
    std::string tagValue = interceptValue->GetMetaDataObjectValue();
    interceptShift = -atoi(tagValue.c_str());
  }

  ShiftScaleType::Pointer shiftScale = ShiftScaleType::New();
  shiftScale->SetInput(resampler->GetOutput());
  shiftScale->SetShift(interceptShift);
#endif

  ////////////////////////////////////////////////
  // 5) Write the new DICOM series

  // Make the output directory and generate the file names.
  itksys::SystemTools::MakeDirectory(argv[2]);

  // Generate the file names
  OutputNamesGeneratorType::Pointer outputNames = OutputNamesGeneratorType::New();
  std::string                       seriesFormat(argv[2]);
  seriesFormat = seriesFormat + "/" + "IM%d.dcm";
  outputNames->SetSeriesFormat(seriesFormat.c_str());
  outputNames->SetStartIndex(1);
  outputNames->SetEndIndex(outputSize[2]);

  SeriesWriterType::Pointer seriesWriter = SeriesWriterType::New();
#if ((ITK_VERSION_MAJOR == 4) && (ITK_VERSION_MINOR < 6))
  seriesWriter->SetInput(shiftScale->GetOutput());
#else
  seriesWriter->SetInput(resampler->GetOutput());
#endif
  seriesWriter->SetImageIO(gdcmIO);
```

(continues on next page)

(continued from previous page)

```
  seriesWriter->SetFileNames(outputNames->GetFileNames());
  seriesWriter->SetMetaDataDictionaryArray(&outputArray);
  try
  {
    seriesWriter->Update();
  }
  catch (itk::ExceptionObject & excp)
  {
    std::cerr << "Exception thrown while writing the series " << std::endl;
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
  }
  std::cout << "The output series in directory " << argv[2] << " has " <<␣
↪outputSize[2] << " files with spacing "
            << outputSpacing << std::endl;
  return EXIT_SUCCESS;
}

void
CopyDictionary(itk::MetaDataDictionary & fromDict, itk::MetaDataDictionary & toDict)
{
  using DictionaryType = itk::MetaDataDictionary;

  DictionaryType::ConstIterator itr = fromDict.Begin();
  DictionaryType::ConstIterator end = fromDict.End();
  using MetaDataStringType = itk::MetaDataObject<std::string>;

  while (itr != end)
  {
    itk::MetaDataObjectBase::Pointer entry = itr->second;

    MetaDataStringType::Pointer entryvalue = dynamic_cast<MetaDataStringType *>(entry.
↪GetPointer());
    if (entryvalue)
    {
      std::string tagkey = itr->first;
      std::string tagvalue = entryvalue->GetMetaDataObjectValue();
      itk::EncapsulateMetaData<std::string>(toDict, tagkey, tagvalue);
    }
    ++itr;
  }
}
```

## Classes demonstrated

**class GDCMImageIO** : **public** itk::*ImageIOBase*

ImageIO class for reading and writing DICOM V3.0 and ACR/NEMA 1&2 uncompressed images. This class is only an adaptor to the GDCM library.

GDCM can be found at: http://sourceforge.net/projects/gdcm

To learn more about the revision shipped with ITK, call

git log Modules/ThirdParty/GDCM/src/

from an ITK Git checkout.

The compressors supported include "JPEG2000" (default), and "JPEG". The compression level parameter is not supported.

**Warning** There are several restrictions to this current writer:

- Even though during the writing process you pass in a DICOM file as input The output file may not contains ALL DICOM field from the input file. In particular:
  - The SeQuence DICOM field (SQ).
  - Fields from Private Dictionary.
- Some very long (>0xfff) binary fields are not loaded (typically 0029|0010), you need to explicitly set the maximum length of elements to load to be bigger (see Get/SetMaxSizeLoadEntry).
- In DICOM some fields are stored directly using their binary representation. When loaded into the MetaDataDictionary some fields are converted to ASCII (only VR: OB/OW/OF and UN are encoded as mime64).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Resample DICOM Series
- Read DICOM Series and Write 3D Image

See itk::GDCMImageIO for additional documentation.

### 3.6.2 ImageBase

#### Convert File Formats

#### Synopsis

Convert from one image file format to another. ITK will automatically detect the file type of the input and output files by their extension, and an *itk::ImageFileReader* and *itk::ImageFileWriter* will use the appropriate *ImageIO* class.

#### Results

#### Code

#### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
```

(continues on next page)

Fig. 337: Gourds.png

Fig. 338: Gourds.jpg

(continued from previous page)

```
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(reader->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**, typename **ConvertPixelTraits** = DefaultConvertPixelTraits<**typename** *TOutputImage*::
**class ImageFileReader** : **public** itk::*ImageSource*<*TOutputImage*>

Data source that reads image data from a single file.

This source object is a general filter to read data from a variety of file formats. It works with a ImageIOBase subclass to actually do the reading of the data. Object factory machinery can be used to automatically create the ImageIOBase, or the ImageIOBase can be manually created and set. Note that this class reads data from a single file; if you wish to read data from a series of files use ImageSeriesReader.

TOutputImage is the type expected by the external users of the filter. If data stored in the file is stored in a different format then specified by TOutputImage, than this filter converts data between the file type and the external expected type. The `ConvertPixelTraits` template parameter is used to do the conversion.

A Pluggable factory pattern is used this allows different kinds of readers to be registered (even at run time) without having to modify the code in this class. Normally just setting the FileName with the appropriate suffix is enough to get the reader to instantiate the correct ImageIO and read the file properly. However, some files (like raw binary format) have no accepted suffix, so you will have to manually create the ImageIO instance of the write type.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Write Vector Image
- Read Unknown Image Type
- Read An Image

See itk::ImageFileReader for additional documentation.

template<typename **TInputImage**>
**class ImageFileWriter** : **public** itk::*ProcessObject*

Writes image data to a single file.

ImageFileWriter writes its input data to a single output file. ImageFileWriter interfaces with an ImageIO class to write out the data. If you wish to write data into a series of files (e.g., a slice per file) use ImageSeriesWriter.

A pluggable factory pattern is used that allows different kinds of writers to be registered (even at run time) without having to modify the code in this class. You can either manually instantiate the ImageIO object and associate it with the ImageFileWriter, or let the class figure it out from the extension. Normally just setting the filename with a suitable suffix (".png", ".jpg", etc) and setting the input to the writer is enough to get the writer to work properly.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Write An image

See itk::ImageFileWriter for additional documentation.

### Convert Image to Another Type

### Synopsis

Converts an image to another type of image.

### Results

### Code

### C++

```
#include "itkRGBPixel.h"
#include "itkRGBAPixel.h"
#include "itkImage.h"
#include "itkDefaultConvertPixelTraits.h"
#include "itkConvertPixelBuffer.h"
```

(continues on next page)

(continued from previous page)

```
#include "itkImageRandomConstIteratorWithIndex.h"

int
main(int, char *[])
{
  constexpr int xDimension = 200;
  constexpr int yDimension = 100;

  using ComponentType = unsigned char;
  using RGBPixelType = itk::RGBPixel<ComponentType>;
  using RGBAPixelType = itk::RGBAPixel<ComponentType>;
  using RGBImageType = itk::Image<RGBPixelType, 2>;
  using RGBAImageType = itk::Image<RGBAPixelType, 2>;
  using TraitsType = itk::DefaultConvertPixelTraits<RGBPixelType>;
  using RGBAConverterType = itk::ConvertPixelBuffer<ComponentType, RGBPixelType,␣
↪TraitsType>;

  RGBImageType::Pointer  rgbImg = RGBImageType::New();
  RGBAImageType::Pointer rgbaImg = RGBAImageType::New();

  // Create the two images
  // RGBAImage
  RGBAImageType::IndexType rgbaStart;
  rgbaStart[0] = 0;
  rgbaStart[1] = 0;

  RGBAImageType::SizeType rgbaSize;
  rgbaSize[0] = xDimension;
  rgbaSize[1] = yDimension;

  itk::ImageRegion<2> rgbaRegion(rgbaStart, rgbaSize);
  rgbaImg->SetRegions(rgbaRegion);
  rgbaImg->Allocate();

  RGBAPixelType rgbaDefault;
  rgbaDefault[0] = 127;
  rgbaDefault[1] = 100;
  rgbaDefault[2] = 230;
  rgbaDefault[3] = 255;
  rgbaImg->FillBuffer(rgbaDefault);

  // RGBImage
  itk::ImageRegion<2> rgbRegion = rgbaRegion;
  rgbImg->SetRegions(rgbRegion);
  rgbImg->Allocate();

  size_t numberOfPixels = rgbImg->GetLargestPossibleRegion().GetNumberOfPixels();

  // Convert a raw buffer to a buffer of pixel types
  RGBAConverterType::Convert(rgbaImg->GetBufferPointer()->GetDataPointer(),
                             rgbaImg->GetNumberOfComponentsPerPixel(),
                             rgbImg->GetBufferPointer(),
                             numberOfPixels);

  // Check a few random values
  itk::ImageRandomConstIteratorWithIndex<RGBImageType> rgbIterator(rgbImg, rgbImg->
↪GetLargestPossibleRegion());
```

(continues on next page)

(continued from previous page)

```
  rgbIterator.SetNumberOfSamples(numberOfPixels / 10);
  rgbIterator.GoToBegin();

  while (!rgbIterator.IsAtEnd())
  {
    if (rgbImg->GetPixel(rgbIterator.GetIndex())[0] != rgbaImg->GetPixel(rgbIterator.
↪GetIndex())[0] ||

        rgbImg->GetPixel(rgbIterator.GetIndex())[1] != rgbaImg->GetPixel(rgbIterator.
↪GetIndex())[1] ||

        rgbImg->GetPixel(rgbIterator.GetIndex())[2] != rgbaImg->GetPixel(rgbIterator.
↪GetIndex())[2])
    {
      std::cout << "Copy failed for index " << rgbIterator.GetIndex() << " got "
                << rgbImg->GetPixel(rgbIterator.GetIndex()) << " but expected "
                << rgbaImg->GetPixel(rgbIterator.GetIndex()) << std::endl;
    }
    ++rgbIterator;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **InputPixelType**, typename **OutputPixelType**, typename **OutputConvertTraits**>
**class ConvertPixelBuffer**

Class to convert blocks of data from one type to another.

ConvertPixelBuffer has a static method Convert(). It is used by itkImageFileReader to convert from an unknown type at run-time to the compile-time template pixel type in itkImageFileReader. To work with complex pixel types like RGB and RGBA a traits class is used. OutputConvertTraits() is the traits class. The default one used is DefaultConvertPixelTraits.

Derived from ConvertPixelBuffer has a static method Convert(). It is used to work with pixel type as VariableLengthVector type.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Convert Image To Another Type

Derived from ConvertPixelBuffer has a static method Convert(). It is used to work with pixel type as Array type.

See itk::ConvertPixelBuffer for additional documentation.

### Create 3D Volume From Series of 2D Images

### Synopsis

Create a 3D volume from a series of 2D images.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkImageSeriesReader.h"
#include "itkImageFileWriter.h"
#include "itkNumericSeriesFileNames.h"

int
main(int argc, char * argv[])
{
  // Verify the number of parameters in the command line
  if (argc < 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " pattern firstSliceValue lastSliceValue outputImageFile"
↪<< std::endl;
    return EXIT_FAILURE;
  }

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 3;

  using ImageType = itk::Image<PixelType, Dimension>;
  using ReaderType = itk::ImageSeriesReader<ImageType>;
  using WriterType = itk::ImageFileWriter<ImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  WriterType::Pointer writer = WriterType::New();

  const unsigned int first = std::stoi(argv[2]);
  const unsigned int last = std::stoi(argv[3]);

  const char * outputFilename = argv[4];

  using NameGeneratorType = itk::NumericSeriesFileNames;

  NameGeneratorType::Pointer nameGenerator = NameGeneratorType::New();
```

(continues on next page)

(continued from previous page)

```
  nameGenerator->SetSeriesFormat(argv[1]);

  nameGenerator->SetStartIndex(first);
  nameGenerator->SetEndIndex(last);
  nameGenerator->SetIncrementIndex(1);
  std::vector<std::string> names = nameGenerator->GetFileNames();

  // List the files
  //
  std::vector<std::string>::iterator nit;
  for (nit = names.begin(); nit != names.end(); nit++)
  {
    std::cout << "File: " << (*nit).c_str() << std::endl;
  }

  reader->SetFileNames(names);

  writer->SetFileName(outputFilename);
  writer->SetInput(reader->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**>
**class ImageSeriesReader** : **public** itk::*ImageSource<TOutputImage>*

Data source that reads image data from a series of disk files.

This class builds an n-dimension image from multiple n-1 dimension image files. The files stored in a vector of strings are read using the ImageFileReader. File format may vary between the files, but the image data must have the same Size for all dimensions.

**See** GDCMSeriesFileNames

**See** NumericSeriesFileNames

See itk::ImageSeriesReader for additional documentation.

### Create a List of File Names

### Synopsis

Create a list of numbered file names.

### Results

Output:

```
output_0.png
output_10.png
output_20.png
output_30.png
output_40.png
output_50.png
output_60.png
output_70.png
output_80.png
output_90.png
output_100.png
output_110.png
output_120.png
output_130.png
output_140.png
output_150.png

***************

output_0000.png
output_0010.png
output_0020.png
output_0030.png
output_0040.png
output_0050.png
output_0060.png
output_0070.png
output_0080.png
output_0090.png
output_0100.png
output_0110.png
output_0120.png
output_0130.png
output_0140.png
output_0150.png
```

### Code

### Python

```python
#!/usr/bin/env python

import itk

numeric_series_file_names = itk.NumericSeriesFileNames.New(
    start_index=0, end_index=150, increment_index=10, series_format="output_%d.png"
)

file_names = numeric_series_file_names.GetFileNames()

for file_name in file_names:
    print(file_name)

print()
print("***************")
print()

numeric_series_file_names.SetSeriesFormat("output_%04d.png")

file_names = numeric_series_file_names.GetFileNames()

for file_name in file_names:
    print(file_name)
```

### C++

```cpp
#include "itkNumericSeriesFileNames.h"

int
main(int, char *[])
{
  itk::NumericSeriesFileNames::Pointer numericSeriesFileNames = itk::
↪NumericSeriesFileNames::New();
  numericSeriesFileNames->SetStartIndex(0);
  numericSeriesFileNames->SetEndIndex(150);
  numericSeriesFileNames->SetIncrementIndex(10);
  numericSeriesFileNames->SetSeriesFormat("output_%d.png");

  std::vector<std::string> fileNames = numericSeriesFileNames->GetFileNames();

  for (const auto & fileName : fileNames)
  {
    std::cout << fileName << std::endl;
  }

  std::cout << std::endl;
  std::cout << "***************" << std::endl;
  std::cout << std::endl;

  numericSeriesFileNames->SetSeriesFormat("output_%04d.png");
```

(continues on next page)

(continued from previous page)

```
  fileNames = numericSeriesFileNames->GetFileNames();

  for (const auto & fileName : fileNames)
  {
    std::cout << fileName << std::endl;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class NumericSeriesFileNames** : **public** itk::*Object*

Generate an ordered sequence of filenames.

This class generate an ordered sequence of files whose filenames contain a single unique, non-negative, integral value (e.g. test.1.png, test2.png, foo.3, etc.).

The file name is created from a sprintf-style series format which should contain an integer format string like "%d". Bad formats will cause the series reader to throw an exception.

Warning: returned filenames (which may be full or relative paths) are not checked against any system-imposed path-length limit, because of difficulties finding a portable method to do so.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A List Of File Names

See itk::NumericSeriesFileNames for additional documentation.

### Generate Slices From Volume

### Synopsis

Create series of 2D images from a given volume.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkNumericSeriesFileNames.h"
#include "itkImageSeriesWriter.h"
```

(continues on next page)

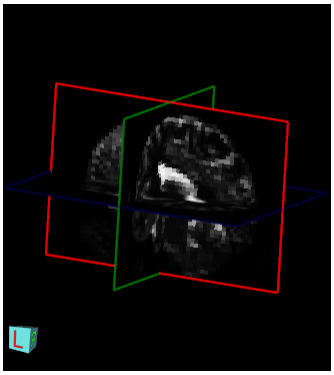


Fig. 339: Input image

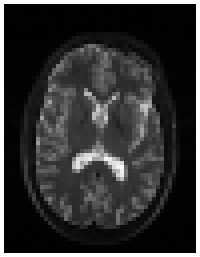


Fig. 340: Output image, slice 20

Fig. 341: Output image, slice 30

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  std::string format = std::string(outputFileName) + std::string("-%d.png");

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using InputImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using OutputPixelType = unsigned char;
  using RescaleImageType = itk::Image<OutputPixelType, Dimension>;

  using RescaleFilterType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪RescaleImageType>;
  RescaleFilterType::Pointer rescale = RescaleFilterType::New();
  rescale->SetInput(reader->GetOutput());
  rescale->SetOutputMinimum(0);
  rescale->SetOutputMaximum(255);
  rescale->UpdateLargestPossibleRegion();

  InputImageType::RegionType region = reader->GetOutput()->GetLargestPossibleRegion();
  InputImageType::SizeType   size = region.GetSize();

  itk::NumericSeriesFileNames::Pointer fnames = itk::NumericSeriesFileNames::New();
  fnames->SetStartIndex(0);
  fnames->SetEndIndex(size[2] - 1);
  fnames->SetIncrementIndex(1);
  fnames->SetSeriesFormat(format.c_str());

  using OutputImageType = itk::Image<OutputPixelType, 2>;

  using WriterType = itk::ImageSeriesWriter<RescaleImageType, OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(rescale->GetOutput());
  writer->SetFileNames(fnames->GetFileNames());

  try
  {
    writer->Update();
  }
```

(continues on next page)

(continued from previous page)

```
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

```
#!/usr/bin/env python
import itk
import argparse

itk.auto_progress(2)

parser = argparse.ArgumentParser(description="Generate Slices From Volume.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("extension", nargs="?")
args = parser.parse_args()

if args.extension:
    extension = args.extension
else:
    extension = ".png"

fileNameFormat = args.output_image + "-%d" + extension

Dimension = 3

PixelType = itk.UC
InputImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)

OutputPixelType = itk.UC
RescaleImageType = itk.Image[OutputPixelType, Dimension]

RescaleFilterType = itk.RescaleIntensityImageFilter[InputImageType, RescaleImageType]
rescale = RescaleFilterType.New()
rescale.SetInput(reader.GetOutput())
rescale.SetOutputMinimum(0)
rescale.SetOutputMaximum(255)
rescale.UpdateLargestPossibleRegion()

region = reader.GetOutput().GetLargestPossibleRegion()
size = region.GetSize()

fnames = itk.NumericSeriesFileNames.New()
fnames.SetStartIndex(0)
fnames.SetEndIndex(size[2] - 1)
fnames.SetIncrementIndex(1)
fnames.SetSeriesFormat(fileNameFormat)
```

(continues on next page)

(continued from previous page)

```
OutputImageType = itk.Image[OutputPixelType, 2]

WriterType = itk.ImageSeriesWriter[RescaleImageType, OutputImageType]
writer = WriterType.New()
writer.SetInput(rescale.GetOutput())
writer.SetFileNames(fnames.GetFileNames())

writer.Update()
```

### Classes demonstrated

**class NumericSeriesFileNames** : **public** itk::*Object*
Generate an ordered sequence of filenames.

This class generate an ordered sequence of files whose filenames contain a single unique, non-negative, integral value (e.g. test.1.png, test2.png, foo.3, etc.).

The file name is created from a sprintf-style series format which should contain an integer format string like "%d". Bad formats will cause the series reader to throw an exception.

Warning: returned filenames (which may be full or relative paths) are not checked against any system-imposed path-length limit, because of difficulties finding a portable method to do so.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create A List Of File Names

See itk::NumericSeriesFileNames for additional documentation.

template<typename **TInputImage**, typename **TOutputImage**>
**class ImageSeriesWriter** : **public** itk::*ProcessObject*
Writes image data to a series of data files.

ImageSeriesWriter writes its input data to a series of output files. The writer is templated over an input image type and an output image type. Usually, the output image type will have fewer dimensions than the input image type. Each file has a name created using the SeriesFormat. This string is used as a sprintf argument to build a filename. The string should contain zero or one "%d" or equivalent. The "%d" is an incremental file number that starts at StartIndex and is incremented by IncrementIndex. Since this writer uses an internal instance of an ImageFileWriter, the type of file is determined by either the file extension or an ImageIO class if specified.

**See** ImageFileWriter

**See** ImageIOBase

**See** ImageSeriesReader

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Creade 3D Volume From Series Of 2D Images

See itk::ImageSeriesWriter for additional documentation.

### Process Image Chunks

### Synopsis

Process an image in small chunks.

### Results

Fig. 342: Input image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMedianImageFilter.h"

int
```

(continues on next page)

Fig. 343: Output image

(continued from previous page)

```
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  const int xDiv = 6;
  const int yDiv = 4;
  const int zDiv = 1; // 1 for 2D input

  constexpr unsigned int Dimension = 3;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);
  reader->UpdateOutputInformation();
  ImageType::RegionType fullRegion = reader->GetOutput()->GetLargestPossibleRegion();
  ImageType::SizeType   fullSize = fullRegion.GetSize();
  // when reading image from file, start index is always 0

  ImageType::IndexType start;
  ImageType::IndexType end;
  ImageType::SizeType  size;

  using MedianType = itk::MedianImageFilter<ImageType, ImageType>;
  MedianType::Pointer median = MedianType::New();
  median->SetInput(reader->GetOutput());
  median->SetRadius(2);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);

  // Use for loops to split the image into chunks.
  // This way of splitting gives us easy control over it.
  // For example, we could make the regions always be of equal size
  // in order to create samples for a deep learning algorithm.
  // ImageRegionSplitterMultidimensional has a similar
  // functionality to what is implemented below.
  for (int z = 0; z < zDiv; z++)
  {
    start[2] = fullSize[2] * double(z) / zDiv;
    end[2] = fullSize[2] * (z + 1.0) / zDiv;
    size[2] = end[2] - start[2];

    for (int y = 0; y < yDiv; y++)
```

(continues on next page)

(continued from previous page)

```
    {
      start[1] = fullSize[1] * double(y) / yDiv;
      end[1] = fullSize[1] * (y + 1.0) / yDiv;
      size[1] = end[1] - start[1];

      for (int x = 0; x < xDiv; x++)
      {
        start[0] = fullSize[0] * double(x) / xDiv;
        end[0] = fullSize[0] * (x + 1.0) / xDiv;
        size[0] = end[0] - start[0];

        ImageType::RegionType region;
        region.SetIndex(start);
        region.SetSize(size);

        // now that we have our chunk, request the filter to only process that
        median->GetOutput()->SetRequestedRegion(region);
        median->Update();
        ImageType::Pointer result = median->GetOutput();

        // only needed in case of further manual processing
        result->DisconnectPipeline();

        // possible additional non-ITK pipeline processing

        writer->SetInput(result);

        // convert region into IO region
        itk::ImageIORegion ioRegion(Dimension);
        ioRegion.SetIndex(0, start[0]);
        ioRegion.SetIndex(1, start[1]);
        ioRegion.SetIndex(2, start[2]);
        ioRegion.SetSize(0, region.GetSize()[0]);
        ioRegion.SetSize(1, region.GetSize()[1]);
        ioRegion.SetSize(2, region.GetSize()[2]);
        // tell the writer this is only a chunk of a larger image
        writer->SetIORegion(ioRegion);

        try
        {
          writer->Update();
        }
        catch (itk::ExceptionObject & error)
        {
          std::cerr << "Exception for chunk: " << x << ' ' << y << ' ' << z << error <
↪< std::endl;
          return EXIT_FAILURE;
        }
      }
    }
  }

  return EXIT_SUCCESS;
}
```

```
#!/usr/bin/env python
```

(continues on next page)

(continued from previous page)

```
import itk
import argparse

itk.auto_progress(2)

parser = argparse.ArgumentParser(description="Process Image Chunks.")
parser.add_argument("input_image")
parser.add_argument("output_image")
args = parser.parse_args()

xDiv = 6
yDiv = 4
zDiv = 1
# 1 for 2D input

Dimension = 3

PixelType = itk.UC
ImageType = itk.Image[PixelType, Dimension]

ReaderType = itk.ImageFileReader[ImageType]
reader = ReaderType.New()
reader.SetFileName(args.input_image)
reader.UpdateOutputInformation()
fullRegion = reader.GetOutput().GetLargestPossibleRegion()
fullSize = fullRegion.GetSize()
# when reading image from file, start index is always 0

# create variables of the proper types
start = itk.Index[Dimension]()
end = itk.Index[Dimension]()
size = itk.Size[Dimension]()

MedianType = itk.MedianImageFilter[ImageType, ImageType]
median = MedianType.New()
median.SetInput(reader.GetOutput())
median.SetRadius(2)

WriterType = itk.ImageFileWriter[ImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)

# Use for loops to split the image into chunks.
# This way of splitting gives us easy control over it.
# For example, we could make the regions always be of equal size
# in order to create samples for a deep learning algorithm.
# ImageRegionSplitterMultidimensional has a similar
# functionality to what is implemented below.
for z in range(zDiv):
    start[2] = int(fullSize[2] * float(z) / zDiv)
    end[2] = int(fullSize[2] * (z + 1.0) / zDiv)
    size[2] = end[2] - start[2]

    for y in range(yDiv):
        start[1] = int(fullSize[1] * float(y) / yDiv)
        end[1] = int(fullSize[1] * (y + 1.0) / yDiv)
```

(continues on next page)

(continued from previous page)

```
        size[1] = end[1] - start[1]

        for x in range(xDiv):
            start[0] = int(fullSize[0] * float(x) / xDiv)
            end[0] = int(fullSize[0] * (x + 1.0) / xDiv)
            size[0] = end[0] - start[0]

            region = itk.ImageRegion[Dimension]()
            region.SetIndex(start)
            region.SetSize(size)

            # now that we have our chunk, request the filter to only process that
            median.GetOutput().SetRequestedRegion(region)
            median.Update()
            result = median.GetOutput()

            # only needed in case of further manual processing
            result.DisconnectPipeline()

            # possible additional non-ITK pipeline processing

            writer.SetInput(result)

            # convert region into IO region
            ioRegion = itk.ImageIORegion(Dimension)
            ioRegion.SetIndex(0, start[0])
            ioRegion.SetIndex(1, start[1])
            ioRegion.SetIndex(2, start[2])
            ioRegion.SetSize(0, region.GetSize()[0])
            ioRegion.SetSize(1, region.GetSize()[1])
            ioRegion.SetSize(2, region.GetSize()[2])
            # tell the writer this is only a chunk of a larger image
            writer.SetIORegion(ioRegion)

            try:
                writer.Update()
            except Exception as e:
                print("Exception for chunk:", x, y, z, "\n", e)
                sys.exit(1)
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MedianImageFilter** : **public** itk::BoxImageFilter<*TInputImage*, *TOutputImage*>

Applies a median filter to an image.

Computes an image where a given pixel is the median value of the the pixels in a neighborhood about the corresponding input pixel.

A median filter is one of the family of nonlinear filters. It is used to smooth an image without being biased by outliers or shot noise.

This filter requires that the input pixel type provides an operator<() (LessThan Comparable).

**See** Image

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Median Filter Of An Image
- Median Filter Of An RGB Image

See itk::MedianImageFilter for additional documentation.

template<typename **TInputImage**>
**class ImageFileWriter** : **public** itk::*ProcessObject*

Writes image data to a single file.

ImageFileWriter writes its input data to a single output file. ImageFileWriter interfaces with an ImageIO class to write out the data. If you wish to write data into a series of files (e.g., a slice per file) use ImageSeriesWriter.

A pluggable factory pattern is used that allows different kinds of writers to be registered (even at run time) without having to modify the code in this class. You can either manually instantiate the ImageIO object and associate it with the ImageFileWriter, or let the class figure it out from the extension. Normally just setting the filename with a suitable suffix (".png", ".jpg", etc) and setting the input to the writer is enough to get the writer to work properly.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Write An image

See itk::ImageFileWriter for additional documentation.

### Read an Image

### Synopsis

This examples demonstrates how to read an image file into an itk::Image. The file type is determined by the extension of the specified filename.

### Results

NA

### Code

### Python

```python
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(description="Read An Image.")
parser.add_argument("input_image")
args = parser.parse_args()

image = itk.imread(args.input_image)
```

### C++

```cpp
#include "itkImage.h"
#include "itkImageFileReader.h"

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  ImageType::Pointer image = reader->GetOutput();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**, typename **ConvertPixelTraits** = DefaultConvertPixelTraits<**typename** *TOutputImage*::
**class ImageFileReader** : **public** itk::*ImageSource*<*TOutputImage*>

Data source that reads image data from a single file.

This source object is a general filter to read data from a variety of file formats. It works with a ImageIOBase subclass to actually do the reading of the data. Object factory machinery can be used to automatically create the ImageIOBase, or the ImageIOBase can be manually created and set. Note that this class reads data from a single file; if you wish to read data from a series of files use ImageSeriesReader.

TOutputImage is the type expected by the external users of the filter. If data stored in the file is stored in a different format then specified by TOutputImage, than this filter converts data between the file type and the external expected type. The `ConvertPixelTraits` template parameter is used to do the conversion.

A Pluggable factory pattern is used this allows different kinds of readers to be registered (even at run time) without having to modify the code in this class. Normally just setting the FileName with the appropriate suffix is enough to get the reader to instantiate the correct ImageIO and read the file properly. However, some files (like raw binary format) have no accepted suffix, so you will have to manually create the ImageIO instance of the write type.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Write Vector Image
- Read Unknown Image Type
- Read An Image

See itk::ImageFileReader for additional documentation.

### Read Unknown Image Type

### Synopsis

Read one unknown image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImageIOBase.h"

template <class TImage>
int
ReadImage(const char * fileName, typename TImage::Pointer image)
```

(continues on next page)

(continued from previous page)

```
{
  using ImageType = TImage;
  using ImageReaderType = itk::ImageFileReader<ImageType>;

  typename ImageReaderType::Pointer reader = ImageReaderType::New();
  reader->SetFileName(fileName);

  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & e)
  {
    std::cerr << e.what() << std::endl;
    return EXIT_FAILURE;
  }

  image->Graft(reader->GetOutput());

  return EXIT_SUCCESS;
}

template <unsigned int VDimension>
int
ReadScalarImage(const char * inputFileName, const itk::IOComponentEnum componentType)
{
  switch (componentType)
  {
    default:
    case itk::IOComponentEnum::UNKNOWNCOMPONENTTYPE:
      std::cerr << "Unknown and unsupported component type!" << std::endl;
      return EXIT_FAILURE;

    case itk::IOComponentEnum::UCHAR:
    {
      using PixelType = unsigned char;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }

      std::cout << image << std::endl;
      break;
    }

    case itk::IOComponentEnum::CHAR:
    {
      using PixelType = char;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
```

(continues on next page)

(continued from previous page)

```
    {
      return EXIT_FAILURE;
    }

    std::cout << image << std::endl;
    break;
  }

  case itk::IOComponentEnum::USHORT:
  {
    using PixelType = unsigned short;
    using ImageType = itk::Image<PixelType, VDimension>;

    typename ImageType::Pointer image = ImageType::New();

    if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
    {
      return EXIT_FAILURE;
    }

    std::cout << image << std::endl;
    break;
  }

  case itk::IOComponentEnum::SHORT:
  {
    using PixelType = short;
    using ImageType = itk::Image<PixelType, VDimension>;

    typename ImageType::Pointer image = ImageType::New();

    if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
    {
      return EXIT_FAILURE;
    }

    std::cout << image << std::endl;
    break;
  }

  case itk::IOComponentEnum::UINT:
  {
    using PixelType = unsigned int;
    using ImageType = itk::Image<PixelType, VDimension>;

    typename ImageType::Pointer image = ImageType::New();

    if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
    {
      return EXIT_FAILURE;
    }

    std::cout << image << std::endl;
    break;
  }

  case itk::IOComponentEnum::INT:
```

(continues on next page)

(continued from previous page)

```
    {
      using PixelType = int;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }

      std::cout << image << std::endl;
      break;
    }

    case itk::IOComponentEnum::ULONG:
    {
      using PixelType = unsigned long;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }

      std::cout << image << std::endl;
      break;
    }

    case itk::IOComponentEnum::LONG:
    {
      using PixelType = long;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }

      std::cout << image << std::endl;
      break;
    }

    case itk::IOComponentEnum::FLOAT:
    {
      using PixelType = float;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
      }

      std::cout << image << std::endl;
      break;
    }

    case itk::IOComponentEnum::DOUBLE:
    {
      using PixelType = double;
      using ImageType = itk::Image<PixelType, VDimension>;

      typename ImageType::Pointer image = ImageType::New();

      if (ReadImage<ImageType>(inputFileName, image) == EXIT_FAILURE)
      {
        return EXIT_FAILURE;
      }

      std::cout << image << std::endl;
      break;
    }
  }
  return EXIT_SUCCESS;
}

int
main(int argc, char * argv[])
{
  if (argc != 2)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];

  itk::ImageIOBase::Pointer imageIO =
    itk::ImageIOFactory::CreateImageIO(inputFileName, itk::CommonEnums::IOFileMode::
↪ReadMode);

  imageIO->SetFileName(inputFileName);
  imageIO->ReadImageInformation();

  using IOPixelType = itk::IOPixelEnum;
  const IOPixelType pixelType = imageIO->GetPixelType();

  std::cout << "Pixel Type is " << itk::ImageIOBase::GetPixelTypeAsString(pixelType) <
↪< std::endl;

  using IOComponentType = itk::IOComponentEnum;
  const IOComponentType componentType = imageIO->GetComponentType();

  std::cout << "Component Type is " << imageIO->
↪GetComponentTypeAsString(componentType) << std::endl;
```

(continues on next page)

(continued from previous page)

```
  const unsigned int imageDimension = imageIO->GetNumberOfDimensions();

  std::cout << "Image Dimension is " << imageDimension << std::endl;

  switch (pixelType)
  {
    case itk::IOPixelEnum::SCALAR:
    {
      if (imageDimension == 2)
      {
        return ReadScalarImage<2>(inputFileName, componentType);
      }
      else if (imageDimension == 3)
      {
        return ReadScalarImage<3>(inputFileName, componentType);
      }
      else if (imageDimension == 4)
      {
        return ReadScalarImage<4>(inputFileName, componentType);
      }
    }

    default:
      std::cerr << "not implemented yet!" << std::endl;
      return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class ImageIOBase** : **public** itk::LightProcessObject

Abstract superclass defines image IO interface.

ImageIOBase is a class that reads and/or writes image data of a particular format (such as PNG or raw binary). The ImageIOBase encapsulates both the reading and writing of data. The ImageIOBase is used by the ImageFileReader class (to read data) and the ImageFileWriter (to write data) into a single file. The ImageSeriesReader and ImageSeriesWriter classes are used to read and write data (in conjunction with ImageIOBase) when the data is represented by a series of files. Normally the user does not directly manipulate this class other than to instantiate it, set the FileName, and assign it to a ImageFileReader/ImageFileWriter or ImageSeriesReader/ImageSeriesWriter.

A Pluggable factory pattern is used this allows different kinds of readers to be registered (even at run time) without having to modify the code in this class.

**See** ImageFileWriter

**See** ImageFileReader

**See** ImageSeriesWriter

**See** ImageSeriesReader

Subclassed by itk::BioRadImageIO, itk::BMPImageIO, itk::Bruker2dseqImageIO, itk::DCMTKImageIO, itk::GDCMImageIO, itk::GiplImageIO, itk::IPLCommonImageIO, itk::JPEGImageIO, itk::MetaImageIO,

itk::MINCImageIO, itk::NiftiImageIO, itk::NrrdImageIO, itk::PNGImageIO, itk::RawImageIO< TPixel, VImageDimension >, itk::StimulateImageIO, itk::StreamingImageIOBase, itk::TIFFImageIO, itk::VideoIOBase, itk::VoxBoCUBImageIO

See itk::ImageIOBase for additional documentation.

### Register IO Factories

### Synopsis

When CMake is not used to build an executable or library against ITK, the Image and Transform IO format objects are not automatically registered to ITK's object factory system, so they must be manually registered. There is not a static list of IO classes because the classes available depend on which modules are enabled when ITK is configured. This examples shows how to register the ImageIOBase objects so the ImageFileReader can read the formats corresponding to the registered objects.

### Results

```
When CMake is not used to register the IO classes, there are
0 IO objects available to the ImageFileReader.

When we try to read a MetaImage, we will fail.

After registering the MetaImageIO object, there are
1 IO objects available to the ImageFileReader.

Now, when we try to read a MetaImage, we will succeed.

Every format desired to be supported by the reader
must be registered.
```

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkMetaImageIOFactory.h"
#include "itkPNGImageIOFactory.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <MetaImageFileName> <PNGFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * metaImageFileName = argv[1];
```

(continues on next page)

(continued from previous page)

```
  const char * pngFileName = argv[2];

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();

  using RegisteredObjectsContainerType = std::list<itk::LightObject::Pointer>;


  RegisteredObjectsContainerType registeredIOs = itk::ObjectFactoryBase::
↪CreateAllInstance("itkImageIOBase");
  std::cout << "When CMake is not used to register the IO classes, there are\n"
            << registeredIOs.size() << " IO objects available to the ImageFileReader.\
↪n"
            << std::endl;


  std::cout << "When we try to read a MetaImage, we will ";
  reader->SetFileName(metaImageFileName);
  try
  {
    reader->Update();
  }
  catch (itk::ImageFileReaderException &)
  {
    std::cout << "fail.\n" << std::endl;
  }


  std::cout << "After registering the MetaImageIO object, ";
  itk::MetaImageIOFactory::RegisterOneFactory();

  std::cout << "there are\n";
  registeredIOs = itk::ObjectFactoryBase::CreateAllInstance("itkImageIOBase");
  std::cout << registeredIOs.size() << " IO objects available to the ImageFileReader.\
↪n" << std::endl;

  std::cout << "Now, when we try to read a MetaImage, we will ";
  try
  {
    reader->Update();
    std::cout << "succeed.\n" << std::endl;
  }
  catch (itk::ImageFileReaderException & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }


  std::cout << "Every format desired to be supported by the reader\n"
            << "must be registered." << std::endl;
  itk::PNGImageIOFactory::RegisterOneFactory();
```

(continues on next page)

(continued from previous page)

```
  reader->SetFileName(pngFileName);
  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }


  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputImage**, typename **ConvertPixelTraits** = DefaultConvertPixelTraits<**typename** *TOutputImage*::
**class ImageFileReader** : **public** itk::*ImageSource*<*TOutputImage*>
Data source that reads image data from a single file.

This source object is a general filter to read data from a variety of file formats. It works with a ImageIOBase subclass to actually do the reading of the data. Object factory machinery can be used to automatically create the ImageIOBase, or the ImageIOBase can be manually created and set. Note that this class reads data from a single file; if you wish to read data from a series of files use ImageSeriesReader.

TOutputImage is the type expected by the external users of the filter. If data stored in the file is stored in a different format then specified by TOutputImage, than this filter converts data between the file type and the external expected type. The `ConvertPixelTraits` template parameter is used to do the conversion.

A Pluggable factory pattern is used this allows different kinds of readers to be registered (even at run time) without having to modify the code in this class. Normally just setting the FileName with the appropriate suffix is enough to get the reader to instantiate the correct ImageIO and read the file properly. However, some files (like raw binary format) have no accepted suffix, so you will have to manually create the ImageIO instance of the write type.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Write Vector Image
- Read Unknown Image Type
- Read An Image

See itk::ImageFileReader for additional documentation.

**class ObjectFactoryBase** : **public** itk::*Object*
Create instances of classes using an object factory.

ObjectFactoryBase is used to create itk objects. The base class ObjectFactoryBase contains a static method CreateInstance() that is used to create itk objects from the list of registerd ObjectFactoryBase sub-classes. The first time CreateInstance() is called, all dll's or shared libraries in the environment variable ITK_AUTOLOAD_PATH are loaded into the current process. The C function itkLoad is called on each dll. itkLoad should return an instance of the factory sub-class implemented in the shared library. ITK_AUTOLOAD_PATH is an environment variable containing a colon separated (semi-colon on win32) list of paths.

This can be use to override the creation of any object in ITK.

Subclassed by itk::BioRadImageIOFactory, itk::BMPImageIOFactory, itk::Bruker2dseqImageIOFactory, itk::BYUMeshIOFactory, itk::DCMTKImageIOFactory, itk::FileListVideoIOFactory, itk::FreeSurferAsciiMeshIOFactory, itk::FreeSurferBinaryMeshIOFactory, itk::GDCMImageIOFactory, itk::GE4ImageIOFactory, itk::GE5ImageIOFactory, itk::GEAdwImageIOFactory, itk::GiftiMeshIOFactory, itk::GiplImageIOFactory, itk::GPUBinaryThresholdImageFilterFactory, itk::GPUDemonsRegistrationFilterFactory, itk::GPUGradientAnisotropicDiffusionImageFilterFactory, itk::GPUImageFactory, itk::GPUMeanImageFilterFactory, itk::HDF5ImageIOFactory, itk::HDF5TransformIOFactory, itk::JPEG2000ImageIOFactory, itk::JPEGImageIOFactory, itk::LSMImageIOFactory, itk::MatlabTransformIOFactory, itk::MetaImageIOFactory, itk::MINCImageIOFactory, itk::MRCImageIOFactory, itk::NiftiImageIOFactory, itk::NrrdImageIOFactory, itk::ObjectFactory< T >, itk::OBJMeshIOFactory, itk::OFFMeshIOFactory, itk::OpenCVVideoIOFactory, itk::PNGImageIOFactory, itk::RawImageIOFactory< TPixel, VImageDimension >, itk::SiemensVisionImageIOFactory, itk::SpatialObjectFactoryBase, itk::StimulateImageIOFactory, itk::TIFFImageIOFactory, itk::TransformFactoryBase, itk::TxtTransformIOFactory, itk::VoxBoCUBImageIOFactory, itk::VTKImageIOFactory, itk::VTKPolyDataMeshIOFactory, itk::VXLVideoIOFactory

See itk::ObjectFactoryBase for additional documentation.

**class MetaImageIOFactory** : **public** itk::*ObjectFactoryBase*
Create instances of MetaImageIO objects using an object factory.

See itk::MetaImageIOFactory for additional documentation.

**class PNGImageIOFactory** : **public** itk::*ObjectFactoryBase*
Create instances of PNGImageIO objects using an object factory.

See itk::PNGImageIOFactory for additional documentation.

### Write an Image

### Synopsis

This example demonstrates how to write an itk::Image to a file. The file type is determined by the extension of the specified filename.

### Results

### Code

### Python

```
#!/usr/bin/env python

import itk
```

(continues on next page)

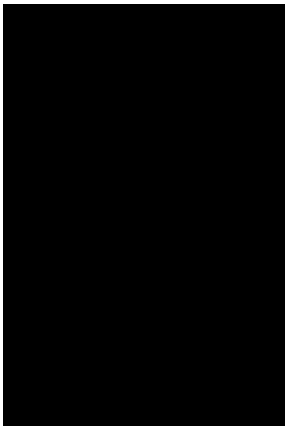


Fig. 344: test.png

(continued from previous page)

```
import argparse

parser = argparse.ArgumentParser(description="Write An Image.")
parser.add_argument("output_image", nargs="?")
args = parser.parse_args()

if args.output_image:
    output_filename = args.output_image
else:
    output_filename = "testPython.png"

pixel_type = itk.UC
dimension = 2
image_type = itk.Image[pixel_type, dimension]

start = itk.Index[dimension]()
start.Fill(0)

size = itk.Size[dimension]()
size[0] = 200
size[1] = 300

region = itk.ImageRegion[dimension]()
region.SetIndex(start)
region.SetSize(size)

image = image_type.New(Regions=region)
image.Allocate()

itk.imwrite(image, output_filename)
```

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"

#include <iostream>
#include <string>

int
main(int argc, char * argv[])
{
  std::string outputFilename;
  if (argc > 1)
  {
    outputFilename = argv[1];
  }
  else
  {
    outputFilename = "test.png";
  }

  using PixelType = unsigned char;
  constexpr unsigned int Dimension = 2;
  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->Allocate();

  ImageType::IndexType ind;
  ind[0] = 10;
  ind[1] = 10;

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFilename);
  writer->SetInput(image);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
```

(continues on next page)

(continued from previous page)

```
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**>
**class ImageFileWriter** : **public** itk::*ProcessObject*

Writes image data to a single file.

ImageFileWriter writes its input data to a single output file. ImageFileWriter interfaces with an ImageIO class to write out the data. If you wish to write data into a series of files (e.g., a slice per file) use ImageSeriesWriter.

A pluggable factory pattern is used that allows different kinds of writers to be registered (even at run time) without having to modify the code in this class. You can either manually instantiate the ImageIO object and associate it with the ImageFileWriter, or let the class figure it out from the extension. Normally just setting the filename with a suitable suffix (".png", ".jpg", etc) and setting the input to the writer is enough to get the writer to work properly.

**See** ImageSeriesReader

**See** ImageIOBase

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Write An image

See itk::ImageFileWriter for additional documentation.

## 3.6.3 Mesh

### Read Mesh

### Synopsis

Read a mesh and display the Euclidean distance in between 2 given vertices

### Results

Distance:

```
0.604093
```

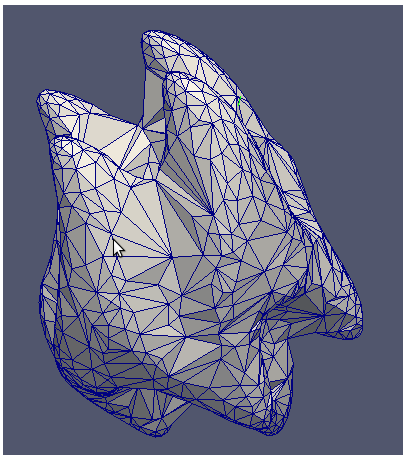


Fig. 345: Input mesh

## Code

### C++

```
#include "itkMesh.h"
#include "itkMeshFileReader.h"
#include "itkMeshFileWriter.h"

int
main(int argc, char * argv[])
{
  if ((argc != 2) && (argc != 4))
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <id 1 (optional)> <id 2 (optional)>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 3;

  using CoordinateType = double;
  using MeshType = itk::Mesh<CoordinateType, Dimension>;

  using ReaderType = itk::MeshFileReader<MeshType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);
  reader->Update();

  MeshType::Pointer mesh = reader->GetOutput();

  if (argc == 4)
  {
    MeshType::PointIdentifier id1 = std::stoi(argv[2]);
    MeshType::PointIdentifier id2 = std::stoi(argv[3]);
```

(continues on next page)

(continued from previous page)

```
    MeshType::PointType p = mesh->GetPoint(id1);
    MeshType::PointType q = mesh->GetPoint(id2);

    std::cout << p.EuclideanDistanceTo(q) << std::endl;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TOutputMesh**, typename **ConvertPointPixelTraits** = MeshConvertPixelTraits<**typename** *TOutputMe*
**class MeshFileReader** : **public** itk::MeshSource<*TOutputMesh*>

Mesh source that reads mesh data from a single file.

This source object is a general filter to read data from a variety of file formats. It works with a MeshIOBase subclass to actually do the reading of the data. Object factory machinery can be used to automatically create the MeshIOBase, or the MeshIOBase can be manually created and set.

TOutputMesh is the type expected by the external users of the filter. If data stored in the file is stored in a different format then specified by TOutputMesh, than this filter converts data between the file type and the external expected type. The ConvertTraits template argument is used to do the conversion.

A Pluggable factory pattern is used this allows different kinds of readers to be registered (even at run time) without having to modify the code in this class. Normally just setting the FileName with the appropriate suffix is enough to get the reader to instantiate the correct MeshIO and read the file properly. However, some files have no accepted suffix, so you will have to manually create the MeshIO instance of the write type.

**See** MeshIOBase

**Author** Wanlin Zhu. Uviversity of New South Wales, Australia.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Mesh

See itk::MeshFileReader for additional documentation.

## 3.6.4 TIFF

### Write a TIFF Image

### Synopsis

This example demonstrates how to explicitly specify the type of the image to write, regardless of the extension of the specified filename.

### Results

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkTIFFImageIO.h"

int
main(int argc, char * argv[])
{
  std::string outputFilename;
  if (argc > 1)
  {
    outputFilename = argv[1];
  }
  else
  {
    outputFilename = "test.tif";
  }

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;

  using ImageType = itk::Image<PixelType, Dimension>;

  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  ImageType::Pointer image = ImageType::New();
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 100)
    {
      imageIterator.Set(100);
    }
    else
    {
      imageIterator.Set(200);
```

(continues on next page)

(continued from previous page)

```
    }
    ++imageIterator;
  }

  using WriterType = itk::ImageFileWriter<ImageType>;
  using TIFFIOType = itk::TIFFImageIO;

  TIFFIOType::Pointer tiffIO = TIFFIOType::New();
  tiffIO->SetPixelType(itk::IOPixelEnum::SCALAR);

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFilename);
  writer->SetInput(image);
  writer->SetImageIO(tiffIO);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

. breathelink: itk::TIFFImageIO

### 3.6.5 TransformBase

#### Read Transform From File

#### Synopsis

Read a transform from a file.

#### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkTransformFileReader.h"
#include "itkTransformFactoryBase.h"

int
main(int argc, char * argv[])
{
  std::string fileName;
  if (argc == 1) // No arguments were provided
  {
    fileName = "test.tfm";
  }
  else
  {
    fileName = argv[1];
  }

  // Register default transforms
  itk::TransformFactoryBase::RegisterDefaultTransforms();

#if (ITK_VERSION_MAJOR == 4 && ITK_VERSION_MINOR >= 5) || ITK_VERSION_MAJOR > 4
  itk::TransformFileReaderTemplate<float>::Pointer reader = itk::
↪TransformFileReaderTemplate<float>::New();
#else
  itk::TransformFileReader::Pointer writer = itk::TransformFileReader::New();
#endif
  reader->SetFileName(fileName);
  reader->Update();

  // Display the transform
  std::cout << *(reader->GetTransformList()->begin()) << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TParametersValueType**>
**class TransformFileReaderTemplate** : **public** itk::LightProcessObject
TODO.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Transform From File

See itk::TransformFileReaderTemplate for additional documentation.

### Write Transform From File

### Synopsis

Write a transform from a file.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkVersion.h"

#include "itkRigid2DTransform.h"
#include "itkTransformFileWriter.h"

int
main(int argc, char * argv[])
{
  std::string fileName;
  if (argc == 1) // No arguments were provided
  {
    fileName = "test.tfm";
  }
  else
  {
    fileName = argv[1];
  }

  using TransformType = itk::Rigid2DTransform<float>;
  TransformType::Pointer transform = TransformType::New();

#if (ITK_VERSION_MAJOR == 4 && ITK_VERSION_MINOR >= 5) || ITK_VERSION_MAJOR > 4
  itk::TransformFileWriterTemplate<float>::Pointer writer = itk::
↪TransformFileWriterTemplate<float>::New();
#else
  itk::TransformFileWriter::Pointer writer = itk::TransformFileWriter::New();
#endif

  writer->SetInput(transform);
  writer->SetFileName(fileName);
  writer->Update();

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TParametersValueType**>
**class TransformFileReaderTemplate** : **public** itk::LightProcessObject
TODO.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Read Transform From File

See itk::TransformFileReaderTemplate for additional documentation.

### 3.6.6 TransformFactory

#### Register Transform With Transform Factory

#### Synopsis

Register a non-default transform with the transform factory.

#### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

#### Code

#### C++

```
#include "itkTransformFileReader.h"
#include "itkTransformFactoryBase.h"
#include "itkTransformFactory.h"
#include "itkMatrixOffsetTransformBase.h"

int
main(int argc, char * argv[])
{
  std::string fileName;
  if (argc == 1) // No arguments were provided
  {
    fileName = "test.tfm";
  }
  else
  {
    fileName = argv[1];
  }

  using MatrixOffsetTransformType = itk::MatrixOffsetTransformBase<double, 3, 3>;
  itk::TransformFactory<MatrixOffsetTransformType>::RegisterTransform();
```

(continues on next page)

(continued from previous page)

```
#if (ITK_VERSION_MAJOR == 4 && ITK_VERSION_MINOR >= 5) || ITK_VERSION_MAJOR > 4
  itk::TransformFileReaderTemplate<float>::Pointer reader = itk::
↪TransformFileReaderTemplate<float>::New();
#else
  itk::TransformFileReader::Pointer reader = itk::TransformFileReader::New();
#endif
  reader->SetFileName(fileName);
  reader->Update();

  std::cout << *(reader->GetTransformList()->begin()) << std::endl;

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **T**>
**class TransformFactory** : **public** itk::TransformFactoryBase
Create instances of Transforms.

/sphinx /sphinxexample{IO/TransformFactory/RegisterTransformWithTransformFactory,Register Transform With Transform Factory} /endsphinx

See itk::TransformFactory for additional documentation.

## 3.7 Numerics

### 3.7.1 Optimizers

#### Amoeba Optimizer

#### Synopsis

This will optimize a function using the AmoebaOptimizer class. This example demonstrates optimizing a simple paraboloid function.

#### Results

Output:

```
Position: [-5.003825599641884, 6.998563761340231]
Value: 5.00002
```

### Code

### C++

```
#include "itkAmoebaOptimizer.h"
#include "ExampleCostFunction.h"

namespace
{
// Typedef the optimizer and cost function, for convenience
using OptimizerType = itk::AmoebaOptimizer;
using CostType = itk::ExampleCostFunction2;
} // namespace

int
main(int, char *[])
{

  // Instantiate the optimizer
  OptimizerType::Pointer optimizer = OptimizerType::New();

  // Set properties pertinent to convergence
  optimizer->SetMaximumNumberOfIterations(100);
  optimizer->SetParametersConvergenceTolerance(0.01);
  optimizer->SetFunctionConvergenceTolerance(0.01);

  // Instantiate the cost function
  // The cost function is a 2D paraboloid in the x-y plane
  // with the equation f(x,y) = (x+5)^2+(y-7)^2 + 5
  // and a global minimum at (x,y) = (-5, 7)
  CostType::Pointer cost = CostType::New();

  // Assign the cost function to the optimizer
  optimizer->SetCostFunction(cost.GetPointer());

  // Set the initial parameters of the cost function
  OptimizerType::ParametersType initial(2);
  initial[0] = 123;
  initial[1] = -97.4;
  optimizer->SetInitialPosition(initial);

  // Begin the optimization!
  optimizer->StartOptimization();

  // Print out some information about the optimization
  std::cout << "Position: " << optimizer->GetCurrentPosition() << std::endl;
  std::cout << "Value: " << optimizer->GetValue() << std::endl;

  // As expected, the position is near to (-5, 7) and the value to 5
  // Position: [-5.003825599641884, 6.998563761340231]
  // Value: 5.00002
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class AmoebaOptimizer** : **public** itk::SingleValuedNonLinearVnlOptimizer

Wrap of the vnl_amoeba algorithm.

AmoebaOptimizer is a wrapper around the vnl_amoeba algorithm which is an implementation of the Nelder-Meade downhill simplex problem. For most problems, it is a few times slower than a Levenberg-Marquardt algorithm but does not require derivatives of its cost function. It works by creating a simplex (n+1 points in ND space). The cost function is evaluated at each corner of the simplex. The simplex is then modified (by reflecting a corner about the opposite edge, by shrinking the entire simplex, by contracting one edge of the simplex, or by expanding the simplex) in searching for the minimum of the cost function.

The methods AutomaticInitialSimplex() and SetInitialSimplexDelta() control whether the optimizer defines the initial simplex automatically (by constructing a very small simplex around the initial position) or uses a user supplied simplex size.

The method SetOptimizeWithRestarts() indicates that the amoeabe algorithm should be rerun after if converges. This heuristic increases the chances of escaping from a local optimum. Each time the simplex is initialized with the best solution obtained by the previous runs. The edge length is half of that from the previous iteration. The heuristic is terminated if the total number of iterations is greater-equal than the maximal number of iterations (SetMaximumNumberOfIterations) or the difference between the current function value and the best function value is less than a threshold (SetFunctionConvergenceTolerance) and max(|best_parameters_i - current_parameters_i|) is less than a threshold (SetParametersConvergenceTolerance).

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Amoeba Optimization

See itk::AmoebaOptimizer for additional documentation.

### Exhaustive Optimizer

### Visualize Parameter Space with Exhaustive Optimizer

ITK optimizers are commonly used to select suitable values for various parameters, such as choosing how to transform a moving image to register with a fixed image. A variety of image metrics and transform classes are available to guide the optimization process, each of which may employ parameters unique to its own implementation. It is often useful to visualize how changes in parameters will impact the metric value and the optimization process.

The `ExhaustiveOptimizer` class exists to evaluate a metric over a windowed parameter space of fixed step size. This example shows how to use `ExhaustiveOptimizerv4` with the `MeanSquaresImageToImageMetricv4` metric and `Euler2DTransform` transform to survey performance over a parameter space and visualize the results with `matplotlib`.

[1]:
```
import os
import sys
import itertools
from math import pi, sin, cos, sqrt
from urllib.request import urlretrieve

import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import itk
from itkwidgets import view, compare, checkerboard, cm

module_path = os.path.abspath(os.path.join('.'))

if module_path not in sys.path:
    sys.path.append(module_path)
```

### Get sample data to register

In this example we seek to transform an image of an orange to overlay on the image of an apple. We will eventually use the `MeanSquaresImageToImageMetricv4` class to inform the optimizer about how the two images are related given the current parameter state. We can visualize the fixed and moving images with `ITKWidgets`.

[2]:
```
fixed_img_path = 'apple.jpg'
moving_img_path = 'orange.jpg'
```

[3]:
```
if not os.path.exists(fixed_img_path):
    url = 'https://data.kitware.com/api/v1/file/5cad1aec8d777f072b181870/download'
    urlretrieve(url, fixed_img_path)
if not os.path.exists(moving_img_path):
    url = 'https://data.kitware.com/api/v1/file/5cad1aed8d777f072b181879/download'
    urlretrieve(url, moving_img_path)
```

[4]:
```
fixed_img = itk.imread(fixed_img_path, itk.F)
moving_img = itk.imread(moving_img_path, itk.F)
```

[5]:
```
compare(fixed_img, moving_img, ui_collapsed=True)
```

```
AppLayout(children=(HBox(children=(Label(value='Link:'), Checkbox(value=False,␣
↪description='cmap'), Checkbox(v...
```

### Define and Initialize the Transform

In this example we will use an `Euler2DTransform` instance to represent how the moving image will be sampled from the fixed image. The Euler2DTransform documentation shows that the transform has three parameters, first a rotation around a fixed center, followed by a 2D translation. We use a `CenteredTransformInitializer` to estimate what may be a "good" fixed center point at which to define the transform prior to conducting optimization.

[6]:
```
dimension = 2
FixedImageType = itk.Image[itk.F, dimension]
MovingImageType = itk.Image[itk.F, dimension]
TransformType = itk.Euler2DTransform[itk.D]
OptimizerType = itk.ExhaustiveOptimizerv4[itk.D]
MetricType = itk.MeanSquaresImageToImageMetricv4[FixedImageType, MovingImageType]
TransformInitializerType = \
    itk.CenteredTransformInitializer[itk.MatrixOffsetTransformBase[itk.D,2,2],
                                     FixedImageType, MovingImageType]
RegistrationType = itk.ImageRegistrationMethodv4[FixedImageType,MovingImageType]
```

[7]:
```
transform = TransformType.New()

initializer = TransformInitializerType.New(
    Transform=transform,
    FixedImage=fixed_img,
    MovingImage=moving_img,
)
initializer.InitializeTransform()
```

### Run Optimization

We rely on the `ExhaustiveOptimizerv4` class to visualize the parameter space. For this example we choose to visualize the metric value over the first two parameters only, so we set the number of steps in the third dimension to zero. The angle and translation parameters are measured on different scales, so we set the optimizer to take steps of reasonable size along each dimension. An observer is used to save the results of each step.

[8]:
```
metric_results = dict()

metric = MetricType.New()
optimizer = OptimizerType.New()

optimizer.SetNumberOfSteps([10,10,0])

scales = optimizer.GetScales()
scales.SetSize(3)
scales.SetElement(0, 0.1)
scales.SetElement(1, 1.0)
scales.SetElement(2, 1.0)
optimizer.SetScales(scales)

def collect_metric_results():
    metric_results[tuple(optimizer.GetCurrentPosition())] = \
            optimizer.GetCurrentValue()

optimizer.AddObserver(itk.IterationEvent(), collect_metric_results)

registration = RegistrationType.New(Metric=metric,
    Optimizer=optimizer,
    FixedImage=fixed_img,
    MovingImage=moving_img,
    InitialTransform=transform,
    NumberOfLevels=1)
```

[9]:
```
registration.Update()

print(f'MinimumMetricValue: {optimizer.GetMinimumMetricValue():.4f}\t'
      f'MaximumMetricValue: {optimizer.GetMaximumMetricValue():.4f}\n'
      f'MinimumMetricValuePosition: {list(optimizer.GetMinimumMetricValuePosition())}\
↪t'
      f'MaximumMetricValuePosition: {list(optimizer.GetMaximumMetricValuePosition())}\
↪n'
      f'StopConditionDescription: {optimizer.GetStopConditionDescription()}\t')
```

```
MinimumMetricValue: 8195.5765   MaximumMetricValue: 13799.5822
MinimumMetricValuePosition: [0.0, 0.0, 0.0]     MaximumMetricValuePosition: [0.
↪6000000000000001, 10.0, 0.0]
```

(continues on next page)

(continued from previous page)

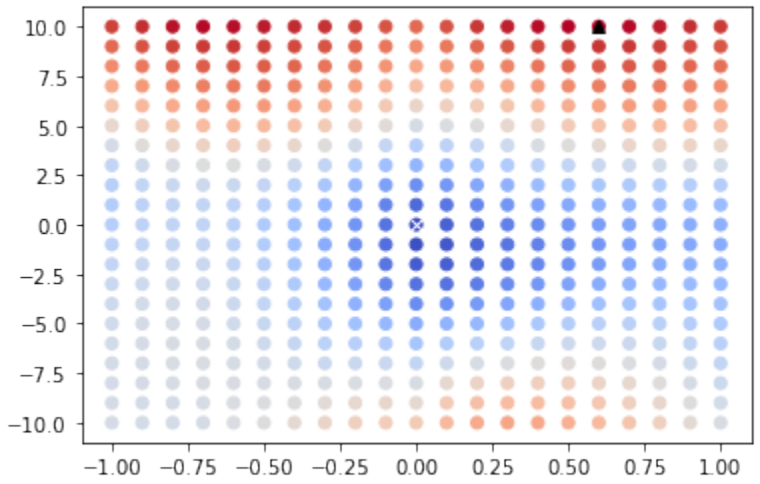
```
StopConditionDescription: ExhaustiveOptimizerv4: Completed sampling of parametric␣
↪space of size 3
```

### Visualize Parameter Space as 2D Scatter Plot

We can use `matplotlib` to view the results of each discrete optimizer step as a 2D scatter plot. In this case the horizontal axis represents the angle that the image is rotated about its fixed center in radians and the vertical axis represents the horizontal distance that the image is translated after rotation. The value of `MeanSquaresImageToImageMetricv4` for each transformation is represented via color gradients. We can also directly plot optimizer extrema for visualization.

[10]:
```
fig = plt.figure()
ax = plt.axes()
ax.scatter([x[0] for x in metric_results.keys()],
           [x[1] for x in metric_results.keys()],
           c=list(metric_results.values()),
           cmap='coolwarm');
ax.plot(optimizer.GetMinimumMetricValuePosition().GetElement(0),
        optimizer.GetMinimumMetricValuePosition().GetElement(1),
        'wx')
ax.plot(optimizer.GetMaximumMetricValuePosition().GetElement(0),
        optimizer.GetMaximumMetricValuePosition().GetElement(1),
        'k^')
```

[10]:
```
[<matplotlib.lines.Line2D at 0x19ced452370>]
```




### Visualize Parameter Space as 3D Surface

We can also plot results in 3D space with `numpy` and `matplotlib`. In this example we use `np.meshgrid` to define the parameter domain and define corresponding metric results as an accompanying `numpy` array. The resulting graph can be used to visualize gradients and visually identify extrema.

[11]:
```
x_unique = list(set(x for (x,y,_) in metric_results.keys()))
y_unique = list(set(y for (x,y,_) in metric_results.keys()))
x_unique.sort()
y_unique.sort()
```

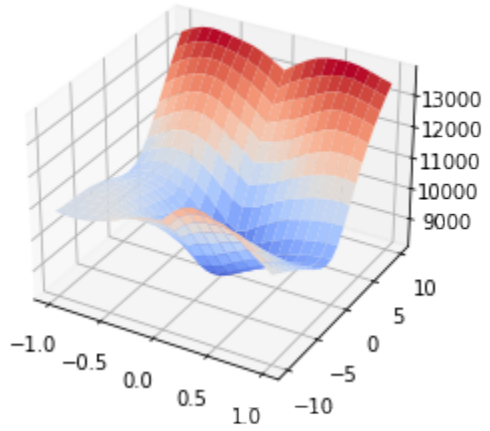
[12]:
```
X, Y = np.meshgrid(x_unique, y_unique)
Z = np.array([[metric_results[(x,y,0)] for x in x_unique] for y in y_unique])
```

[13]:
```
np.shape(Z)
```

[13]:
```
(21, 21)
```

[14]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_surface(X,Y,Z,cmap='coolwarm')
```

[14]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x19ced539fa0>
```




### Clean up

[15]:
```
os.remove(fixed_img_path)
os.remove(moving_img_path)
```

### Synopsis

An optimizer that fully samples a grid on the parametric space.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Jupyter Notebook

launch binder

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkEuler2DTransform.h"
#include "itkExhaustiveOptimizerv4.h"
#include "itkMeanSquaresImageToImageMetricv4.h"
#include "itkCenteredTransformInitializer.h"
#include "itkImageRegistrationMethodv4.h"
#include "itkImage.h"

//  Command observer to monitor the evolution of the registration process.
//
#include "itkCommand.h"
class CommandIterationUpdate : public itk::Command
{
public:
  using Self = CommandIterationUpdate;
  using Superclass = itk::Command;
  using Pointer = itk::SmartPointer<Self>;
  itkNewMacro(Self);

protected:
  CommandIterationUpdate() = default;

public:
  using OptimizerType = itk::ExhaustiveOptimizerv4<double>;
  using OptimizerPointer = const OptimizerType *;

  void
  Execute(itk::Object * caller, const itk::EventObject & event) override
  {
    Execute((const itk::Object *)caller, event);
  }

  void
  Execute(const itk::Object * object, const itk::EventObject & event) override
  {
    auto optimizer = static_cast<OptimizerPointer>(object);
    if (!(itk::IterationEvent().CheckEvent(&event)))
    {
      return;
    }
    std::cout << "Iteration: ";
    std::cout << optimizer->GetCurrentIteration() << "   ";
    std::cout << optimizer->GetCurrentIndex() << "   ";
    std::cout << optimizer->GetCurrentValue() << "   ";
    std::cout << optimizer->GetCurrentPosition() << "   ";
    std::cout << std::endl;
```

(continues on next page)

(continued from previous page)

```
  }
};

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cout << "Usage: " << argv[0] << " fixedImage movingImage" << std::endl;
    return EXIT_FAILURE;
  }
  using FixedImageType = itk::Image<double, 2>;
  using MovingImageType = itk::Image<double, 2>;
  using FixedImageReaderType = itk::ImageFileReader<FixedImageType>;
  using MovingImageReaderType = itk::ImageFileReader<MovingImageType>;
  using TransformType = itk::Euler2DTransform<double>;
  using OptimizerType = itk::ExhaustiveOptimizerv4<double>;
  using MetricType = itk::MeanSquaresImageToImageMetricv4<FixedImageType,␣
↪MovingImageType>;
  using TransformInitializerType = itk::CenteredTransformInitializer<TransformType,␣
↪FixedImageType, MovingImageType>;
  using RegistrationType = itk::ImageRegistrationMethodv4<FixedImageType,␣
↪MovingImageType, TransformType>;

  FixedImageReaderType::Pointer     fixedImageReader = FixedImageReaderType::New();
  MovingImageReaderType::Pointer    movingImageReader = MovingImageReaderType::New();
  FixedImageType::Pointer           fixedImage = FixedImageType::New();
  MovingImageType::Pointer          movingImage = MovingImageType::New();
  TransformType::Pointer            transform = TransformType::New();
  MetricType::Pointer               metric = MetricType::New();
  OptimizerType::Pointer            optimizer = OptimizerType::New();
  RegistrationType::Pointer         registration = RegistrationType::New();
  TransformInitializerType::Pointer initializer = TransformInitializerType::New();

  fixedImageReader->SetFileName(argv[1]);
  fixedImageReader->Update();
  fixedImage = fixedImageReader->GetOutput();

  movingImageReader->SetFileName(argv[2]);
  movingImageReader->Update();
  movingImage = movingImageReader->GetOutput();

  // Create the Command observer and register it with the optimizer.
  //
  CommandIterationUpdate::Pointer observer = CommandIterationUpdate::New();
  optimizer->AddObserver(itk::IterationEvent(), observer);

  unsigned int             angles = 12;
  OptimizerType::StepsType steps(transform->GetNumberOfParameters());
  steps[0] = int(angles / 2);
  steps[1] = 0;
  steps[2] = 0;
  optimizer->SetNumberOfSteps(steps);

  OptimizerType::ScalesType scales(transform->GetNumberOfParameters());
  scales[0] = 2.0 * itk::Math::pi / angles;
  scales[1] = 1.0;
```

(continues on next page)

(continued from previous page)

```
  scales[2] = 1.0;

  optimizer->SetScales(scales);

  initializer->SetTransform(transform);
  initializer->SetFixedImage(fixedImage);
  initializer->SetMovingImage(movingImage);
  initializer->InitializeTransform();

  // Initialize registration
  registration->SetMetric(metric);
  registration->SetOptimizer(optimizer);
  registration->SetFixedImage(fixedImage);
  registration->SetMovingImage(movingImage);
  registration->SetInitialTransform(transform);
  registration->SetNumberOfLevels(1);
  try
  {
    registration->Update();
    std::cout << "  MinimumMetricValue: " << optimizer->GetMinimumMetricValue() <<␣
↪std::endl;
    std::cout << "  MaximumMetricValue: " << optimizer->GetMaximumMetricValue() <<␣
↪std::endl;
    std::cout << "  MinimumMetricValuePosition: " << optimizer->
↪GetMinimumMetricValuePosition() << std::endl;
    std::cout << "  MaximumMetricValuePosition: " << optimizer->
↪GetMaximumMetricValuePosition() << std::endl;
    std::cout << "  StopConditionDescription: " << optimizer->
↪GetStopConditionDescription() << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

### Python

```
# Python example demonstrating itk.ExhaustiveOptimizer usage
import sys
from math import pi

import itk

EXIT_SUCCESS = 0
EXIT_FAILURE = 1


def main(argv):
    if len(argv) < 3:
        raise Exception(f"Usage: {argv[0]} fixed_image moving_image")
```

(continues on next page)

(continued from previous page)

```
    fixed_image = itk.imread(argv[1], itk.F)
    moving_image = itk.imread(argv[2], itk.F)

    dimension = fixed_image.GetImageDimension()
    FixedImageType = type(fixed_image)
    MovingImageType = type(moving_image)

    if dimension == 2:
        transform = itk.Euler2DTransform[itk.D].New()
    else:
        raise Exception(f"Unsupported dimension: {dimension}")

    TransformInitializerType = itk.CenteredTransformInitializer[
        itk.MatrixOffsetTransformBase[itk.D, dimension, dimension],
        FixedImageType,
        MovingImageType,
    ]
    initializer = TransformInitializerType.New(
        Transform=transform,
        FixedImage=fixed_image,
        MovingImage=moving_image,
    )
    initializer.InitializeTransform()

    metric = itk.MeanSquaresImageToImageMetricv4[FixedImageType, MovingImageType].
↪New()

    optimizer = itk.ExhaustiveOptimizerv4[itk.D].New()

    def print_iteration():
        print(
            f"Iteration:"
            f"{optimizer.GetCurrentIteration()}\t"
            f"{list(optimizer.GetCurrentIndex())} \t"
            f"{optimizer.GetCurrentValue():10.4f}\t"
            f"{list(optimizer.GetCurrentPosition())}\t"
        )

    optimizer.AddObserver(itk.IterationEvent(), print_iteration)

    angles = 12
    optimizer.SetNumberOfSteps([int(angles / 2), 0, 0])

    # Initialize scales and set back to optimizer
    scales = optimizer.GetScales()
    scales.SetSize(3)
    scales.SetElement(0, 2.0 * pi / angles)
    scales.SetElement(1, 1.0)
    scales.SetElement(2, 1.0)
    optimizer.SetScales(scales)

    RegistrationType = itk.ImageRegistrationMethodv4[FixedImageType, MovingImageType]
    registration = RegistrationType.New(
        Metric=metric,
        Optimizer=optimizer,
        FixedImage=fixed_image,
```

(continues on next page)

(continued from previous page)

```
        MovingImage=moving_image,
        InitialTransform=transform,
        NumberOfLevels=1,
    )

    try:
        registration.Update()

        print(
            f"MinimumMetricValue: {optimizer.GetMinimumMetricValue():.4f}\t"
            f"MaximumMetricValue: {optimizer.GetMaximumMetricValue():.4f}\n"
            f"MinimumMetricValuePosition: {list(optimizer.
↪GetMinimumMetricValuePosition())}\t"
            f"MaximumMetricValuePosition: {list(optimizer.
↪GetMaximumMetricValuePosition())}\n"
            f"StopConditionDescription: {optimizer.GetStopConditionDescription()}\t"
        )

    except Exception as e:
        print(f"Exception caught: {e}")
        return EXIT_FAILURE

    return EXIT_SUCCESS


if __name__ == "__main__":
    main(sys.argv)
```

## Classes demonstrated

**class ExhaustiveOptimizer** : **public** itk::SingleValuedNonLinearOptimizer
Optimizer that fully samples a grid on the parametric space.

This optimizer is equivalent to an exhaustive search in a discrete grid defined over the parametric space. The grid is centered on the initial position. The subdivisions of the grid along each one of the dimensions of the parametric space is defined by an array of number of steps.

A typical use is to plot the metric space to get an idea of how noisy it is. An example is given below, where it is desired to plot the metric space with respect to translations along x, y and z in a 3D registration application: Here it is assumed that the transform is Euler3DTransform.

```
OptimizerType::StepsType steps( m_Transform->GetNumberOfParameters() );
steps[0] = 10;
steps[1] = 10;
steps[2] = 10;
m_Optimizer->SetNumberOfSteps( steps );
m_Optimizer->SetStepLength( 2 );
```

The optimizer throws IterationEvents after every iteration. We use this to plot the metric space in an image as follows:

```
if( itk::IterationEvent().CheckEvent(& event ) )
{
  IndexType index;
  index[0] = m_Optimizer->GetCurrentIndex()[0];
```

(continues on next page)

(continued from previous page)

```
  index[1] = m_Optimizer->GetCurrentIndex()[1];
  index[2] = m_Optimizer->GetCurrentIndex()[2];
  image->SetPixel( index, m_Optimizer->GetCurrentValue() );
}
```

The image size is expected to be 11 x 11 x 11.

If you wish to use different step lengths along each parametric axis, you can use the SetScales() method. This accepts an array, each element represents the number of subdivisions per step length. For instance scales of [0.5 1 4] along with a step length of 2 will cause the optimizer to search the metric space on a grid with x,y,z spacing of [1 2 8].

Physical dimensions of the grid are influenced by both the scales and the number of steps along each dimension, a side of the region is stepLength*(2*numberOfSteps[d]+1)*scaling[d].

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Exhaustive Optimizer

See itk::ExhaustiveOptimizer for additional documentation.

### Levenberg-Marquardt Optimization

### Synopsis

Optimize a function using the LevenbergMarquardtOptimization class.

### Results

Output:

```
Position: [5.762595813691092, 0.4800781816497411]
Value: [0.18761225810035853, 0.22149805874930717, 0.19626506063745364, 0.
↪008440292123000681, -0.2246669049722252, -0.06499867890559052, -0.03009462205999469,
↪ 0.037078071678616986, 0.1179535364891624, -0.20197691757229475, 0.
↪23114283424137572, -0.09633797060122617, -0.1181447420933539, -0.016296554008191855,
↪ 0.15351231349591643, 0.04282427169413161, -0.011028322317460315, -0.
↪014874778904555797, 0.026054762464067238, 0.08282156821610176, -0.18170363558960823,
↪ 0.014342598340990165, -0.05669067326929067, -0.19960985461710923, 0.
↪07906076620078828, 0.053467861432614505, -0.04393281305375307, -0.21198352299156475,
↪ 0.13281781677553717, 0.19836431320892967, -0.05863022207617963, 0.
↪04153068096802581, 0.027536776558543252, -0.2217222950679556, -0.052451620797857146,
↪ 0.0037302084610484343, 0.10780178216799818, 0.03909757445646189, -0.
↪19500845421763202, 0.13552303389775222, 0.12067588331830059, -0.1646559748487526, -
↪0.17905579039027675, -0.2047507980917569, 0.13126702388154232, 0.2038583258729716,␣
↪0.2315159987359916, 0.03741545669200619, 0.0640979627527889, -0.2180880841666788, 0.
↪09985426440562328, 0.05599101173689913, 0.018079402649374465, -0.11392806561205493,␣
↪0.12335904899417738, -0.1946137987174792, -0.11640787926864604, 0.02577531924348353,
↪ -0.20075160702307215, -0.06148859349321256, 0.13543279691396393, -0.
↪017197866393147798, -0.034013725658296856, -0.057410320040360396, 0.
↪03428199478561922, 0.12079221893803638, -0.16693885394333918, 0.21030957837668662, -
↪0.11858742760250163, -0.05222840443865451, -0.20187684802252903, 0.
↪13116194362153877, -0.04916327491714512, 0.1116844217460855, -0.052688035510255206,␣
↪0.18635323207406884, -0.2359521189116638, -0.03685975982932632, -0.1408713799105108,
↪ 0.04996752375214619, -0.1635648070609097, 0.2208479366940992, 0.
↪00600936209241191364, 0.12473423700590125, -0.061608327743822855,
↪20618061051637326, -0.012960406675505354, 0.1520753363006424, -0.006297055122873374,
↪ 0.13247319130254463, 0.2377413446210319, -0.13626735303743231, -0.
↪005897280453501352, -0.09485834355640499, 0.19074286440427457, -0.19759247327942475,
```

(continues on next page)

(continued from previous page)

### Code

#### C++

```
// Include the Levenberg-Marquardt optimizer and a custom cost function
#include "itkLevenbergMarquardtOptimizer.h"
#include "itkExampleCostFunction.h"

// Typedef the optimizer and cost function, for convenience
using OptimizerType = itk::LevenbergMarquardtOptimizer;
using CostType = itk::ExampleCostFunction;

int
main(int, char *[])
{

  // Instantiate the cost function and optimizer
  CostType::Pointer      cost = CostType::New();
  OptimizerType::Pointer optimizer = OptimizerType::New();

  optimizer->SetNumberOfIterations(100);
  optimizer->UseCostFunctionGradientOff();
  optimizer->SetCostFunction(cost.GetPointer());

  // This is the initial guess for the parameter values, which we set to one
  CostType::ParametersType p(cost->GetNumberOfParameters());
  p.Fill(1);
  optimizer->SetInitialPosition(p);
  optimizer->StartOptimization();

  // Print out some information about the optimization
  // The parameters come out to be near to, but not exactly [5.5, 0.5]
  std::cout << "Position: " << optimizer->GetCurrentPosition() << std::endl;
  std::cout << "Value: " << optimizer->GetValue() << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class LevenbergMarquardtOptimizer** : **public** itk::MultipleValuedNonLinearVnlOptimizer
Wrap of the vnl_levenberg_marquardt algorithm.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Levenberg-Marquardt Optimiztion

See itk::LevenbergMarquardtOptimizer for additional documentation.

## 3.7.2 Statistics

### 2D Gaussian Mixture Model Expectation Maximum

#### Synopsis

2D Gaussian Mixture Model Expectation Maximization.

#### Results

Output:

```
Cluster[0]
    Parameters:
          [101.40933830302448, 99.43004497807948, 1098.5993639665169, -107.
↪16526601343287, -107.16526601343287, 913.9641556669595]
    Proportion:          0.495716
Cluster[1]
    Parameters:
          [196.3354813961237, 195.29542020949035, 991.7367739288584, 84.51759523418217,
↪ 84.51759523418217, 845.9604643808337]
    Proportion:          0.504284
```

#### Code

#### C++

```
#include "itkVector.h"
#include "itkListSample.h"
#include "itkGaussianMixtureModelComponent.h"
#include "itkExpectationMaximizationMixtureModelEstimator.h"
#include "itkNormalVariateGenerator.h"

int
main(int, char *[])
{
  unsigned int numberOfClasses = 2;
  using MeasurementVectorType = itk::Vector<double, 2>;
  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  SampleType::Pointer sample = SampleType::New();


  using NormalGeneratorType = itk::Statistics::NormalVariateGenerator;
  NormalGeneratorType::Pointer normalGenerator = NormalGeneratorType::New();

  // Create the first set of 2D Gaussian samples
  normalGenerator->Initialize(101);

  MeasurementVectorType mv;
  double                mean = 100;
  double                standardDeviation = 30;
  for (unsigned int i = 0; i < 100; ++i)
  {
```

(continues on next page)

(continued from previous page)

```
    mv[0] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    mv[1] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    sample->PushBack(mv);
  }

  // Create the second set of 2D Gaussian samples
  normalGenerator->Initialize(3024);
  mean = 200;
  standardDeviation = 30;
  for (unsigned int i = 0; i < 100; ++i)
  {
    mv[0] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    mv[1] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    sample->PushBack(mv);
  }

  using ParametersType = itk::Array<double>;
  ParametersType params(6);

  // Create the first set of initial parameters
  std::vector<ParametersType> initialParameters(numberOfClasses);
  params[0] = 110.0; // mean of dimension 1
  params[1] = 115.0; // mean of dimension 2
  params[2] = 800.0; // covariance(0,0)
  params[3] = 0;     // covariance(0,1)
  params[4] = 0;     // covariance(1,0)
  params[5] = 805.0; // covariance(1,1)
  initialParameters[0] = params;

  // Create the second set of initial parameters
  params[0] = 210.0; // mean of dimension 1
  params[1] = 215.0; // mean of dimension 2
  params[2] = 850.0; // covariance(0,0)
  params[3] = 0;     // covariance(0,1)
  params[4] = 0;     // covariance(1,0)
  params[5] = 855.0; // covariance(1,1)
  initialParameters[1] = params;

  using ComponentType = itk::Statistics::GaussianMixtureModelComponent<SampleType>;

  // Create the components
  std::vector<ComponentType::Pointer> components;
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    components.push_back(ComponentType::New());
    (components[i])->SetSample(sample);
    (components[i])->SetParameters(initialParameters[i]);
  }

  using EstimatorType = itk::Statistics::ExpectationMaximizationMixtureModelEstimator
↪<SampleType>;
  EstimatorType::Pointer estimator = EstimatorType::New();

  estimator->SetSample(sample);
  estimator->SetMaximumIteration(200);

  itk::Array<double> initialProportions(numberOfClasses);
```

(continues on next page)

(continued from previous page)

```
  initialProportions[0] = 0.5;
  initialProportions[1] = 0.5;

  estimator->SetInitialProportions(initialProportions);

  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    estimator->AddComponent((ComponentType::Superclass *)(components[i]).
↪GetPointer());
  }

  estimator->Update();

  // Output the results
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    std::cout << "Cluster[" << i << "]" << std::endl;
    std::cout << "    Parameters:" << std::endl;
    std::cout << "          " << (components[i])->GetFullParameters() << std::endl;
    std::cout << "    Proportion: ";
    std::cout << "          " << estimator->GetProportions()[i] << std::endl;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TSample**>
**class ExpectationMaximizationMixtureModelEstimator** : **public** itk::*Object*

This class generates the parameter estimates for a mixture model using expectation maximization strategy.

The first template argument is the type of the target sample data. This estimator expects one or more mixture model component objects of the classes derived from the MixtureModelComponentBase. The actual component (or module) parameters are updated by each component. Users can think this class as a strategy or a integration point for the EM procedure. The initial proportion (SetInitialProportions), the input sample (SetSample), the mixture model components (AddComponent), and the maximum iteration (SetMaximumIteration) are required. The EM procedure terminates when the current iteration reaches the maximum iteration or the model parameters converge.

**Recent API changes:** The static const macro to get the length of a measurement vector, `MeasurementVectorSize` has been removed to allow the length of a measurement vector to be specified at run time. It is now obtained at run time from the sample set as input. Please use the function GetMeasurementVectorSize() to get the length.

**See** MixtureModelComponentBase, GaussianMixtureModelComponent

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- 2D Gaussian Mixture Model Expectation Maximum
- Distribution Of Pixels Using GMM EM

- Distribute Sampling Using GMM EM

See itk::Statistics::ExpectationMaximizationMixtureModelEstimator for additional documentation.

### Compute Histogram From Grayscale Image

### Synopsis

Compute a histogram from a grayscale image

### Results

Fig. 346: Input grayscale image.

Output:

```
Frequency = [ 0,
8593,
17734,
11515,
5974,
2225,
2400,
3422,
3531,
3283,
2125,
2628,
1954,
152,
0,
0 ]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageToHistogramFilter.h"
#include "itkImageRandomIteratorWithIndex.h"

int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << argv[0] << "InputFileName NumberOfBins" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  constexpr unsigned int MeasurementVectorSize = 1; // Grayscale
  const auto             binsPerDimension = static_cast<unsigned int>(std::
↪stoi(argv[2]));

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  ImageType::Pointer image = reader->GetOutput();

  using ImageToHistogramFilterType = itk::Statistics::ImageToHistogramFilter
↪<ImageType>;

  ImageToHistogramFilterType::HistogramType::MeasurementVectorType␣
↪lowerBound(binsPerDimension);
  lowerBound.Fill(0);

  ImageToHistogramFilterType::HistogramType::MeasurementVectorType␣
↪upperBound(binsPerDimension);
  upperBound.Fill(255);

  ImageToHistogramFilterType::HistogramType::SizeType size(MeasurementVectorSize);
  size.Fill(binsPerDimension);

  ImageToHistogramFilterType::Pointer imageToHistogramFilter =␣
↪ImageToHistogramFilterType::New();
  imageToHistogramFilter->SetInput(image);
  imageToHistogramFilter->SetHistogramBinMinimum(lowerBound);
  imageToHistogramFilter->SetHistogramBinMaximum(upperBound);
  imageToHistogramFilter->SetHistogramSize(size);

  try
  {
    imageToHistogramFilter->Update();
  }
```

(continues on next page)

(continued from previous page)

```
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  ImageToHistogramFilterType::HistogramType * histogram = imageToHistogramFilter->
↪GetOutput();

  std::cout << "Frequency = [ ";
  for (unsigned int i = 0; i < histogram->GetSize()[0]; ++i)
  {
    std::cout << histogram->GetFrequency(i);

    if (i != histogram->GetSize()[0] - 1)
    {
      std::cout << "," << std::endl;
    }
  }

  std::cout << " ]" << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageToHistogramFilter** : **public** itk::ImageSink<*TImage*>

This class generates a histogram from an image.

The concept of Histogram in ITK is quite generic. It has been designed to manage multiple components data. This class facilitates the computation of an histogram from an image.

This filter is automatically multi-threaded. When AutoMinimumMaximum is off and the NumberOfStreamDivisions is set to more than one, then this filter streams its input in a series of requested regions. A histogram is computed for each streamed and threaded region then merged.

Subclassed by itk::Statistics::MaskedImageToHistogramFilter< TImage, TMaskImage >

See itk::Statistics::ImageToHistogramFilter for additional documentation.

### Compute Histogram of Masked Region in Image

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Compute the histogram of a masked region of an image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkMaskedImageToHistogramFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRGBPixel.h"
#include "itkImageRegionIteratorWithIndex.h"
#include "itkImageRegionIterator.h"
#include "itkRescaleIntensityImageFilter.h"

using RGBPixelType = itk::RGBPixel<unsigned char>;
using RGBImageType = itk::Image<RGBPixelType, 2>;

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
            CreateImage(RGBImageType::Pointer image);
static void CreateHalfMask(itk::ImageRegion<2>, UnsignedCharImageType::Pointer mask);

int
main(int, char *[])
{
  constexpr unsigned int MeasurementVectorSize = 3; // RGB

  RGBImageType::Pointer image = RGBImageType::New();
  CreateImage(image);

  UnsignedCharImageType::Pointer mask = UnsignedCharImageType::New();
  CreateHalfMask(image->GetLargestPossibleRegion(), mask);

  using HistogramFilterType = itk::Statistics::MaskedImageToHistogramFilter
↪<RGBImageType, UnsignedCharImageType>;
  using HistogramMeasurementVectorType = HistogramFilterType::
↪HistogramMeasurementVectorType;
  using HistogramSizeType = HistogramFilterType::HistogramSizeType;
  using HistogramType = HistogramFilterType::HistogramType;

  HistogramFilterType::Pointer histogramFilter = HistogramFilterType::New();
  histogramFilter->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  histogramFilter->SetMaskImage(mask);
  histogramFilter->SetAutoMinimumMaximum(true);

  HistogramSizeType histogramSize(MeasurementVectorSize);

  histogramSize[0] = 4; // number of bins for the Red   channel
  histogramSize[1] = 4; // number of bins for the Green channel
  histogramSize[2] = 4; // number of bins for the Blue  channel

  histogramFilter->SetHistogramSize(histogramSize);
  histogramFilter->SetMarginalScale(10); // Required (could this be set in the filter?
↪)
  histogramFilter->Update();

  const HistogramType * histogram = histogramFilter->GetOutput();

  HistogramType::ConstIterator histogramIterator = histogram->Begin();

  // std::string filename = "/home/doriad/histogram.txt";
  // std::ofstream fout(filename.c_str());

  while (histogramIterator != histogram->End())
  {
    // std::cout << "Index = " << histogramIterator.GetMeasurementVector() <<
↪"Frequency = " <<
    // histogramIterator.GetFrequency() << std::endl; std::cout << "Index = " <<␣
↪histogramIterator.GetIndex() <<
    // "Frequency = " << histogramIterator.GetFrequency() << std::endl; fout <<
↪"Index = " <<
    // histogram->GetIndex(histogramItr.GetMeasurementVector()) << "Frequency = " <<␣
↪histogramItr.GetFrequency() <<
    // std::endl;
    ++histogramIterator;
  }
  // fout.close();

  HistogramType::MeasurementVectorType mv(3);
  mv[0] = 255;
  mv[1] = 0;
  mv[2] = 0;
  std::cout << "Frequency = " << histogram->GetFrequency(histogram->GetIndex(mv)) <<␣
↪std::endl;
  return EXIT_SUCCESS;
}

void
CreateImage(RGBImageType::Pointer image)
{
  // Create a black image with a red square and a green square.
  // This should produce a histogram with very strong spikes.
  RGBImageType::SizeType size;
  size[0] = 3;
  size[1] = 3;

  RGBImageType::IndexType start;
  start[0] = 0;
  start[1] = 0;
```

(continues on next page)

(continued from previous page)

```
  RGBImageType::RegionType region;
  region.SetIndex(start);
  region.SetSize(size);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIteratorWithIndex<RGBImageType> iterator(image, image->
↪GetLargestPossibleRegion());
  iterator.GoToBegin();

  RGBPixelType redPixel;
  redPixel.SetRed(255);
  redPixel.SetGreen(0);
  redPixel.SetBlue(0);

  RGBPixelType blackPixel;
  blackPixel.SetRed(0);
  blackPixel.SetGreen(0);
  blackPixel.SetBlue(0);

  itk::ImageRegionIterator<RGBImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    imageIterator.Set(blackPixel);
    ++imageIterator;
  }

  RGBImageType::IndexType index;
  index[0] = 0;
  index[1] = 0;
  image->SetPixel(index, redPixel);

  index[0] = 1;
  index[1] = 0;
  image->SetPixel(index, redPixel);
}

void CreateHalfMask(itk::ImageRegion<2> region, UnsignedCharImageType::Pointer mask)
{
  mask->SetRegions(region);
  mask->Allocate();
  mask->FillBuffer(0);

  itk::Size<2> regionSize = region.GetSize();

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(mask, region);

  // Make the left half of the mask white and the right half black
  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > regionSize[0] / 2)
    {
      imageIterator.Set(0);
    }
```

(continues on next page)

(continued from previous page)

```
    else
    {
      imageIterator.Set(1);
    }

    ++imageIterator;
  }

  using RescaleFilterType = itk::RescaleIntensityImageFilter<UnsignedCharImageType,␣
↪UnsignedCharImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(mask);
  rescaleFilter->Update();

  using WriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName("mask.png");
  writer->SetInput(rescaleFilter->GetOutput());
  writer->Update();
}
```

### Classes demonstrated

template<typename **TImage**, typename **TMaskImage**>
**class MaskedImageToHistogramFilter** : **public** itk::Statistics::*ImageToHistogramFilter<TImage>*

Generate a histogram from the masked pixels of an image.

This class expands the features of the ImageToHistogramFilter by adding a required MaskImage input image. Only the pixel in the input image where the MaskImage's value is the MaskValue will be added to the computed histogram.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Histogram Of Masked Region In Image

See itk::Statistics::MaskedImageToHistogramFilter for additional documentation.

### Compute Texture Features

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Compute texture features.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkScalarImageToTextureFeaturesFilter.h"

using ImageType = itk::Image<float, 2>;

static void CreateImage(ImageType::Pointer);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using TextureFilterType = itk::Statistics::ScalarImageToTextureFeaturesFilter
↪<ImageType>;
  TextureFilterType::Pointer textureFilter = TextureFilterType::New();
  textureFilter->SetInput(image);
  textureFilter->Update();

  const TextureFilterType::FeatureValueVector * output = textureFilter->
↪GetFeatureMeans();
  for (unsigned int i = 0; i < output->size(); ++i)
  {
    std::cout << (*output)[i] << std::endl;
  }

  return EXIT_SUCCESS;
}

static void
CreateImage(ImageType::Pointer image)
{
  itk::Index<2> index;
  index.Fill(0);

  itk::Size<2> size;
```

(continues on next page)

(continued from previous page)

```
  size.Fill(100);

  itk::ImageRegion<2> region(index, size);
  image->SetRegions(region);
  image->Allocate();
}
```

### Classes demonstrated

template<typename **TImageType**, typename **THistogramFrequencyContainer** = DenseFrequencyContainer2>
**class ScalarImageToTextureFeaturesFilter** : **public** itk::*ProcessObject*

This class computes texture descriptions from an image.

This class computes features that summarize the texture of a given image. The texture features are computed a la Haralick, and have proven to be useful in image classification for biological and medical imaging. This class computes the texture features of an image (optionally in a masked region), averaged across several spatial directions so that they are invariant to rotation.

By default, texture features are computed for each spatial direction and then averaged afterward, so it is possible to access the standard deviations of the texture features. These values give a clue as to texture anisotropy. However, doing this is much more work, because it involved computing one GLCM for each offset given. To compute a single GLCM using the first offset , call FastCalculationsOn(). If this is called, then the texture standard deviations will not be computed (and will be set to zero), but texture computation will be much faster.

This class is templated over the input image type.

Template Parameters: The image type, and the type of histogram frequency container. If you are using a large number of bins per axis, a sparse frequency container may be advisable. The default is to use a dense frequency container.

Inputs and parameters:

a. An image

b. A mask defining the region over which texture features will be calculated. (Optional)

c. The pixel value that defines the "inside" of the mask. (Optional, defaults to 1 if a mask is set.)

d. The set of features to be calculated. These features are defined in the GreyLevelCooccurrenceMatrix-TextureCoefficientsCalculator class. (Optional, defaults to {Energy, Entropy, InverseDifferenceMoment, Inertia, ClusterShade, ClusterProminence}, as in Conners, Trivedi and Harlow.)

e. The number of intensity bins. (Optional, defaults to 256.)

f. The set of directions (offsets) to average across. (Optional, defaults to {(-1, 0), (-1, -1), (0, -1), (1, -1)} for 2D images and scales analogously for ND images.)

g. The pixel intensity range over which the features will be calculated. (Optional, defaults to the full dynamic range of the pixel type.)

In general, the default parameter values should be sufficient.

Outputs: (1) The average value of each feature. (2) The standard deviation in the values of each feature.

Web reference: http://www.fp.ucalgary.ca/mhallbey/tutorial.htm

Print references: Haralick, R.M., K. Shanmugam and I. Dinstein. 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man and Cybernetics. SMC-3(6):610-620.

Haralick, R.M. 1979. Statistical and Structural Approaches to Texture. Proceedings of the IEEE, 67:786-804.

R.W. Conners and C.A. Harlow. A Theoretical Comaprison of Texture Algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2:204-222, 1980.

R.W. Conners, M.M. Trivedi, and C.A. Harlow. Segmentation of a High-Resolution Urban Scene using Texture Operators. Computer Vision, Graphics and Image Processing, 25:273-310, 1984.

Author: Zachary Pincus

**See** ScalarImageToCooccurrenceMatrixFilter

**See** HistogramToTextureFeaturesFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Texture Features

See itk::Statistics::ScalarImageToTextureFeaturesFilter for additional documentation.

### Create Gaussian Distribution

### Synopsis

Create a Gaussian distribution.

### Results

Output:

```
0.396953
```

### Code

### C++

```
#include "itkGaussianDistribution.h"

int
main(int, char *[])
{
  itk::Statistics::GaussianDistribution::Pointer gaussian = itk::Statistics::
↪GaussianDistribution::New();
  gaussian->SetMean(2.0);
  gaussian->SetVariance(1.0);
  std::cout << gaussian->EvaluatePDF(2.1) << std::endl;
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class GaussianDistribution** : **public** itk::Statistics::ProbabilityDistribution

GaussianDistribution class defines the interface for a univariate Gaussian distribution (pdfs, cdfs, etc.).

GaussianDistribution provides access to the probability density function (pdf), the cumulative distribution function (cdf), and the inverse cumulative distribution function for a Gaussian distribution.

The EvaluatePDF(), EvaluateCDF, EvaluateInverseCDF() methods are all virtual, allowing algorithms to be written with an abstract interface to a distribution (with said distribution provided to the algorithm at run-time). Static methods, not requiring an instance of the distribution, are also provided. The static methods allow for optimized access to distributions when the distribution is known a priori to the algorithm.

GaussianDistributions are univariate. Multivariate versions may be provided under a separate superclass (since the parameters to the pdf and cdf would have to be vectors not scalars).

GaussianDistributions can be used for Z-score statistical tests.

**Note** This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://commonfund.nih.gov/bioinformatics.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Gaussian Distribution

See itk::Statistics::GaussianDistribution for additional documentation.

### Create Histogram From List of Measurements

### Synopsis

Create a histogram from a list of sample measurements.

### Results

Output:

```
Histogram vector size: 1
Frequency of 0 : (1 to 1.1001) = 2
Frequency of 1 : (1.1001 to 1.2002) = 0
Frequency of 2 : (1.2002 to 1.3003) = 0
Frequency of 3 : (1.3003 to 1.4004) = 0
Frequency of 4 : (1.4004 to 1.5005) = 0
Frequency of 5 : (1.5005 to 1.6006) = 0
Frequency of 6 : (1.6006 to 1.7007) = 0
Frequency of 7 : (1.7007 to 1.8008) = 0
Frequency of 8 : (1.8008 to 1.9009) = 0
Frequency of 9 : (1.9009 to 2.001) = 1
Total count 3
```

### Code

### C++

```
#include "itkSampleToHistogramFilter.h"
#include "itkListSample.h"
#include "itkHistogram.h"

using MeasurementVectorType = itk::Vector<unsigned char, 1>;
using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;

using HistogramType = itk::Statistics::Histogram<float, itk::Statistics::
↪DenseFrequencyContainer2>;

void
CreateSample(SampleType::Pointer sample);

int
main(int, char *[])
{
  SampleType::Pointer sample = SampleType::New();
  CreateSample(sample);

  using SampleToHistogramFilterType = itk::Statistics::SampleToHistogramFilter
↪<SampleType, HistogramType>;
  SampleToHistogramFilterType::Pointer sampleToHistogramFilter =␣
↪SampleToHistogramFilterType::New();
  sampleToHistogramFilter->SetInput(sample);

  SampleToHistogramFilterType::HistogramSizeType histogramSize(1);
  histogramSize.Fill(10);
  sampleToHistogramFilter->SetHistogramSize(histogramSize);

  sampleToHistogramFilter->Update();

  const HistogramType * histogram = sampleToHistogramFilter->GetOutput();
  std::cout << "Histogram vector size: " << histogram->GetMeasurementVectorSize() <<␣
↪std::endl;


  for (unsigned int i = 0; i < histogram->GetSize()[0]; i++)
  {
    std::cout << "Frequency of " << i << " : (" << histogram->GetBinMin(0, i) << " to
↪" << histogram->GetBinMax(0, i)
              << ") = " << histogram->GetFrequency(i) << std::endl;
  }

  std::cout << "Total count " << histogram->GetTotalFrequency() << std::endl;

  return EXIT_SUCCESS;
}

void
CreateSample(SampleType::Pointer sample)
{
  MeasurementVectorType mv;
  mv[0] = 1.0;
```

(continues on next page)

(continued from previous page)

```
  sample->PushBack(mv);

  mv[0] = 1.0;
  sample->PushBack(mv);

  mv[0] = 2.0;
  sample->PushBack(mv);
}
```

### Classes demonstrated

template<typename **TSample**, typename **THistogram**>
**class SampleToHistogramFilter** : **public** itk::*ProcessObject*
Computes the Histogram corresponding to a Sample.

This filter produces as output the histogram corresponding to the values of a Sample.

**See** Sample, Histogram

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create Histogram From List Of Measurements

See itk::Statistics::SampleToHistogramFilter for additional documentation.

### Create List of Sample Measurements

### Synopsis

Create a lsit of sample measurements.

### Results

Output:

```
id = 0          measurement vector = [1, 2, 4]  frequency = 1
id = 1          measurement vector = [2, 4, 5]  frequency = 1
id = 2          measurement vector = [3, 8, 6]  frequency = 1
id = 0          measurement vector = [1, 2, 4]  frequency = 1
id = 1          measurement vector = [2, 4, 5]  frequency = 1
id = 2          measurement vector = [3, 8, 6]  frequency = 1
Size = 3
Total frequency = 3
```

### Code

### C++

```
#include "itkListSample.h"
#include "itkVector.h"

int
main(int, char *[])
{
  using MeasurementVectorType = itk::Vector<float, 3>;
  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  SampleType::Pointer sample = SampleType::New();

  MeasurementVectorType mv;
  mv[0] = 1.0;
  mv[1] = 2.0;
  mv[2] = 4.0;

  sample->PushBack(mv);

  sample->Resize(3);

  mv[0] = 2.0;
  mv[1] = 4.0;
  mv[2] = 5.0;
  sample->SetMeasurementVector(1, mv);

  mv[0] = 3.0;
  mv[1] = 8.0;
  mv[2] = 6.0;
  sample->SetMeasurementVector(2, mv);

  for (unsigned long i = 0; i < sample->Size(); ++i)
  {
    std::cout << "id = " << i << "\t measurement vector = " << sample->
↪GetMeasurementVector(i)
              << "\t frequency = " << sample->GetFrequency(i) << std::endl;
  }

  SampleType::Iterator iter = sample->Begin();

  while (iter != sample->End())
  {
    std::cout << "id = " << iter.GetInstanceIdentifier() << "\t measurement vector =
↪" << iter.GetMeasurementVector()
              << "\t frequency = " << iter.GetFrequency() << std::endl;
    ++iter;
  }

  std::cout << "Size = " << sample->Size() << std::endl;
  std::cout << "Total frequency = " << sample->GetTotalFrequency() << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TMeasurementVector**>
**class ListSample** : **public** itk::Statistics::Sample<*TMeasurementVector*>
This class is the native implementation of the a Sample with an STL container.

ListSample stores measurements in a list type structure (as opposed to a Histogram, etc.). ListSample allows duplicate measurements. ListSample is not sorted.

ListSample does not allow the user to specify the frequency of a measurement directly. The GetFrequency() methods returns 1 if the measurement exists in the list, 0 otherwise.

**See** Sample, Histogram

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create List Of Sample Measurements

See itk::Statistics::ListSample for additional documentation.

### Create List of Samples From Image Without Duplication

### Synopsis

Create a list of samples from an image without duplicating the data.

### Results

Output:

```
[96.6165]
[833.994]
[935.002]
[571.885]
[663.087]
[496.426]
[429.681]
[649.809]
[333.22]
[425.118]
[965.227]
[568.819]
[133.191]
[547.472]
[361.405]
[136.599]
[823.276]
[802.849]
[475.275]
[947.74]
[96.6244]
[965.532]
[690.606]
```

(continues on next page)

(continued from previous page)

```
[13.2403]
[529.497]
[258.332]
[780.933]
[135.776]
[985.543]
[23.581]
[325.735]
[623.222]
[485.055]
...
```

### Code

### C++

```
#include "itkImageToListSampleAdaptor.h"
#include "itkImage.h"
#include "itkRandomImageSource.h"
#include "itkComposeImageFilter.h"

int
main(int, char *[])
{
  using FloatImage2DType = itk::Image<float, 2>;

  itk::RandomImageSource<FloatImage2DType>::Pointer random;
  random = itk::RandomImageSource<FloatImage2DType>::New();

  random->SetMin(0.0);
  random->SetMax(1000.0);

  using SpacingValueType = FloatImage2DType::SpacingValueType;
  using SizeValueType = FloatImage2DType::SizeValueType;
  using PointValueType = FloatImage2DType::PointValueType;

  SizeValueType size[2] = { 20, 20 };
  random->SetSize(size);

  SpacingValueType spacing[2] = { 0.7, 2.1 };
  random->SetSpacing(spacing);

  PointValueType origin[2] = { 15, 400 };
  random->SetOrigin(origin);

  using MeasurementVectorType = itk::FixedArray<float, 1>;
  using ArrayImageType = itk::Image<MeasurementVectorType, 2>;
  using CasterType = itk::ComposeImageFilter<FloatImage2DType, ArrayImageType>;

  CasterType::Pointer caster = CasterType::New();
  caster->SetInput(random->GetOutput());
  caster->Update();

  using SampleType = itk::Statistics::ImageToListSampleAdaptor<ArrayImageType>;
```

(continues on next page)

(continued from previous page)

```
  SampleType::Pointer sample = SampleType::New();

  sample->SetImage(caster->GetOutput());

  SampleType::Iterator iter = sample->Begin();

  while (iter != sample->End())
  {
    std::cout << iter.GetMeasurementVector() << std::endl;
    ++iter;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TImage**>
**class ImageToListSampleAdaptor** : **public** itk::Statistics::*ListSample*<MeasurementVectorPixelTraits<*TImage*::PixelType>

This class provides ListSample interface to ITK Image.

After calling SetImage( const Image * ) method to plug in the image object, users can use Sample interfaces to access Image data. The resulting data are a list of measurement vectors.

The measurement vector type is determined from the image pixel type. This class handles images with scalar, fixed array or variable length vector pixel types.

**See** Sample, ListSample

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create List Of Samples From Image Without Duplication

See itk::Statistics::ImageToListSampleAdaptor for additional documentation.

### Create List of Samples With Associated ID's

### Synopsis

Create a list of samples with associated class IDs.

### Results

Output:

```
instance identifier = 0        measurement vector = [1, 2, 4]  frequency = 1   class␣
↪label = 0
instance identifier = 1        measurement vector = [2, 4, 5]  frequency = 1   class␣
↪label = 0
instance identifier = 2        measurement vector = [3, 8, 6]  frequency = 1   class␣
↪label = 1
instance identifier = 0        measurement vector = [1, 2, 4]  frequency = 1
instance identifier = 1        measurement vector = [2, 4, 5]  frequency = 1
```

### Code

### C++

```
#include "itkListSample.h"
#include "itkMembershipSample.h"
#include "itkVector.h"

int
main(int, char *[])
{
  using MeasurementVectorType = itk::Vector<float, 3>;
  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  SampleType::Pointer   sample = SampleType::New();
  MeasurementVectorType mv;

  mv[0] = 1.0;
  mv[1] = 2.0;
  mv[2] = 4.0;
  sample->PushBack(mv);

  mv[0] = 2.0;
  mv[1] = 4.0;
  mv[2] = 5.0;
  sample->PushBack(mv);

  mv[0] = 3.0;
  mv[1] = 8.0;
  mv[2] = 6.0;
  sample->PushBack(mv);
  using MembershipSampleType = itk::Statistics::MembershipSample<SampleType>;

  MembershipSampleType::Pointer membershipSample = MembershipSampleType::New();

  membershipSample->SetSample(sample);
  membershipSample->SetNumberOfClasses(2);

  membershipSample->AddInstance(0U, 0UL);
  membershipSample->AddInstance(0U, 1UL);
  membershipSample->AddInstance(1U, 2UL);

  MembershipSampleType::ConstIterator iter = membershipSample->Begin();
```

(continues on next page)

(continued from previous page)

```
  while (iter != membershipSample->End())
  {
    std::cout << "instance identifier = " << iter.GetInstanceIdentifier()
              << "\t measurement vector = " << iter.GetMeasurementVector() << "\t␣
↪frequency = " << iter.GetFrequency()
              << "\t class label = " << iter.GetClassLabel() << std::endl;
    ++iter;
  }


  MembershipSampleType::ClassSampleType::ConstPointer classSample = membershipSample->
↪GetClassSample(0);

  MembershipSampleType::ClassSampleType::ConstIterator c_iter = classSample->Begin();

  while (c_iter != classSample->End())
  {
    std::cout << "instance identifier = " << c_iter.GetInstanceIdentifier()
              << "\t measurement vector = " << c_iter.GetMeasurementVector()
              << "\t frequency = " << c_iter.GetFrequency() << std::endl;
    ++c_iter;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TSample**>
**class MembershipSample** : **public** itk::DataObject

Container for storing the instance-identifiers of other sample with their associated class labels.

This class does not store any measurement data. In a sense, you can think it as an additional information to basic samples (such as Histogram, PointSetListSampleAdaptor, and ImageToListSampleAdaptor). The additional information is a class label for a measurement vector. Obviously without such basic types of sample, this one is meaningless. You can call any basic methods that has been defined in the Sample class such as GetMeasurementVector and GetFrequency. You can query the class label for an instance using an instance-identifier. Another new and important method is the GetClassSample method. With a given class label, it returns a pointer to the Subsample object that has all the instance-identifiers of instances that belong to the class.

This class is templated over the type of the basic sample. To use all the method, you should first plug in a basic type sample using the SetSample method

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Create List Of Samples With Associated ID's

See itk::Statistics::MembershipSample for additional documentation.

### Distribute Sampling Using GMM EM

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Compute distributions of samples using GMM EM.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkVector.h"
#include "itkListSample.h"
#include "itkGaussianMixtureModelComponent.h"
#include "itkExpectationMaximizationMixtureModelEstimator.h"
#include "itkNormalVariateGenerator.h"

int
main(int, char *[])
{
  unsigned int numberOfClasses = 2;
  using MeasurementVectorType = itk::Vector<double, 1>;
  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  SampleType::Pointer sample = SampleType::New();

  using NormalGeneratorType = itk::Statistics::NormalVariateGenerator;
  NormalGeneratorType::Pointer normalGenerator = NormalGeneratorType::New();

  normalGenerator->Initialize(101);

  MeasurementVectorType mv;
  double                mean = 100;
  double                standardDeviation = 30;
  for (unsigned int i = 0; i < 10; ++i)
  {
    mv[0] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    std::cout << "m[" << i << "] = " << mv[0] << std::endl;
    sample->PushBack(mv);
  }

  normalGenerator->Initialize(3024);
  mean = 200;
```

(continues on next page)

(continued from previous page)

```
  standardDeviation = 30;
  for (unsigned int i = 0; i < 10; ++i)
  {
    mv[0] = (normalGenerator->GetVariate() * standardDeviation) + mean;
    std::cout << "m[" << i << "] = " << mv[0] << std::endl;
    sample->PushBack(mv);
  }

  using ParametersType = itk::Array<double>;
  ParametersType params1(2);

  std::vector<ParametersType> initialParameters(numberOfClasses);
  params1[0] = 110.0;
  params1[1] = 50.0;
  initialParameters[0] = params1;

  ParametersType params2(2);
  params2[0] = 210.0;
  params2[1] = 50.0;
  initialParameters[1] = params2;

  using ComponentType = itk::Statistics::GaussianMixtureModelComponent<SampleType>;

  std::vector<ComponentType::Pointer> components;
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    components.push_back(ComponentType::New());
    components[i]->SetSample(sample);
    components[i]->SetParameters(initialParameters[i]);
  }

  using EstimatorType = itk::Statistics::ExpectationMaximizationMixtureModelEstimator
↪<SampleType>;
  EstimatorType::Pointer estimator = EstimatorType::New();

  estimator->SetSample(sample);
  estimator->SetMaximumIteration(500);

  itk::Array<double> initialProportions(numberOfClasses);
  initialProportions[0] = 0.5;
  initialProportions[1] = 0.5;

  estimator->SetInitialProportions(initialProportions);

  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    estimator->AddComponent((ComponentType::Superclass *)components[i].GetPointer());
  }

  estimator->Update();

  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    std::cout << "Cluster[" << i << "]" << std::endl;
    std::cout << "    Parameters:" << std::endl;
    std::cout << "        " << components[i]->GetFullParameters() << std::endl;
    std::cout << "    Proportion: ";
```

(continues on next page)

(continued from previous page)

```
    std::cout << "          " << estimator->GetProportions()[i] << std::endl;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TSample**>
**class ExpectationMaximizationMixtureModelEstimator** : **public** itk::*Object*

This class generates the parameter estimates for a mixture model using expectation maximization strategy.

The first template argument is the type of the target sample data. This estimator expects one or more mixture model component objects of the classes derived from the MixtureModelComponentBase. The actual component (or module) parameters are updated by each component. Users can think this class as a strategy or a integration point for the EM procedure. The initial proportion (SetInitialProportions), the input sample (SetSample), the mixture model components (AddComponent), and the maximum iteration (SetMaximumIteration) are required. The EM procedure terminates when the current iteration reaches the maximum iteration or the model parameters converge.

**Recent API changes:** The static const macro to get the length of a measurement vector, `MeasurementVectorSize` has been removed to allow the length of a measurement vector to be specified at run time. It is now obtained at run time from the sample set as input. Please use the function GetMeasurementVectorSize() to get the length.

**See** MixtureModelComponentBase, GaussianMixtureModelComponent

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- 2D Gaussian Mixture Model Expectation Maximum
- Distribution Of Pixels Using GMM EM
- Distribute Sampling Using GMM EM

See itk::Statistics::ExpectationMaximizationMixtureModelEstimator for additional documentation.

## Distribution of Pixels Using GMM EM

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

## Synopsis

Compute distributions of image pixels using GMM EM.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkVector.h"
#include "itkListSample.h"
#include "itkGaussianMixtureModelComponent.h"
#include "itkExpectationMaximizationMixtureModelEstimator.h"
#include "itkSampleClassifierFilter.h"
#include "itkMaximumDecisionRule.h"
#include "itkImageToListSampleFilter.h"
#include "itkCovariantVector.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileReader.h"
#include "itkSimpleFilterWatcher.h"

using PixelType = itk::CovariantVector<unsigned char, 3>;
using ImageType = itk::Image<PixelType, 2>;

#undef USE_CONTROLLED_IMAGE
#ifdef USE_CONTROLLED_IMAGE
static void
ControlledImage(ImageType::Pointer image);
#else
static void
RandomImage(ImageType::Pointer image);
#endif

int
main(int /*argc*/, char * /*argv*/[])
{

  ImageType::Pointer image = ImageType::New();

#ifdef USE_CONTROLLED_IMAGE
  ControlledImage(image);
#else
  RandomImage(image);
#endif

  using ImageToListSampleFilterType = itk::Statistics::ImageToListSampleFilter
↪<ImageType>;
  ImageToListSampleFilterType::Pointer imageToListSampleFilter =␣
↪ImageToListSampleFilterType::New();
  imageToListSampleFilter->SetInput(image);
  imageToListSampleFilter->Update();
```

(continues on next page)

(continued from previous page)

```
  unsigned int numberOfClasses = 3;

  using ParametersType = itk::Array<double>;
  ParametersType params(numberOfClasses + numberOfClasses * numberOfClasses); // 3␣
↪for means and 9 for 3x3 covariance

  // Create the first set (for the first cluster/model) of initial parameters
  std::vector<ParametersType> initialParameters(numberOfClasses);
  for (unsigned int i = 0; i < 3; i++)
  {
    params[i] = 5.0; // mean of dimension i
  }
  unsigned int counter = 0;
  for (unsigned int i = 0; i < 3; i++)
  {
    for (unsigned int j = 0; j < 3; j++)
    {
      if (i == j)
      {
        params[3 + counter] = 5; // diagonal
      }
      else
      {
        params[3 + counter] = 0; // off-diagonal
      }
      counter++;
    }
  }


  initialParameters[0] = params;

  // Create the second set (for the second cluster/model) of initial parameters
  params[0] = 210.0;
  params[1] = 5.0;
  params[2] = 5.0;
  counter = 0;
  for (unsigned int i = 0; i < 3; i++)
  {
    for (unsigned int j = 0; j < 3; j++)
    {
      if (i == j)
      {
        params[3 + counter] = 5; // diagonal
      }
      else
      {
        params[3 + counter] = 0; // off-diagonal
      }
      counter++;
    }
  }
  initialParameters[1] = params;

  // Create the third set (for the third cluster/model) of initial parameters
  params[0] = 5.0;
  params[1] = 210.0;
```

(continues on next page)

(continued from previous page)

```
  params[2] = 5.0;
  counter = 0;
  for (unsigned int i = 0; i < 3; i++)
  {
    for (unsigned int j = 0; j < 3; j++)
    {
      if (i == j)
      {
        params[3 + counter] = 5; // diagonal
      }
      else
      {
        params[3 + counter] = 0; // off-diagonal
      }
      counter++;
    }
  }
  initialParameters[2] = params;

  std::cout << "Initial parameters: " << std::endl;
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    std::cout << initialParameters[i] << std::endl;
  }

  using ComponentType = itk::Statistics::GaussianMixtureModelComponent
↪<ImageToListSampleFilterType::ListSampleType>;

  std::cout << "Number of samples: " << imageToListSampleFilter->GetOutput()->
↪GetTotalFrequency() << std::endl;

  // Create the components
  std::vector<ComponentType::Pointer> components;
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    components.push_back(ComponentType::New());
    (components[i])->SetSample(imageToListSampleFilter->GetOutput());
    (components[i])->SetParameters(initialParameters[i]);
  }


  using EstimatorType =
    itk::Statistics::ExpectationMaximizationMixtureModelEstimator
↪<ImageToListSampleFilterType::ListSampleType>;
  EstimatorType::Pointer estimator = EstimatorType::New();

  estimator->SetSample(imageToListSampleFilter->GetOutput());
  estimator->SetMaximumIteration(200);

  itk::Array<double> initialProportions(numberOfClasses);
  initialProportions[0] = 0.33;
  initialProportions[1] = 0.33;
  initialProportions[2] = 0.33;

  std::cout << "Initial proportions: " << initialProportions << std::endl;

  estimator->SetInitialProportions(initialProportions);
```

(continues on next page)

(continued from previous page)

```
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    estimator->AddComponent(components[i]);
  }
  // itk::SimpleFilterWatcher watcher(estimator);
  estimator->Update();

  // Output the results
  for (unsigned int i = 0; i < numberOfClasses; i++)
  {
    std::cout << "Cluster[" << i << "]" << std::endl;
    std::cout << "    Parameters:" << std::endl;
    std::cout << "          " << (components[i])->GetFullParameters() << std::endl;
    std::cout << "    Proportion: ";
    // Outputs: // mean of dimension 1, mean of dimension 2, covariance(0,0),␣
↪covariance(0,1), covariance(1,0),
    // covariance(1,1)
    std::cout << "          " << estimator->GetProportions()[i] << std::endl;
  }

  // Display the membership of each sample
  using FilterType = itk::Statistics::SampleClassifierFilter
↪<ImageToListSampleFilterType::ListSampleType>;

  using DecisionRuleType = itk::Statistics::MaximumDecisionRule;
  DecisionRuleType::Pointer decisionRule = DecisionRuleType::New();

  using ClassLabelVectorObjectType = FilterType::ClassLabelVectorObjectType;
  using ClassLabelVectorType = FilterType::ClassLabelVectorType;

  ClassLabelVectorObjectType::Pointer classLabelsObject = ClassLabelVectorObjectType::
↪New();
  ClassLabelVectorType &              classLabelVector = classLabelsObject->Get();

  using ClassLabelType = FilterType::ClassLabelType;

  ClassLabelType class0 = 0;
  classLabelVector.push_back(class0);

  ClassLabelType class1 = 1;
  classLabelVector.push_back(class1);

  ClassLabelType class2 = 2;
  classLabelVector.push_back(class2);

  FilterType::Pointer sampleClassifierFilter = FilterType::New();
  sampleClassifierFilter->SetInput(imageToListSampleFilter->GetOutput());
  sampleClassifierFilter->SetNumberOfClasses(numberOfClasses);
  sampleClassifierFilter->SetClassLabels(classLabelsObject);
  sampleClassifierFilter->SetDecisionRule(decisionRule);
  sampleClassifierFilter->SetMembershipFunctions(estimator->GetOutput());
  sampleClassifierFilter->Update();

  const FilterType::MembershipSampleType *        membershipSample =␣
↪sampleClassifierFilter->GetOutput();
  FilterType::MembershipSampleType::ConstIterator iter = membershipSample->Begin();
```

(continues on next page)

(continued from previous page)

```
  while (iter != membershipSample->End())
  {
    std::cout << (int)iter.GetMeasurementVector()[0] << " " << (int)iter.
↪GetMeasurementVector()[1] << " "
              << (int)iter.GetMeasurementVector()[2] << " : " << iter.GetClassLabel()
↪<< std::endl;
    ++iter;
  }

  return EXIT_SUCCESS;
}

#ifdef USE_CONTROLLED_IMAGE
void
ControlledImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 10;
  size[1] = 10;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a red and a green square
  itk::CovariantVector<unsigned char, 3> green;
  green[0] = 0;
  green[1] = 255;
  green[2] = 0;

  itk::CovariantVector<unsigned char, 3> red;
  red[0] = 255;
  red[1] = 0;
  red[2] = 0;

  itk::CovariantVector<unsigned char, 3> black;
  black[0] = 0;
  black[1] = 0;
  black[2] = 0;

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);
  imageIterator.GoToBegin();

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 2 && imageIterator.GetIndex()[0] < 5 &&␣
↪imageIterator.GetIndex()[1] > 2 &&
        imageIterator.GetIndex()[1] < 5)
```

(continues on next page)

(continued from previous page)

```
    {
      imageIterator.Set(green);
    }
    else if (imageIterator.GetIndex()[0] > 6 && imageIterator.GetIndex()[0] < 9 &&␣
↪imageIterator.GetIndex()[1] > 6 &&
             imageIterator.GetIndex()[1] < 9)
    {
      imageIterator.Set(red);
    }
    else
    {
      imageIterator.Set(black);
    }
    ++imageIterator;
  }
}
#else

void
RandomImage(ImageType::Pointer image)
{
  // Create an image
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 10;
  size[1] = 10;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);
  imageIterator.GoToBegin();

  while (!imageIterator.IsAtEnd())
  {
    // Get a random color
    itk::CovariantVector<unsigned char, 3> pixel;
    pixel[0] = rand() * 255;
    pixel[1] = rand() * 255;
    pixel[2] = rand() * 255;
    imageIterator.Set(pixel);
    ++imageIterator;
  }
}
#endif
```

### Classes demonstrated

template<typename **TSample**>
**class ExpectationMaximizationMixtureModelEstimator** : **public** itk::*Object*

This class generates the parameter estimates for a mixture model using expectation maximization strategy.

The first template argument is the type of the target sample data. This estimator expects one or more mixture model component objects of the classes derived from the MixtureModelComponentBase. The actual component (or module) parameters are updated by each component. Users can think this class as a strategy or a integration point for the EM procedure. The initial proportion (SetInitialProportions), the input sample (SetSample), the mixture model components (AddComponent), and the maximum iteration (SetMaximumIteration) are required. The EM procedure terminates when the current iteration reaches the maximum iteration or the model parameters converge.

**Recent API changes:** The static const macro to get the length of a measurement vector, `MeasurementVectorSize` has been removed to allow the length of a measurement vector to be specified at run time. It is now obtained at run time from the sample set as input. Please use the function GetMeasurementVectorSize() to get the length.

**See** MixtureModelComponentBase, GaussianMixtureModelComponent

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- 2D Gaussian Mixture Model Expectation Maximum
- Distribution Of Pixels Using GMM EM
- Distribute Sampling Using GMM EM

See itk::Statistics::ExpectationMaximizationMixtureModelEstimator for additional documentation.

## Histogram Creation and Bin Access

### Synopsis

This example shows how to create a `Histogram` object and use it.

We call an instance in a `Histogram` object a *bin*. The Histogram differs from the itk::Statistics::ListSample, itk::Statistics::ImageToListSampleAdaptor, or itk::Statistics::PointSetToListSampleAdaptor in significant ways. Histograms can have a variable number of values (`unsigned long` type) for each measurement vector, while the three other classes have a fixed value (one) for all measurement vectors. Also those array-type containers can have multiple instances (data elements) with identical measurement vector values. However, in a Histogram object, there is one unique instance for any given measurement vector.

Here we create a histogram with dense frequency containers. In this example we will not have any zero-frequency measurements, so the dense frequency container is the appropriate choice. If the histogram is expected to have many empty (zero) bins, a sparse frequency container would be the better option. Note that this is not configurable in Python. Here we also set the size of the measurement vectors to be 2 components.

Output from the code below:

```
Frequency of the bin at index [0, 2] is 5 and the bin's instance identifier is 6
```

### Code

### Python

```
#!/usr/bin/env python

from __future__ import print_function

import itk

numberOfComponents = 2

histogram = itk.Histogram.New(MeasurementVectorSize=numberOfComponents)

# We initialize it as a 3x3 histogram with equal size intervals.
size = itk.Array.UL(numberOfComponents)
size.Fill(3)

lowerBound = 1.1, 2.6
upperBound = 7.1, 8.6
histogram.Initialize(size, lowerBound, upperBound)

# Now the histogram is ready for storing frequency values. There
# are three ways of accessing data elements in the histogram:
# - using instance identifiers---just like any other Sample object;
# - using n-dimensional indices---just like an Image object;
# - using an iterator---just like any other Sample object.
#
# In this example, the index (0, 0) refers the same bin as the instance
# identifier (0) refers to. The instance identifier of the index (0,
# 1) is (3), (0, 2) is (6), (2, 2) is (8), and so on.

frequencies = [(0, 0), (1, 2), (2, 3), (3, 2), (4, 1), (5, 1), (6, 5), (7, 4), (8, 0)]

for instance_identifier, frequency in frequencies:
    histogram.SetFrequency(instance_identifier, frequency)

# Let us examine if the frequency is set correctly by calling the
# GetFrequency(index) method. We can use the
# GetFrequency(instance identifier) method for the same purpose.

index = [0, 2]
print(
    "Frequency of the bin at index",
    index,
    "is",
    histogram.GetFrequency(index),
    "and the bin's instance identifier is",
    histogram.GetInstanceIdentifier(index),
)
```

### C++

```
#include "itkHistogram.h"
#include "itkDenseFrequencyContainer2.h"

int
main()
{
  using MeasurementType = float;
  using FrequencyContainerType = itk::Statistics::DenseFrequencyContainer2;

  constexpr unsigned int numberOfComponents = 2;
  using HistogramType = itk::Statistics::Histogram<MeasurementType,␣
↪FrequencyContainerType>;

  HistogramType::Pointer histogram = HistogramType::New();
  histogram->SetMeasurementVectorSize(numberOfComponents);

  // We initialize it as a 3x3 histogram with equal size intervals.

  HistogramType::SizeType size(numberOfComponents);
  size.Fill(3);
  HistogramType::MeasurementVectorType lowerBound(numberOfComponents);
  HistogramType::MeasurementVectorType upperBound(numberOfComponents);
  lowerBound[0] = 1.1;
  lowerBound[1] = 2.6;
  upperBound[0] = 7.1;
  upperBound[1] = 8.6;

  histogram->Initialize(size, lowerBound, upperBound);

  // Now the histogram is ready for storing frequency values. There
  // are three ways of accessing data elements in the histogram:
  // - using instance identifiers---just like any other Sample object;
  // - using n-dimensional indices---just like an Image object;
  // - using an iterator---just like any other Sample object.
  //
  // In this example, the index (0, 0) refers the same bin as the instance
  // identifier (0) refers to. The instance identifier of the index (0,
  // 1) is (3), (0, 2) is (6), (2, 2) is (8), and so on.

  histogram->SetFrequency(0, 0);
  histogram->SetFrequency(1, 2);
  histogram->SetFrequency(2, 3);
  histogram->SetFrequency(3, 2);
  histogram->SetFrequency(4, 1);
  histogram->SetFrequency(5, 1);
  histogram->SetFrequency(6, 5);
  histogram->SetFrequency(7, 4);
  histogram->SetFrequency(8, 0);

  // Let us examine if the frequency is set correctly by calling the
  // GetFrequency(index) method. We can use the
  // GetFrequency(instance identifier) method for the same purpose.

  HistogramType::IndexType index(numberOfComponents);
  index[0] = 0;
```

(continues on next page)

(continued from previous page)

```
  index[1] = 2;
  std::cout << "Frequency of the bin at index " << index << " is " << histogram->
↪GetFrequency(index)
            << " and the bin's instance identifier is " << histogram->
↪GetInstanceIdentifier(index) << std::endl;

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TMeasurement** = float, typename **TFrequencyContainer** = DenseFrequencyContainer2>
**class Histogram** : **public** itk::Statistics::Sample<Array<*TMeasurement*>>

This class stores measurement vectors in the context of n-dimensional histogram.

Histogram represents an ND histogram. Histogram bins can be regularly or irregularly spaced. The storage for the histogram is managed via the FrequencyContainer specified by the template argument. The default frequency container is a DenseFrequencyContainer. A SparseFrequencyContainer can be used as an alternative.

Frequencies of a bin (SetFrequency(), IncreaseFrequency()) can be specified by measurement, index, or instance identifier.

Measurements can be queried by bin index or instance identifier. In this case, the measurement returned is the centroid of the histogram bin.

The Initialize() method is used to specify the number of bins for each dimension of the histogram. An overloaded version also allows for regularly spaced bins to defined. To define irregularly sized bins, use the SetBinMin()/SetBinMax() methods.

If you do not know the length of the measurement vector at compile time, you should use the VariableDimensionHistogram class, instead of the Histogram class.

If you know the length of the measurement vector at compile time, it can conveniently be obtained from MeasurementVectorTraits. For instance, instantiate a histogram as below:

```
using HistogramType = Histogram< THistogramMeasurement, typename␣
↪TFrequencyContainer >;
```

**See** Sample, DenseFrequencyContainer, SparseFrequencyContainer, VariableDimensionHistogram

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Histogram Creation And Bin Access

See itk::Statistics::Histogram for additional documentation.

### Spatial Search

### Synopsis

Spatial search.

### Results

Output:

```
K-Neighbor search:
[9, 9]
[7, 7]
[8, 8]
Radius search:
There are 4 neighbors.
[7, 7]
[8, 8]
[9, 9]
[10, 10]
```

### Code

### C++

```
#include "itkVector.h"
#include "itkListSample.h"
#include "itkWeightedCentroidKdTreeGenerator.h"
#include "itkEuclideanDistanceMetric.h"

int
main(int, char *[])
{
  using MeasurementVectorType = itk::Vector<float, 2>;

  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  SampleType::Pointer sample = SampleType::New();
  sample->SetMeasurementVectorSize(2);

  MeasurementVectorType mv;
  for (unsigned int i = 0; i < 100; ++i)
  {
    mv[0] = static_cast<float>(i);
    mv[1] = static_cast<float>(i);
    sample->PushBack(mv);
  }

  using TreeGeneratorType = itk::Statistics::KdTreeGenerator<SampleType>;
  TreeGeneratorType::Pointer treeGenerator = TreeGeneratorType::New();
  treeGenerator->SetSample(sample);
  treeGenerator->SetBucketSize(16);
  treeGenerator->Update();
```

(continues on next page)

(continued from previous page)

```
  using TreeType = TreeGeneratorType::KdTreeType;

  TreeType::Pointer tree = treeGenerator->GetOutput();

  MeasurementVectorType queryPoint;
  queryPoint[0] = 10.0;
  queryPoint[1] = 7.0;

  // K-Neighbor search
  std::cout << "K-Neighbor search:" << std::endl;
  unsigned int                           numberOfNeighbors = 3;
  TreeType::InstanceIdentifierVectorType neighbors;
  tree->Search(queryPoint, numberOfNeighbors, neighbors);

  for (unsigned long neighbor : neighbors)
  {
    std::cout << tree->GetMeasurementVector(neighbor) << std::endl;
  }

  // Radius search
  std::cout << "Radius search:" << std::endl;
  double radius = 4.0;
  tree->Search(queryPoint, radius, neighbors);
  std::cout << "There are " << neighbors.size() << " neighbors." << std::endl;
  for (unsigned long neighbor : neighbors)
  {
    std::cout << tree->GetMeasurementVector(neighbor) << std::endl;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TSample**>
**class KdTreeGenerator** : **public** itk::*Object*

This class generates a KdTree object without centroid information.

The KdTree object stores measurement vectors in a k-d tree structure that is a binary tree. The partition value is the median value of one of the k dimension (partition dimension). The partition dimension is determined by the spread of measurement values in each dimension. The partition dimension is the dimension has the widest spread. Our implementation of k-d tree doesn't have any construction or insertion logic. Users should use this class or the WeightedCentroidKdTreeGenerator class.

The number of the measurement vectors in a terminal node is set by the SetBucketSize method. If we use too small number for this, it might cause computational overhead to calculate bound conditions. However, too large number will cause more distance calculation between the measurement vectors in a terminal node and the query point.

To run this generator, users should provides the bucket size (SetBucketSize method) and the input sample (SetSample method). The Update method will run this generator. To get the resulting KdTree object, call the GetOutput method.

**Recent API changes:** The static const macro to get the length of a measurement vector, 'MeasurementVectorSize' has been removed to allow the length of a measurement vector to be specified at run time. It is now

obtained from the sample set as input. You may query this length using the function GetMeasurementVectorSize().

**See** KdTree, KdTreeNode, KdTreeNonterminalNode, KdTreeTerminalNode, WeightedCentroidKdTreeGenerator

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Spatial Search

Subclassed by itk::Statistics::WeightedCentroidKdTreeGenerator< TSample >

See itk::Statistics::KdTreeGenerator for additional documentation.

## 3.8 Nonunit

### 3.8.1 Review

#### Geometric Properties of Labeled Region

#### Synopsis

Get geometric properties of labeled regions in an image.

#### Results

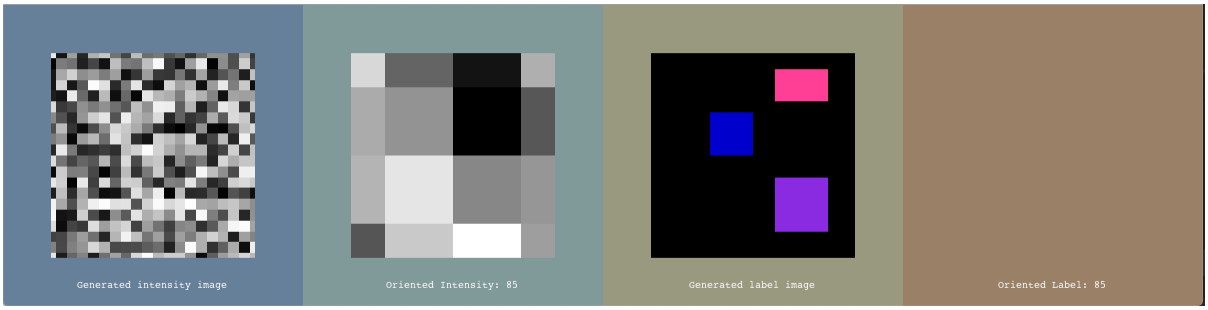



Fig. 347: Output In QuickView

Output:

```
Image (0x7fd274c5f3c0)
RTTI typeinfo:   itk::Image<unsigned int, 2u>
Reference Count: 1
Modified Time: 46
Debug: Off
Object Name:
Observers:
  none
Source: (none)
```

(continues on next page)

(continued from previous page)

```
Source output name: (none)
Release Data: Off
Data Released: False
Global Release Data: Off
PipelineMTime: 0
UpdateMTime: 0
RealTimeStamp: 0 seconds
LargestPossibleRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [20, 20]
BufferedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [20, 20]
RequestedRegion:
  Dimension: 2
  Index: [0, 0]
  Size: [20, 20]
Spacing: [1, 1]
Origin: [0, 0]
Direction:
1 0
0 1

IndexToPointMatrix:
1 0
0 1

PointToIndexMatrix:
1 0
0 1

Inverse Direction:
1 0
0 1

PixelContainer:
  ImportImageContainer (0x7fd274c5f5b0)
    RTTI typeinfo:   itk::ImportImageContainer<unsigned long, unsigned int>
    Reference Count: 1
    Modified Time: 47
    Debug: Off
    Object Name:
    Observers:
      none
    Pointer: 0x7fd27502d800
    Container manages memory: true
    Size: 400
    Capacity: 400
Number of labels: 4

Label: 0
Volume: 344
Integrated Intensity: 44128
Centroid: [9.06977, 9.54942]
Weighted Centroid: [9.44273, 9.96021]
```

(continues on next page)

(continued from previous page)

```
Axes Length: [23.6173, 23.9599]
MajorAxisLength: 23.9599
MinorAxisLength: 23.6173
Eccentricity: 0.168492
Elongation: 1.0145
Orientation: 2.74768
Bounding box: [0, 19, 0, 19]


Label: 85
Volume: 16
Integrated Intensity: 2544
Centroid: [7.5, 7.5]
Weighted Centroid: [7.44811, 7.47602]
Axes Length: [4.6188, 4.6188]
MajorAxisLength: 4.6188
MinorAxisLength: 4.6188
Eccentricity: 0
Elongation: 1
Orientation: 1.5708
Bounding box: [6, 9, 6, 9]

Label: 127
Volume: 25
Integrated Intensity: 3263
Centroid: [14, 14]
Weighted Centroid: [13.6883, 14.1192]
Axes Length: [5.7735, 5.7735]
MajorAxisLength: 5.7735
MinorAxisLength: 5.7735
Eccentricity: 0
Elongation: 1
Orientation: 1.5708
Bounding box: [12, 16, 12, 16]

Label: 191
Volume: 15
Integrated Intensity: 1840
Centroid: [14, 3]
Weighted Centroid: [13.8647, 3.10978]
Axes Length: [3.4641, 5.7735]
MajorAxisLength: 5.7735
MinorAxisLength: 3.4641
Eccentricity: 0.8
Elongation: 1.66667
Orientation: 0
Bounding box: [12, 16, 2, 4]
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkImageRegionIterator.h"
#include "itkImageFileReader.h"
#include "itkLabelGeometryImageFilter.h"
#include "itkLabelToRGBImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

#include <sstream>

using ImageType = itk::Image<unsigned int, 2>;
using RGBPixelType = itk::RGBPixel<unsigned char>;
using RGBImageType = itk::Image<RGBPixelType, 2>;

static void
CreateIntensityImage(ImageType::Pointer image);
static void
CreateLabelImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer labelImage = ImageType::New();
  ImageType::Pointer intensityImage = ImageType::New();
  int                label = 1;

  if (argc < 2)
  {
    // Create a label image that is 0 in the background and where the
    // objects are labeled
    CreateLabelImage(labelImage);
    labelImage->Print(std::cout);
    // Create an intensity image.  Some LabelGeometry features (such as
    // weighted centroid and integrated intensity) depend on an
    // intensity image.
    CreateIntensityImage(intensityImage);
  }
  else if (argc > 3)
  {
    using ImageReaderType = itk::ImageFileReader<ImageType>;
    ImageReaderType::Pointer labelReader = ImageReaderType::New();
    labelReader->SetFileName(argv[1]);
    labelReader->Update();

    labelImage = labelReader->GetOutput();

    ImageReaderType::Pointer intensityReader = ImageReaderType::New();
    intensityReader->SetFileName(argv[2]);
    intensityReader->Update();
```

(continues on next page)

(continued from previous page)

```
    intensityImage = intensityReader->GetOutput();

    label = std::stoi(argv[3]);
  }

  // NOTE: As of April 8, 2015 the filter does not work with non-zero
  // origins
  double origin[2] = { 0.0, 0.0 };
  labelImage->SetOrigin(origin);
  intensityImage->SetOrigin(origin);

  using LabelGeometryImageFilterType = itk::LabelGeometryImageFilter<ImageType>;
  LabelGeometryImageFilterType::Pointer labelGeometryImageFilter =␣
↪LabelGeometryImageFilterType::New();
  labelGeometryImageFilter->SetInput(labelImage);
  labelGeometryImageFilter->SetIntensityInput(intensityImage);

  // These generate optional outputs.
  labelGeometryImageFilter->CalculatePixelIndicesOn();
  labelGeometryImageFilter->CalculateOrientedBoundingBoxOn();
  labelGeometryImageFilter->CalculateOrientedLabelRegionsOn();
  labelGeometryImageFilter->CalculateOrientedIntensityRegionsOn();

  labelGeometryImageFilter->Update();
  LabelGeometryImageFilterType::LabelsType allLabels = labelGeometryImageFilter->
↪GetLabels();

  using RGBFilterType = itk::LabelToRGBImageFilter<ImageType, RGBImageType>;
  RGBFilterType::Pointer rgbLabelImage = RGBFilterType::New();
  rgbLabelImage->SetInput(labelImage);

  using RGBFilterType = itk::LabelToRGBImageFilter<ImageType, RGBImageType>;
  RGBFilterType::Pointer rgbOrientedImage = RGBFilterType::New();
  rgbOrientedImage->SetInput(labelGeometryImageFilter->
↪GetOrientedLabelImage(allLabels[label]));

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.ShareCameraOff();
  viewer.InterpolateOff();

  viewer.AddImage(rgbLabelImage->GetOutput(),
                  true,
                  argc > 3 ? itksys::SystemTools::GetFilenameName(argv[1]) :
↪"Generated label image");
  viewer.AddImage(intensityImage.GetPointer(),
                  true,
                  argc > 3 ? itksys::SystemTools::GetFilenameName(argv[2]) :
↪"Generated intensity image");

  std::stringstream desc;
  desc << "Oriented Label: " << allLabels[label];
  viewer.AddImage(rgbOrientedImage->GetOutput(), true, desc.str());

  std::stringstream desc2;
  desc2 << "Oriented Intensity: " << allLabels[label];
  viewer.AddImage(labelGeometryImageFilter->
↪GetOrientedIntensityImage(allLabels[label]), true, desc2.str());
```

(continues on next page)

(continued from previous page)

```
  viewer.Visualize();
#endif

  LabelGeometryImageFilterType::LabelsType::iterator allLabelsIt;
  std::cout << "Number of labels: " << labelGeometryImageFilter->GetNumberOfLabels() <
↪< std::endl;
  std::cout << std::endl;

  for (allLabelsIt = allLabels.begin(); allLabelsIt != allLabels.end(); allLabelsIt++)
  {
    LabelGeometryImageFilterType::LabelPixelType labelValue = *allLabelsIt;
    std::cout << "\tLabel: " << (int)labelValue << std::endl;
    std::cout << "\tVolume: " << labelGeometryImageFilter->GetVolume(labelValue) <<␣
↪std::endl;
    std::cout << "\tIntegrated Intensity: " << labelGeometryImageFilter->
↪GetIntegratedIntensity(labelValue)
              << std::endl;
    std::cout << "\tCentroid: " << labelGeometryImageFilter->GetCentroid(labelValue) <
↪< std::endl;
    std::cout << "\tWeighted Centroid: " << labelGeometryImageFilter->
↪GetWeightedCentroid(labelValue) << std::endl;
    std::cout << "\tAxes Length: " << labelGeometryImageFilter->
↪GetAxesLength(labelValue) << std::endl;
    std::cout << "\tMajorAxisLength: " << labelGeometryImageFilter->
↪GetMajorAxisLength(labelValue) << std::endl;
    std::cout << "\tMinorAxisLength: " << labelGeometryImageFilter->
↪GetMinorAxisLength(labelValue) << std::endl;
    std::cout << "\tEccentricity: " << labelGeometryImageFilter->
↪GetEccentricity(labelValue) << std::endl;
    std::cout << "\tElongation: " << labelGeometryImageFilter->
↪GetElongation(labelValue) << std::endl;
    std::cout << "\tOrientation: " << labelGeometryImageFilter->
↪GetOrientation(labelValue) << std::endl;
    std::cout << "\tBounding box: " << labelGeometryImageFilter->
↪GetBoundingBox(labelValue) << std::endl;

    std::cout << std::endl << std::endl;
  }

  return EXIT_SUCCESS;
}

void
CreateIntensityImage(ImageType::Pointer image)
{
  // Create a random image.
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();
```

(continues on next page)

(continued from previous page)

```
  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());
  // Make a random image
  // Create an unchanging seed.
  srand(1);
  while (!imageIterator.IsAtEnd())
  {
    int randomNumber = rand() % 255;
    imageIterator.Set(randomNumber);
    ++imageIterator;
  }
}

void
CreateLabelImage(ImageType::Pointer image)
{
  // Create a black image with labeled squares.
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(20);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);
  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, image->
↪GetLargestPossibleRegion());

  // Make a square
  while (!imageIterator.IsAtEnd())
  {
    if ((imageIterator.GetIndex()[0] > 5 && imageIterator.GetIndex()[0] < 10) &&
        (imageIterator.GetIndex()[1] > 5 && imageIterator.GetIndex()[1] < 10))
    {
      imageIterator.Set(85);
    }
    else if ((imageIterator.GetIndex()[0] > 11 && imageIterator.GetIndex()[0] < 17) &&
             (imageIterator.GetIndex()[1] > 11 && imageIterator.GetIndex()[1] < 17))
    {
      imageIterator.Set(127);
    }
    else if ((imageIterator.GetIndex()[0] > 11 && imageIterator.GetIndex()[0] < 17) &&
             (imageIterator.GetIndex()[1] > 1 && imageIterator.GetIndex()[1] < 5))
    {
      imageIterator.Set(191);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

### Classes demonstrated

template<typename **TLabelImage**, typename **TIntensityImage** = *TLabelImage*>
**class LabelGeometryImageFilter** : **public** itk::*ImageToImageFilter*<*TLabelImage*, *TIntensityImage*>

Given a label map and an optional intensity image, compute geometric features.

This filter enables the measurement of geometric features of all objects in a labeled ND volume. This labeled volume can represent, for instance, a medical image segmented into different anatomical structures or a microscope image segmented into individual cells. This filter is closely related to the itkLabelStatisticsImageFilter, which measures statistics of image regions defined by a labeled mask such as min, max, and mean intensity, intensity standard deviation, and bounding boxes. This filter, however, measures the geometry of the labeled regions themselves.

It calculates features similar to the regionprops command of Matlab. The set of measurements that it enables include: volume, centroid, eigenvalues, eigenvectors, axes lengths, eccentricity, elongation, orientation, bounding box, oriented bounding box, and rotation matrix. These features are based solely on the labeled mask itself. It also calculates integrated intensity and weighted centroid, which are measured on an intensity image under the labeled mask. These features represent the set of currently calculated features, but the framework of the filter is designed so that it can be easily expanded to measure a wide variety of other features.

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from http://commonfund.nih.gov/bioinformatics.

**Authors** Dirk Padfield and James Miller.

This filter was contributed in the Insight Journal paper: "A Label Geometry Image Filter for Multiple Object Measurement" by Padfield D., Miller J http://www.insight-journal.org/browse/publication/301

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Geometric Properties Of Labeled Region

See itk::LabelGeometryImageFilter for additional documentation.

### Multiphase Chan and Vese Sparse Field Level Set Segmentation

**Note: Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Multiphase Chan And Vese Sparse Field Level Set Segmentation.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkScalarChanAndVeseSparseLevelSetImageFilter.h"
#include "itkScalarChanAndVeseLevelSetFunction.h"
#include "itkScalarChanAndVeseLevelSetFunctionData.h"
#include "itkConstrainedRegionBasedLevelSetFunctionSharedData.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkAtanRegularizedHeavisideStepFunction.h"

int
main(int argc, char ** argv)
{
  if (argc < 10)
  {
    std::cerr << "Missing arguments" << std::endl;
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " inputLevelSetImage1 inputLevelSetImage2 inputLevelSetImage3";
    std::cerr << " inputFeatureImage outputLevelSetImage";
    std::cerr << " CurvatureWeight AreaWeight LaplacianWeight";
    std::cerr << " VolumeWeight Volume OverlapWeight" << std::endl;
    return EXIT_FAILURE;
  }

  unsigned int nb_iteration = 50;
  double       rms = 0.;
  double       epsilon = 1.5;
  double       curvature_weight = std::stod(argv[6]);
  double       area_weight = std::stod(argv[7]);
  double       volume_weight = std::stod(argv[8]);
  double       volume = std::stod(argv[9]);
  double       overlap_weight = std::stod(argv[10]);
  double       l1 = 1.;
  double       l2 = 1.;

  constexpr unsigned int Dimension = 2;
  using ScalarPixelType = float;
```

(continues on next page)

(continued from previous page)

```
  using LevelSetImageType = itk::Image<ScalarPixelType, Dimension>;
  using FeatureImageType = itk::Image<unsigned char, Dimension>;
  using OutputImageType = itk::Image<unsigned char, Dimension>;

  using DataHelperType = itk::ScalarChanAndVeseLevelSetFunctionData<LevelSetImageType,
↪ FeatureImageType>;

  using SharedDataHelperType =
    itk::ConstrainedRegionBasedLevelSetFunctionSharedData<LevelSetImageType,␣
↪FeatureImageType, DataHelperType>;

  using LevelSetFunctionType =
    itk::ScalarChanAndVeseLevelSetFunction<LevelSetImageType, FeatureImageType,␣
↪SharedDataHelperType>;

  using MultiLevelSetType = itk::ScalarChanAndVeseSparseLevelSetImageFilter
↪<LevelSetImageType,
                                                                              ␣
↪FeatureImageType,
                                                                              ␣
↪OutputImageType,
                                                                              ␣
↪LevelSetFunctionType,
                                                                              ␣
↪SharedDataHelperType>;

  using LevelSetReaderType = itk::ImageFileReader<LevelSetImageType>;
  using FeatureReaderType = itk::ImageFileReader<FeatureImageType>;
  using WriterType = itk::ImageFileWriter<OutputImageType>;

  using DomainFunctionType = itk::AtanRegularizedHeavisideStepFunction
↪<ScalarPixelType, ScalarPixelType>;

  DomainFunctionType::Pointer domainFunction = DomainFunctionType::New();
  domainFunction->SetEpsilon(epsilon);

  LevelSetReaderType::Pointer contourReader1 = LevelSetReaderType::New();
  contourReader1->SetFileName(argv[1]);
  contourReader1->Update();

  LevelSetReaderType::Pointer contourReader2 = LevelSetReaderType::New();
  contourReader2->SetFileName(argv[2]);
  contourReader2->Update();

  LevelSetReaderType::Pointer contourReader3 = LevelSetReaderType::New();
  contourReader3->SetFileName(argv[3]);
  contourReader3->Update();

  FeatureReaderType::Pointer featureReader = FeatureReaderType::New();
  featureReader->SetFileName(argv[4]);
  featureReader->Update();

  FeatureImageType::Pointer  featureImage = featureReader->GetOutput();
  LevelSetImageType::Pointer contourImage1 = contourReader1->GetOutput();
  LevelSetImageType::Pointer contourImage2 = contourReader2->GetOutput();
  LevelSetImageType::Pointer contourImage3 = contourReader3->GetOutput();
```

(continues on next page)

(continued from previous page)

```
  MultiLevelSetType::Pointer levelSetFilter = MultiLevelSetType::New();
  levelSetFilter->SetFunctionCount(3);
  levelSetFilter->SetFeatureImage(featureImage);
  levelSetFilter->SetLevelSet(0, contourImage1);
  levelSetFilter->SetLevelSet(1, contourImage2);
  levelSetFilter->SetLevelSet(2, contourImage3);
  levelSetFilter->SetNumberOfIterations(nb_iteration);
  levelSetFilter->SetMaximumRMSError(rms);
  levelSetFilter->SetUseImageSpacing(1);
  levelSetFilter->SetInPlace(false);

  for (unsigned int i = 0; i < 3; i++)
  {
    levelSetFilter->GetDifferenceFunction(i)->SetDomainFunction(domainFunction);
    levelSetFilter->GetDifferenceFunction(i)->SetCurvatureWeight(curvature_weight);
    levelSetFilter->GetDifferenceFunction(i)->SetAreaWeight(area_weight);
    levelSetFilter->GetDifferenceFunction(i)->SetOverlapPenaltyWeight(overlap_weight);
    levelSetFilter->GetDifferenceFunction(i)->SetVolumeMatchingWeight(volume_weight);
    levelSetFilter->GetDifferenceFunction(i)->SetVolume(volume);
    levelSetFilter->GetDifferenceFunction(i)->SetLambda1(l1);
    levelSetFilter->GetDifferenceFunction(i)->SetLambda2(l2);
  }

  levelSetFilter->Update();

  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(levelSetFilter->GetOutput());
  writer->SetFileName(argv[5]);

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & excep)
  {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
    return -1;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TFeatureImage**, typename **TOutputImage**, typename **TFunction** = *ScalarCha*
**class ScalarChanAndVeseSparseLevelSetImageFilter** : **public** itk::MultiphaseSparseFiniteDifferenceImageFilter<

Sparse implementation of the Chan and Vese multiphase level set image filter.

This code was adapted from the paper:

```
"An active contour model without edges"
 T. Chan and L. Vese.
 In Scale-Space Theories in Computer Vision, pages 141-151, 1999.
```

This code was taken from the Insight Journal paper:

```
"Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes"
http://www.insight-journal.org/browse/publication/642
https://hdl.handle.net/10380/3055
```

**Author** Mosaliganti K., Smith B., Gelas A., Gouaillard A., Megason S.

That is based on the papers:

```
"Level Set Segmentation: Active Contours without edge"
http://www.insight-journal.org/browse/publication/322
https://hdl.handle.net/1926/1532

and

"Level set segmentation using coupled active surfaces"
http://www.insight-journal.org/browse/publication/323
https://hdl.handle.net/1926/1533
```

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiphase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Dense Field Level Set Segmentation

See itk::ScalarChanAndVeseSparseLevelSetImageFilter for additional documentation.

template<typename **TInputImage**, typename **TFeatureImage**, typename **TSharedData** = ConstrainedRegionBasedLevelSetFu
**class ScalarChanAndVeseLevelSetFunction** : **public** itk::ScalarRegionBasedLevelSetFunction<*TInputImage*, *TFeature*

LevelSet function that computes a speed image based on regional integrals of probabilities.

This class implements a level set function that computes the speed image by integrating values on the image domain.

Based on the papers:

```
  "An active contour model without edges"
   T. Chan and L. Vese.
   In Scale-Space Theories in Computer Vision, pages 141-151, 1999.

   "Segmenting and Tracking Fluorescent Cells in Dynamic 3-D
    Microscopy With Coupled Active Surfaces"
    Dufour, Shinin, Tajbakhsh, Guillen-Aghion, Olivo-Marin
    In IEEE Transactions on Image Processing, vol. 14, No 9, Sep. 2005
```

**Author** Mosaliganti K., Smith B., Gelas A., Gouaillard A., Megason S.

This code was taken from the Insight Journal paper:

```
"Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes"
http://www.insight-journal.org/browse/publication/642
https://hdl.handle.net/10380/3055
```

That is based on the papers:

```
"Level Set Segmentation: Active Contours without edge"
http://www.insight-journal.org/browse/publication/322
https://hdl.handle.net/1926/1532

and

"Level set segmentation using coupled active surfaces"
http://www.insight-journal.org/browse/publication/323
https://hdl.handle.net/1926/1533
```

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiphase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Dense Field Level Set Segmentation

See itk::ScalarChanAndVeseLevelSetFunction for additional documentation.

### Segment Blood Vessels With Multi-Scale Hessian-Based Measure

### Synopsis

Segment blood vessels with multi-scale Hessian-based measure.

### Results

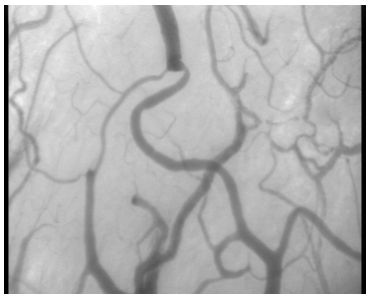


Fig. 348: Input image

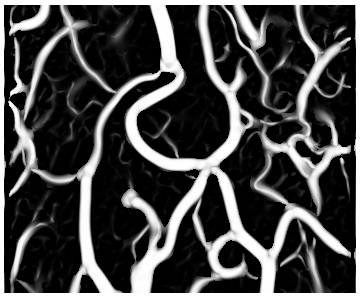


Fig. 349: Output image

### Code

### Python

```
#!/usr/bin/env python

import argparse

import itk
from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("5.0.0"):
    print("ITK 5.0.0 or newer is required.")
    sys.exit(1)

parser = argparse.ArgumentParser(
    description="Segment blood vessels with multi-scale Hessian-based measure."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("--sigma_minimum", type=float, default=1.0)
parser.add_argument("--sigma_maximum", type=float, default=10.0)
parser.add_argument("--number_of_sigma_steps", type=int, default=10)
args = parser.parse_args()

input_image = itk.imread(args.input_image, itk.F)

ImageType = type(input_image)
Dimension = input_image.GetImageDimension()
HessianPixelType = itk.SymmetricSecondRankTensor[itk.D, Dimension]
HessianImageType = itk.Image[HessianPixelType, Dimension]

objectness_filter = itk.HessianToObjectnessMeasureImageFilter[
    HessianImageType, ImageType
].New()
objectness_filter.SetBrightObject(False)
objectness_filter.SetScaleObjectnessMeasure(False)
objectness_filter.SetAlpha(0.5)
objectness_filter.SetBeta(1.0)
objectness_filter.SetGamma(5.0)

multi_scale_filter = itk.MultiScaleHessianBasedMeasureImageFilter[
    ImageType, HessianImageType, ImageType
```

(continues on next page)

(continued from previous page)

```
].New()
multi_scale_filter.SetInput(input_image)
multi_scale_filter.SetHessianToMeasureFilter(objectness_filter)
multi_scale_filter.SetSigmaStepMethodToLogarithmic()
multi_scale_filter.SetSigmaMinimum(args.sigma_minimum)
multi_scale_filter.SetSigmaMaximum(args.sigma_maximum)
multi_scale_filter.SetNumberOfSigmaSteps(args.number_of_sigma_steps)

OutputPixelType = itk.UC
OutputImageType = itk.Image[OutputPixelType, Dimension]

rescale_filter = itk.RescaleIntensityImageFilter[ImageType, OutputImageType].New()
rescale_filter.SetInput(multi_scale_filter)

itk.imwrite(rescale_filter.GetOutput(), args.output_image)
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkHessianToObjectnessMeasureImageFilter.h"
#include "itkMultiScaleHessianBasedMeasureImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc < 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName>";
    std::cerr << " [SigmaMinimum] [SigmaMaximum]";
    std::cerr << " [NumberOfSigmaSteps]";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  double       sigmaMinimum = 1.0;
  if (argc > 3)
  {
    sigmaMinimum = std::stod(argv[3]);
  }
  double sigmaMaximum = 10.0;
  if (argc > 4)
  {
    sigmaMaximum = std::stod(argv[4]);
  }
  unsigned int numberOfSigmaSteps = 10;
  if (argc > 5)
  {
    numberOfSigmaSteps = std::stoi(argv[5]);
```

(continues on next page)

(continued from previous page)

```
  }

  constexpr unsigned int Dimension = 2;

  using PixelType = float;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using HessianPixelType = itk::SymmetricSecondRankTensor<double, Dimension>;
  using HessianImageType = itk::Image<HessianPixelType, Dimension>;
  using ObjectnessFilterType = itk::HessianToObjectnessMeasureImageFilter
↪<HessianImageType, ImageType>;
  ObjectnessFilterType::Pointer objectnessFilter = ObjectnessFilterType::New();
  objectnessFilter->SetBrightObject(false);
  objectnessFilter->SetScaleObjectnessMeasure(false);
  objectnessFilter->SetAlpha(0.5);
  objectnessFilter->SetBeta(1.0);
  objectnessFilter->SetGamma(5.0);

  using MultiScaleEnhancementFilterType =
    itk::MultiScaleHessianBasedMeasureImageFilter<ImageType, HessianImageType,␣
↪ImageType>;
  MultiScaleEnhancementFilterType::Pointer multiScaleEnhancementFilter =␣
↪MultiScaleEnhancementFilterType::New();
  multiScaleEnhancementFilter->SetInput(reader->GetOutput());
  multiScaleEnhancementFilter->SetHessianToMeasureFilter(objectnessFilter);
  multiScaleEnhancementFilter->SetSigmaStepMethodToLogarithmic();
  multiScaleEnhancementFilter->SetSigmaMinimum(sigmaMinimum);
  multiScaleEnhancementFilter->SetSigmaMaximum(sigmaMaximum);
  multiScaleEnhancementFilter->SetNumberOfSigmaSteps(numberOfSigmaSteps);

  using OutputImageType = itk::Image<unsigned char, Dimension>;
  using RescaleFilterType = itk::RescaleIntensityImageFilter<ImageType,␣
↪OutputImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(multiScaleEnhancementFilter->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(rescaleFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **THessianImage**, typename **TOutputImage** = *TInputImage*>
**class MultiScaleHessianBasedMeasureImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*

A filter to enhance structures using Hessian eigensystem-based measures in a multiscale framework.

The filter evaluates a Hessian-based enhancement measure, such as vesselness or objectness, at different scale levels. The Hessian-based measure is computed from the Hessian image at each scale level and the best response is selected.

Minimum and maximum sigma value can be set using SetMinSigma and SetMaxSigma methods respectively. The number of scale levels is set using SetNumberOfSigmaSteps method. Exponentially distributed scale levels are computed within the bound set by the minimum and maximum sigma values

The filter computes a second output image (accessed by the GetScalesOutput method) containing the scales at which each pixel gave the best response.

This code was contributed in the Insight Journal paper: "Generalizing vesselness with respect to dimensionality and shape" by Antiga L. https://www.insight-journal.org/browse/publication/175

**Author** Luca Antiga Ph.D. Medical Imaging Unit, Bioengineering Department, Mario Negri Institute, Italy.

**See** HessianToObjectnessMeasureImageFilter

**See** Hessian3DToVesselnessMeasureImageFilter

**See** HessianSmoothed3DToVesselnessMeasureImageFilter

**See** HessianRecursiveGaussianImageFilter

**See** SymmetricEigenAnalysisImageFilter

**See** SymmetricSecondRankTensor

See itk::MultiScaleHessianBasedMeasureImageFilter for additional documentation.

### Singlephase Chan and Vese Dense Field Level Set Segmentation

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Single-phase Chan And Vese Dense Field Level Set Segmentation

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
// The use of the ScalarChanAndVeseDenseLevelSetImageFilter is
// illustrated in the following example. The implementation of this filter in
// ITK is based on the paper by Chan And Vese.  This
// implementation extends the functionality of the
// level-set filters in ITK by using region-based variational techniques. These␣
↪methods
// do not rely on the presence of edges in the images.
//
// ScalarChanAndVeseDenseLevelSetImageFilter expects two inputs.  The first is
// an initial level set in the form of an \doxygen{Image}. The second input
// is a feature image. For this algorithm, the feature image is the original
// raw or preprocessed image. Several parameters are required by the algorithm
// for regularization and weights of different energy terms. The user is encouraged to
// change different parameter settings to optimize the code example on their images.
//
// Let's start by including the headers of the main filters involved in the
// preprocessing.
//

#include "itkScalarChanAndVeseDenseLevelSetImageFilter.h"
#include "itkScalarChanAndVeseLevelSetFunctionData.h"
#include "itkConstrainedRegionBasedLevelSetFunctionSharedData.h"
#include "itkFastMarchingImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkAtanRegularizedHeavisideStepFunction.h"


int
main(int argc, char ** argv)
{

  if (argc < 6)
  {
    std::cerr << "Missing arguments" << std::endl;
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " featureImage outputImage";
    std::cerr << " startx starty seedValue" << std::endl;
    return EXIT_FAILURE;
  }

  unsigned int nb_iteration = 500;
  double       rms = 0.;
  double       epsilon = 1.;
  double       curvature_weight = 0.;
  double       area_weight = 0.;
  double       reinitialization_weight = 0.;
  double       volume_weight = 0.;
  double       volume = 0.;
  double       l1 = 1.;
  double       l2 = 1.;
```

(continues on next page)

(continued from previous page)

```
  //
  //  We now define the image type using a particular pixel type and
  //  dimension. In this case the \code{float} type is used for the pixels
  //  due to the requirements of the smoothing filter.
  //
  constexpr unsigned int Dimension = 2;
  using ScalarPixelType = float;
  using InternalImageType = itk::Image<ScalarPixelType, Dimension>;

  using DataHelperType = itk::ScalarChanAndVeseLevelSetFunctionData<InternalImageType,
↪ InternalImageType>;

  using SharedDataHelperType =
    itk::ConstrainedRegionBasedLevelSetFunctionSharedData<InternalImageType,␣
↪InternalImageType, DataHelperType>;

  using LevelSetFunctionType =
    itk::ScalarChanAndVeseLevelSetFunction<InternalImageType, InternalImageType,␣
↪SharedDataHelperType>;


  //  We declare now the type of the numerically discretized Step and Delta functions␣
↪that
  //  will be used in the level-set computations for foreground and background regions
  //
  using DomainFunctionType = itk::AtanRegularizedHeavisideStepFunction
↪<ScalarPixelType, ScalarPixelType>;

  DomainFunctionType::Pointer domainFunction = DomainFunctionType::New();
  domainFunction->SetEpsilon(epsilon);

  // We instantiate reader and writer types in the following lines.
  //
  using ReaderType = itk::ImageFileReader<InternalImageType>;
  using WriterType = itk::ImageFileWriter<InternalImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  WriterType::Pointer writer = WriterType::New();

  reader->SetFileName(argv[1]);
  reader->Update();

  writer->SetFileName(argv[2]);

  InternalImageType::Pointer featureImage = reader->GetOutput();

  //  We declare now the type of the FastMarchingImageFilter that
  //  will be used to generate the initial level set in the form of a distance
  //  map.
  //
  using FastMarchingFilterType = itk::FastMarchingImageFilter<InternalImageType,␣
↪InternalImageType>;

  FastMarchingFilterType::Pointer fastMarching = FastMarchingFilterType::New();


  //  The FastMarchingImageFilter requires the user to provide a seed
```

(continues on next page)

(continued from previous page)

```
//  point from which the level set will be generated. The user can actually
//  pass not only one seed point but a set of them. Note the the
//  FastMarchingImageFilter is used here only as a helper in the
//  determination of an initial level set. We could have used the
//  \doxygen{DanielssonDistanceMapImageFilter} in the same way.
//
//  The seeds are passed stored in a container. The type of this
//  container is defined as \code{NodeContainer} among the
//  FastMarchingImageFilter traits.
//
using NodeContainer = FastMarchingFilterType::NodeContainer;
using NodeType = FastMarchingFilterType::NodeType;

NodeContainer::Pointer seeds = NodeContainer::New();

InternalImageType::IndexType seedPosition;

seedPosition[0] = std::stoi(argv[3]);
seedPosition[1] = std::stoi(argv[4]);

const double initialDistance = std::stod(argv[5]);

NodeType node;

const double seedValue = -initialDistance;

node.SetValue(seedValue);
node.SetIndex(seedPosition);

//  The list of nodes is initialized and then every node is inserted using
//  the \code{InsertElement()}.
//
seeds->Initialize();
seeds->InsertElement(0, node);


//  The set of seed nodes is passed now to the
//  FastMarchingImageFilter with the method
//  \code{SetTrialPoints()}.
//
fastMarching->SetTrialPoints(seeds);


//  Since the FastMarchingImageFilter is used here just as a
//  Distance Map generator. It does not require a speed image as input.
//  Instead the constant value $1.0$ is passed using the
//  \code{SetSpeedConstant()} method.
//
fastMarching->SetSpeedConstant(1.0);

//  The FastMarchingImageFilter requires the user to specify the
//  size of the image to be produced as output. This is done using the
//  \code{SetOutputSize()}. Note that the size is obtained here from the
//  output image of the smoothing filter. The size of this image is valid
//  only after the \code{Update()} methods of this filter has been called
//  directly or indirectly.
//
```

(continues on next page)

(continued from previous page)

```
  fastMarching->SetOutputSize(featureImage->GetBufferedRegion().GetSize());
  fastMarching->Update();

  //  We declare now the type of the ScalarChanAndVeseDenseLevelSetImageFilter that
  //  will be used to generate a segmentation.
  //

  using MultiLevelSetType = itk::ScalarChanAndVeseDenseLevelSetImageFilter
↪<InternalImageType,
                                                                              ␣
↪InternalImageType,
                                                                              ␣
↪InternalImageType,
                                                                              ␣
↪LevelSetFunctionType,
                                                                              ␣
↪SharedDataHelperType>;

  MultiLevelSetType::Pointer levelSetFilter = MultiLevelSetType::New();

  //  We set the function count to 1 since a single level-set is being evolved.
  //
  levelSetFilter->SetFunctionCount(1);

  //  Set the feature image and initial level-set image as output of the
  //  fast marching image filter.
  //
  levelSetFilter->SetFeatureImage(featureImage);
  levelSetFilter->SetLevelSet(0, fastMarching->GetOutput());

  //  Once activiated the level set evolution will stop if the convergence
  //  criteria or if the maximum number of iterations is reached.  The
  //  convergence criteria is defined in terms of the root mean squared (RMS)
  //  change in the level set function. The evolution is said to have
  //  converged if the RMS change is below a user specified threshold.  In a
  //  real application is desirable to couple the evolution of the zero set
  //  to a visualization module allowing the user to follow the evolution of
  //  the zero set. With this feedback, the user may decide when to stop the
  //  algorithm before the zero set leaks through the regions of low gradient
  //  in the contour of the anatomical structure to be segmented.
  //
  levelSetFilter->SetNumberOfIterations(nb_iteration);
  levelSetFilter->SetMaximumRMSError(rms);

  //  Often, in real applications, images have different pixel resolutions. In such
  //  cases, it is best to use the native spacings to compute derivatives etc rather
  //  than sampling the images.
  //
  levelSetFilter->SetUseImageSpacing(1);

  //  For large images, we may want to compute the level-set over the initial supplied
  //  level-set image. This saves a lot of memory.
  //
  levelSetFilter->SetInPlace(false);

  //  For the level set with phase 0, set different parameters and weights. This may
  //  to be set in a loop for the case of multiple level-sets evolving simultaneously.
```

(continues on next page)

(continued from previous page)

```
  //
  levelSetFilter->GetDifferenceFunction(0)->SetDomainFunction(domainFunction);
  levelSetFilter->GetDifferenceFunction(0)->SetCurvatureWeight(curvature_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetAreaWeight(area_weight);
  levelSetFilter->GetDifferenceFunction(0)->
↪SetReinitializationSmoothingWeight(reinitialization_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetVolumeMatchingWeight(volume_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetVolume(volume);
  levelSetFilter->GetDifferenceFunction(0)->SetLambda1(l1);
  levelSetFilter->GetDifferenceFunction(0)->SetLambda2(l2);

  levelSetFilter->Update();


  writer->SetInput(levelSetFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & excep)
  {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
    return -1;
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TFeatureImage**, typename **TOutputImage**, typename **TFunction** = *ScalarCha*
**class ScalarChanAndVeseDenseLevelSetImageFilter** : **public** itk::MultiphaseDenseFiniteDifferenceImageFilter<*TI*

Dense implementation of the Chan and Vese multiphase level set image filter.

This code was adapted from the paper:

```
"An active contour model without edges"
 T. Chan and L. Vese.
 In Scale-Space Theories in Computer Vision, pages 141-151, 1999.
```

This code was taken from the Insight Journal paper:

```
"Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes"
http://www.insight-journal.org/browse/publication/642
https://hdl.handle.net/10380/3055
```

**Author** Mosaliganti K., Smith B., Gelas A., Gouaillard A., Megason S.

That is based on the papers:

```
"Level Set Segmentation: Active Contours without edge"
http://www.insight-journal.org/browse/publication/322
```

(continues on next page)

(continued from previous page)

```
https://hdl.handle.net/1926/1532

and

"Level set segmentation using coupled active surfaces"
http://www.insight-journal.org/browse/publication/323
https://hdl.handle.net/1926/1533
```

See itk::ScalarChanAndVeseDenseLevelSetImageFilter for additional documentation.

### Singlephase Chan and Vese Sparse Field Level Set Segmentation

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Single-phase Chan And Vese Sparse Field Level Set Segmentation.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
// The use of the ScalarChanAndVeseSparseLevelSetImageFilter is
// illustrated in the following example. The implementation of this filter in
// ITK is based on the paper by Chan And Vese.  This
// implementation extends the functionality of the
// level-set filters in ITK by using region-based variational techniques. These␣
↪methods
// do not rely on the presence of edges in the images.
//
// ScalarChanAndVeseSparseLevelSetImageFilter expects two inputs.  The first is
// an initial level set in the form of an \doxygen{Image}. The second input
// is a feature image. For this algorithm, the feature image is the original
// raw or preprocessed image. Several parameters are required by the algorithm
// for regularization and weights of different energy terms. The user is encouraged to
// change different parameter settings to optimize the code example on their images.
//
// Let's start by including the headers of the main filters involved in the
// preprocessing.
//
```

(continues on next page)

(continued from previous page)

```
#include "itkScalarChanAndVeseSparseLevelSetImageFilter.h"
#include "itkScalarChanAndVeseLevelSetFunctionData.h"
#include "itkConstrainedRegionBasedLevelSetFunctionSharedData.h"
#include "itkFastMarchingImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkAtanRegularizedHeavisideStepFunction.h"

int
main(int argc, char ** argv)
{
  if (argc < 6)
  {
    std::cerr << "Missing arguments" << std::endl;
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0] << " featureImage outputImage";
    std::cerr << " startx starty seedValue" << std::endl;
    return EXIT_FAILURE;
  }

  unsigned int nb_iteration = 500;
  double       rms = 0.;
  double       epsilon = 1.;
  double       curvature_weight = 0.;
  double       area_weight = 0.;
  double       reinitialization_weight = 0.;
  double       volume_weight = 0.;
  double       volume = 0.;
  double       l1 = 1.;
  double       l2 = 1.;

  //
  //  We now define the image type using a particular pixel type and
  //  dimension. In this case the \code{float} type is used for the pixels
  //  due to the requirements of the smoothing filter.
  //
  constexpr unsigned int Dimension = 2;
  using ScalarPixelType = float;
  using InternalImageType = itk::Image<ScalarPixelType, Dimension>;


  using DataHelperType = itk::ScalarChanAndVeseLevelSetFunctionData<InternalImageType,
↪ InternalImageType>;

  using SharedDataHelperType =
    itk::ConstrainedRegionBasedLevelSetFunctionSharedData<InternalImageType,␣
↪InternalImageType, DataHelperType>;

  using LevelSetFunctionType =
    itk::ScalarChanAndVeseLevelSetFunction<InternalImageType, InternalImageType,␣
↪SharedDataHelperType>;


  //  We declare now the type of the numerically discretized Step and Delta functions␣
↪that
```

(continues on next page)

(continued from previous page)

```
  //  will be used in the level-set computations for foreground and background regions
  //
  using DomainFunctionType = itk::AtanRegularizedHeavisideStepFunction
↪<ScalarPixelType, ScalarPixelType>;

  DomainFunctionType::Pointer domainFunction = DomainFunctionType::New();
  domainFunction->SetEpsilon(epsilon);

  // We instantiate reader and writer types in the following lines.
  //
  using ReaderType = itk::ImageFileReader<InternalImageType>;
  using WriterType = itk::ImageFileWriter<InternalImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  WriterType::Pointer writer = WriterType::New();

  reader->SetFileName(argv[1]);
  reader->Update();

  writer->SetFileName(argv[2]);

  InternalImageType::Pointer featureImage = reader->GetOutput();

  //  We declare now the type of the FastMarchingImageFilter that
  //  will be used to generate the initial level set in the form of a distance
  //  map.
  //
  using FastMarchingFilterType = itk::FastMarchingImageFilter<InternalImageType,␣
↪InternalImageType>;

  FastMarchingFilterType::Pointer fastMarching = FastMarchingFilterType::New();

  //  The FastMarchingImageFilter requires the user to provide a seed
  //  point from which the level set will be generated. The user can actually
  //  pass not only one seed point but a set of them. Note the the
  //  FastMarchingImageFilter is used here only as a helper in the
  //  determination of an initial level set. We could have used the
  //  \doxygen{DanielssonDistanceMapImageFilter} in the same way.
  //
  //  The seeds are passed stored in a container. The type of this
  //  container is defined as \code{NodeContainer} among the
  //  FastMarchingImageFilter traits.
  //
  using NodeContainer = FastMarchingFilterType::NodeContainer;
  using NodeType = FastMarchingFilterType::NodeType;

  NodeContainer::Pointer seeds = NodeContainer::New();

  InternalImageType::IndexType seedPosition;

  seedPosition[0] = std::stoi(argv[3]);
  seedPosition[1] = std::stoi(argv[4]);

  const double initialDistance = std::stod(argv[5]);

  NodeType node;
```

(continues on next page)

(continued from previous page)

```
  const double seedValue = -initialDistance;

  node.SetValue(seedValue);
  node.SetIndex(seedPosition);

  //  The list of nodes is initialized and then every node is inserted using
  //  the \code{InsertElement()}.
  //
  seeds->Initialize();
  seeds->InsertElement(0, node);


  //  The set of seed nodes is passed now to the
  //  FastMarchingImageFilter with the method
  //  \code{SetTrialPoints()}.
  //
  fastMarching->SetTrialPoints(seeds);


  //  Since the FastMarchingImageFilter is used here just as a
  //  Distance Map generator. It does not require a speed image as input.
  //  Instead the constant value $1.0$ is passed using the
  //  \code{SetSpeedConstant()} method.
  //
  fastMarching->SetSpeedConstant(1.0);

  //  The FastMarchingImageFilter requires the user to specify the
  //  size of the image to be produced as output. This is done using the
  //  \code{SetOutputSize()}. Note that the size is obtained here from the
  //  output image of the smoothing filter. The size of this image is valid
  //  only after the \code{Update()} methods of this filter has been called
  //  directly or indirectly.
  //
  fastMarching->SetOutputSize(featureImage->GetBufferedRegion().GetSize());
  fastMarching->Update();

  //  We declare now the type of the ScalarChanAndVeseSparseLevelSetImageFilter that
  //  will be used to generate a segmentation.
  //

  using MultiLevelSetType = itk::ScalarChanAndVeseSparseLevelSetImageFilter
↪<InternalImageType,

↪InternalImageType,

↪InternalImageType,

↪LevelSetFunctionType,

↪SharedDataHelperType>;

  MultiLevelSetType::Pointer levelSetFilter = MultiLevelSetType::New();

  //  We set the function count to 1 since a single level-set is being evolved.
  //
  levelSetFilter->SetFunctionCount(1);
```

(continues on next page)

(continued from previous page)

```
  //  Set the feature image and initial level-set image as output of the
  //  fast marching image filter.
  //
  levelSetFilter->SetFeatureImage(featureImage);
  levelSetFilter->SetLevelSet(0, fastMarching->GetOutput());

  //  Once activiated the level set evolution will stop if the convergence
  //  criteria or if the maximum number of iterations is reached.  The
  //  convergence criteria is defined in terms of the root mean squared (RMS)
  //  change in the level set function. The evolution is said to have
  //  converged if the RMS change is below a user specified threshold.  In a
  //  real application is desirable to couple the evolution of the zero set
  //  to a visualization module allowing the user to follow the evolution of
  //  the zero set. With this feedback, the user may decide when to stop the
  //  algorithm before the zero set leaks through the regions of low gradient
  //  in the contour of the anatomical structure to be segmented.
  //
  levelSetFilter->SetNumberOfIterations(nb_iteration);
  levelSetFilter->SetMaximumRMSError(rms);

  //  Often, in real applications, images have different pixel resolutions. In such
  //  cases, it is best to use the native spacings to compute derivatives etc rather
  //  than sampling the images.
  //
  levelSetFilter->SetUseImageSpacing(1);

  //  For large images, we may want to compute the level-set over the initial supplied
  //  level-set image. This saves a lot of memory.
  //
  levelSetFilter->SetInPlace(false);

  //  For the level set with phase 0, set different parameters and weights. This may
  //  to be set in a loop for the case of multiple level-sets evolving simultaneously.
  //
  levelSetFilter->GetDifferenceFunction(0)->SetDomainFunction(domainFunction);
  levelSetFilter->GetDifferenceFunction(0)->SetCurvatureWeight(curvature_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetAreaWeight(area_weight);
  levelSetFilter->GetDifferenceFunction(0)->
↪SetReinitializationSmoothingWeight(reinitialization_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetVolumeMatchingWeight(volume_weight);
  levelSetFilter->GetDifferenceFunction(0)->SetVolume(volume);
  levelSetFilter->GetDifferenceFunction(0)->SetLambda1(l1);
  levelSetFilter->GetDifferenceFunction(0)->SetLambda2(l2);

  levelSetFilter->Update();

  writer->SetInput(levelSetFilter->GetOutput());

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & excep)
  {
    std::cerr << "Exception caught !" << std::endl;
    std::cerr << excep << std::endl;
    return -1;
```

(continues on next page)

(continued from previous page)

```
  }

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TFeatureImage**, typename **TOutputImage**, typename **TFunction** = *ScalarCha*
**class ScalarChanAndVeseSparseLevelSetImageFilter** : **public** itk::MultiphaseSparseFiniteDifferenceImageFilter<

Sparse implementation of the Chan and Vese multiphase level set image filter.

This code was adapted from the paper:

```
"An active contour model without edges"
 T. Chan and L. Vese.
 In Scale-Space Theories in Computer Vision, pages 141-151, 1999.
```

This code was taken from the Insight Journal paper:

```
"Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes"
http://www.insight-journal.org/browse/publication/642
https://hdl.handle.net/10380/3055
```

**Author** Mosaliganti K., Smith B., Gelas A., Gouaillard A., Megason S.

That is based on the papers:

```
"Level Set Segmentation: Active Contours without edge"
http://www.insight-journal.org/browse/publication/322
https://hdl.handle.net/1926/1532

and

"Level set segmentation using coupled active surfaces"
http://www.insight-journal.org/browse/publication/323
https://hdl.handle.net/1926/1533
```

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiphase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Sparse Field Level Set Segmentation
- Single-phase Chan And Vese Dense Field Level Set Segmentation

See itk::ScalarChanAndVeseSparseLevelSetImageFilter for additional documentation.

# 3.9 Registration

## 3.9.1 Common

### Compute Mean Squares Metric Between Two Images

### Synopsis

Compute the mean squares metric between two images.

### Results

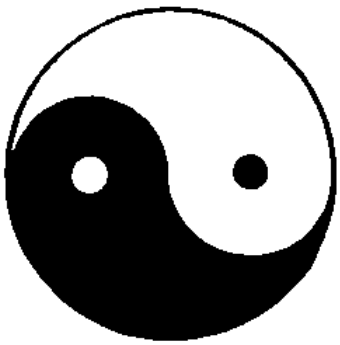


Fig. 350: Input image 1.

Data from 2 images passed through Output:

```
[-10, -10]: 23101.7
[-10, -5]: 23205.7
[-10, 0]: 23260.4
[-10, 5]: 23064.5
[-10, 10]: 22914.5
[-5, -10]: 23271.1
[-5, -5]: 23351.3
[-5, 0]: 23401
[-5, 5]: 23185.1
[-5, 10]: 23026.5
[0, -10]: 23486.5
[0, -5]: 23538.2
[0, 0]: 23566.2
[0, 5]: 23352.1
[0, 10]: 23175.2
[5, -10]: 23590.7
[5, -5]: 23625.7
[5, 0]: 23633.4
[5, 5]: 23401.1
[5, 10]: 23196.7
```

(continues on next page)

Fig. 351: Input image 2.

(continued from previous page)

```
[10, -10]: 23723.5
[10, -5]: 23762.9
[10, 0]: 23767.1
[10, 5]: 23504.9
[10, 10]: 23298.3
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkMeanSquaresImageToImageMetric.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkTranslationTransform.h"

int
main(int argc, char * argv[])
{
  using ImageType = itk::Image<double, 2>;
  using ReaderType = itk::ImageFileReader<ImageType>;

  if (argc < 3)
  {
    std::cout << "Usage: " << argv[0] << " imageFile1 imageFile2" << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  ReaderType::Pointer fixedReader = ReaderType::New();
  fixedReader->SetFileName(argv[1]);
  fixedReader->Update();

  ReaderType::Pointer movingReader = ReaderType::New();
  movingReader->SetFileName(argv[2]);
  movingReader->Update();

  ImageType::Pointer fixedImage = fixedReader->GetOutput();
  ImageType::Pointer movingImage = movingReader->GetOutput();

  using MetricType = itk::MeanSquaresImageToImageMetric<ImageType, ImageType>;
  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;
  using TransformType = itk::TranslationTransform<double, 2>;

  MetricType::Pointer    metric = MetricType::New();
  TransformType::Pointer transform = TransformType::New();

  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  interpolator->SetInputImage(fixedImage);

  metric->SetFixedImage(fixedImage);
  metric->SetMovingImage(movingImage);
  metric->SetFixedImageRegion(fixedImage->GetLargestPossibleRegion());
  metric->SetTransform(transform);
  metric->SetInterpolator(interpolator);

  TransformType::ParametersType params(transform->GetNumberOfParameters());
  params.Fill(0.0);

  metric->Initialize();
  for (double x = -10.0; x <= 10.0; x += 5.0)
  {
    params(0) = x;
    for (double y = -10.0; y <= 10.0; y += 5.0)
    {
      params(1) = y;
      std::cout << params << ": " << metric->GetValue(params) << std::endl;
    }
  }

  return EXIT_SUCCESS;
}
```

Python . . .

```
import itk
import argparse

parser = argparse.ArgumentParser(description="Compute Mean Square Between Two Images.
↪")
parser.add_argument("input_image_1")
parser.add_argument("input_image_2")
args = parser.parse_args()

fixed_image = itk.imread(args.input_image_1, itk.F)
moving_image = itk.imread(args.input_image_2, itk.F)
```

(continues on next page)

(continued from previous page)

```
metric = itk.MeanSquaresImageToImageMetric[type(fixed_image), type(moving_image)].
↪New()
transform = itk.TranslationTransform[itk.D, fixed_image.GetImageDimension()].New()
interpolator = itk.LinearInterpolateImageFunction[type(fixed_image), itk.D].New()

metric.SetFixedImage(fixed_image)
metric.SetMovingImage(moving_image)
metric.SetFixedImageRegion(fixed_image.GetLargestPossibleRegion())
metric.SetTransform(transform)
metric.SetInterpolator(interpolator)
metric.Initialize()

params = itk.OptimizerParameters[itk.D]()
params.SetSize(2)

for x in range(-10, 15, 5):
    params.SetElement(0, x)
    for y in range(-10, 15, 5):
        params.SetElement(1, y)
        print(f"{list(params)}: {metric.GetValue(params):.1f}")
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**>
**class MeanSquaresImageToImageMetric** : **public** itk::ImageToImageMetric<*TFixedImage*, *TMovingImage*>
TODO.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Compute Mean Squares Metric Between Two Images

See itk::MeanSquaresImageToImageMetric for additional documentation.

### Global Registration of Two Images

### Synopsis

A basic global registration of two images.

### Results

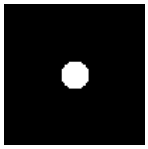


Fig. 352: fixing.png

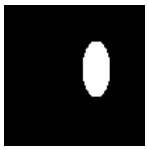


Fig. 353: moving.png

Fig. 354: output.png

Output:

```
Result =
Translation X = 15.0103
Translation Y = -1.12679
Iterations    = 15
Metric value  = 2374.3
```

### Code

### C++

```
#include "itkCastImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkImage.h"
#include "itkImageRegistrationMethod.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkMeanSquaresImageToImageMetric.h"
#include "itkRegularStepGradientDescentOptimizer.h"
#include "itkResampleImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkTranslationTransform.h"

constexpr unsigned int Dimension = 2;
```

(continues on next page)

(continued from previous page)

```
using PixelType = unsigned char;

using ImageType = itk::Image<PixelType, Dimension>;

static void
CreateEllipseImage(ImageType::Pointer image);
static void
CreateSphereImage(ImageType::Pointer image);

int
main(int, char *[])
{
  //  The transform that will map the fixed image into the moving image.
  using TransformType = itk::TranslationTransform<double, Dimension>;

  //  An optimizer is required to explore the parameter space of the transform
  //  in search of optimal values of the metric.
  using OptimizerType = itk::RegularStepGradientDescentOptimizer;

  //  The metric will compare how well the two images match each other. Metric
  //  types are usually parameterized by the image types as it can be seen in
  //  the following type declaration.
  using MetricType = itk::MeanSquaresImageToImageMetric<ImageType, ImageType>;

  //  Finally, the type of the interpolator is declared. The interpolator will
  //  evaluate the intensities of the moving image at non-grid positions.
  using InterpolatorType = itk::LinearInterpolateImageFunction<ImageType, double>;

  //  The registration method type is instantiated using the types of the
  //  fixed and moving images. This class is responsible for interconnecting
  //  all the components that we have described so far.
  using RegistrationType = itk::ImageRegistrationMethod<ImageType, ImageType>;

  // Create components
  MetricType::Pointer       metric = MetricType::New();
  TransformType::Pointer    transform = TransformType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();

  // Each component is now connected to the instance of the registration method.
  registration->SetMetric(metric);
  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);

  // Get the two images
  ImageType::Pointer fixedImage = ImageType::New();
  ImageType::Pointer movingImage = ImageType::New();

  CreateSphereImage(fixedImage);
  CreateEllipseImage(movingImage);

  // Write the two synthetic inputs
  using WriterType = itk::ImageFileWriter<ImageType>;

  WriterType::Pointer fixedWriter = WriterType::New();
```

(continues on next page)

(continued from previous page)

```
fixedWriter->SetFileName("fixed.png");
fixedWriter->SetInput(fixedImage);
fixedWriter->Update();

WriterType::Pointer movingWriter = WriterType::New();
movingWriter->SetFileName("moving.png");
movingWriter->SetInput(movingImage);
movingWriter->Update();

// Set the registration inputs
registration->SetFixedImage(fixedImage);
registration->SetMovingImage(movingImage);

registration->SetFixedImageRegion(fixedImage->GetLargestPossibleRegion());

//  Initialize the transform
using ParametersType = RegistrationType::ParametersType;
ParametersType initialParameters(transform->GetNumberOfParameters());

initialParameters[0] = 0.0; // Initial offset along X
initialParameters[1] = 0.0; // Initial offset along Y

registration->SetInitialTransformParameters(initialParameters);

optimizer->SetMaximumStepLength(4.00);
optimizer->SetMinimumStepLength(0.01);

// Set a stopping criterion
optimizer->SetNumberOfIterations(200);

// Connect an observer
// CommandIterationUpdate::Pointer observer = CommandIterationUpdate::New();
// optimizer->AddObserver( itk::IterationEvent(), observer );

try
{
  registration->Update();
}
catch (itk::ExceptionObject & err)
{
  std::cerr << "ExceptionObject caught !" << std::endl;
  std::cerr << err << std::endl;
  return EXIT_FAILURE;
}

//  The result of the registration process is an array of parameters that
//  defines the spatial transformation in an unique way. This final result is
//  obtained using the \code{GetLastTransformParameters()} method.

ParametersType finalParameters = registration->GetLastTransformParameters();

//  In the case of the \doxygen{TranslationTransform}, there is a
//  straightforward interpretation of the parameters.  Each element of the
//  array corresponds to a translation along one spatial dimension.

const double TranslationAlongX = finalParameters[0];
const double TranslationAlongY = finalParameters[1];
```

(continues on next page)

(continued from previous page)

```
//  The optimizer can be queried for the actual number of iterations
//  performed to reach convergence.  The \code{GetCurrentIteration()}
//  method returns this value. A large number of iterations may be an
//  indication that the maximum step length has been set too small, which
//  is undesirable since it results in long computational times.

const unsigned int numberOfIterations = optimizer->GetCurrentIteration();

//  The value of the image metric corresponding to the last set of parameters
//  can be obtained with the \code{GetValue()} method of the optimizer.

const double bestValue = optimizer->GetValue();

// Print out results
//
std::cout << "Result = " << std::endl;
std::cout << " Translation X = " << TranslationAlongX << std::endl;
std::cout << " Translation Y = " << TranslationAlongY << std::endl;
std::cout << " Iterations    = " << numberOfIterations << std::endl;
std::cout << " Metric value  = " << bestValue << std::endl;

//  It is common, as the last step of a registration task, to use the
//  resulting transform to map the moving image into the fixed image space.
//  This is easily done with the \doxygen{ResampleImageFilter}. Please
//  refer to Section~\ref{sec:ResampleImageFilter} for details on the use
//  of this filter.  First, a ResampleImageFilter type is instantiated
//  using the image types. It is convenient to use the fixed image type as
//  the output type since it is likely that the transformed moving image
//  will be compared with the fixed image.

using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;

//  A resampling filter is created and the moving image is connected as  its input.

ResampleFilterType::Pointer resampler = ResampleFilterType::New();
resampler->SetInput(movingImage);

//  The Transform that is produced as output of the Registration method is
//  also passed as input to the resampling filter. Note the use of the
//  methods \code{GetOutput()} and \code{Get()}. This combination is needed
//  here because the registration method acts as a filter whose output is a
//  transform decorated in the form of a \doxygen{DataObject}. For details in
//  this construction you may want to read the documentation of the
//  \doxygen{DataObjectDecorator}.

resampler->SetTransform(registration->GetOutput()->Get());

//  As described in Section \ref{sec:ResampleImageFilter}, the
//  ResampleImageFilter requires additional parameters to be specified, in
//  particular, the spacing, origin and size of the output image. The default
//  pixel value is also set to a distinct gray level in order to highlight
//  the regions that are mapped outside of the moving image.

resampler->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
resampler->SetOutputOrigin(fixedImage->GetOrigin());
resampler->SetOutputSpacing(fixedImage->GetSpacing());
```

(continues on next page)

(continued from previous page)

```
  resampler->SetOutputDirection(fixedImage->GetDirection());
  resampler->SetDefaultPixelValue(100);

  //  The output of the filter is passed to a writer that will store the
  //  image in a file. An \doxygen{CastImageFilter} is used to convert the
  //  pixel type of the resampled image to the final type used by the
  //  writer. The cast and writer filters are instantiated below.

  using CastFilterType = itk::CastImageFilter<ImageType, ImageType>;

  WriterType::Pointer     writer = WriterType::New();
  CastFilterType::Pointer caster = CastFilterType::New();
  writer->SetFileName("output.png");

  caster->SetInput(resampler->GetOutput());
  writer->SetInput(caster->GetOutput());
  writer->Update();

  /*
  //  The fixed image and the transformed moving image can easily be compared
  //  using the \doxygen{SubtractImageFilter}. This pixel-wise filter computes
  //  the difference between homologous pixels of its two input images.


  using DifferenceFilterType = itk::SubtractImageFilter<
      FixedImageType,
      FixedImageType,
      FixedImageType >;

  DifferenceFilterType::Pointer difference = DifferenceFilterType::New();

  difference->SetInput1( fixedImageReader->GetOutput() );
  difference->SetInput2( resampler->GetOutput() );
  */


  return EXIT_SUCCESS;
}

void
CreateEllipseImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
```

(continues on next page)

(continued from previous page)

```
  spacing.Fill(1);
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 20;
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = 65;
  translation[1] = 45;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}

void
CreateSphereImage(ImageType::Pointer image)
{
  using EllipseType = itk::EllipseSpatialObject<Dimension>;

  using SpatialObjectToImageFilterType = itk::SpatialObjectToImageFilter<EllipseType,␣
↪ImageType>;

  SpatialObjectToImageFilterType::Pointer imageFilter =␣
↪SpatialObjectToImageFilterType::New();

  ImageType::SizeType size;
  size[0] = 100;
  size[1] = 100;

  imageFilter->SetSize(size);

  ImageType::SpacingType spacing;
  spacing.Fill(1);
  imageFilter->SetSpacing(spacing);

  EllipseType::Pointer   ellipse = EllipseType::New();
  EllipseType::ArrayType radiusArray;
  radiusArray[0] = 10;
  radiusArray[1] = 10;
```

(continues on next page)

(continued from previous page)

```
  ellipse->SetRadiusInObjectSpace(radiusArray);

  using TransformType = EllipseType::TransformType;
  TransformType::Pointer transform = TransformType::New();
  transform->SetIdentity();

  TransformType::OutputVectorType translation;
  translation[0] = 50;
  translation[1] = 50;
  transform->Translate(translation, false);

  ellipse->SetObjectToParentTransform(transform);

  imageFilter->SetInput(ellipse);

  ellipse->SetDefaultInsideValue(255);
  ellipse->SetDefaultOutsideValue(0);
  imageFilter->SetUseObjectValue(true);
  imageFilter->SetOutsideValue(0);

  imageFilter->Update();

  image->Graft(imageFilter->GetOutput());
}
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**>
**class ImageRegistrationMethod** : **public** itk::*ProcessObject*

Base class for Image Registration Methods.

This Class define the generic interface for a registration method.

This class is templated over the type of the two image to be registered. A generic Transform is used by this class. That allows to select at run time the particular type of transformation that is to be applied for registering the images.

This method use a generic Metric in order to compare the two images. the final goal of the registration method is to find the set of parameters of the Transformation that optimizes the metric.

The registration method also support a generic optimizer that can be selected at run-time. The only restriction for the optimizer is that it should be able to operate in single-valued cost functions given that the metrics used to compare images provide a single value as output.

The terms : Fixed image and Moving image are used in this class to indicate what image is being mapped by the transform.

This class uses the coordinate system of the Fixed image as a reference and searches for a Transform that will map points from the space of the Fixed image to the space of the Moving image.

For doing so, a Metric will be continuously applied to compare the Fixed image with the Transformed Moving image. This process also requires to interpolate values from the Moving image.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples

- Global Registration Of Two Images
- Global Registration Two Images (Affine)
- Global Registration Of Two Images (BSpline)

See itk::ImageRegistrationMethod for additional documentation.

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class TranslationTransform** : **public** itk::*Transform*<*TParametersValueType*, *NDimensions*, *NDimensions*>
Translation transformation of a vector space (e.g. space coordinates)

The same functionality could be obtained by using the Affine transform, but with a large difference in performance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Vector Image
- Global Registration Of Two Images
- Mutual Information

See itk::TranslationTransform for additional documentation.

### Match Feature Points

**Note:** **Wish List** Still needs additional work to finish proper creation of example.

### Synopsis

Match feature points.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkBlockMatchingImageFilter.h"
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkPoint.h"
#include "itkPointSet.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image, const unsigned int x);

int
main(int /*argc*/, char * /*argv*/[])
{
  // Create input images
  ImageType::Pointer fixedImage = ImageType::New();
  CreateImage(fixedImage, 40);

  ImageType::Pointer movingImage = ImageType::New();
  CreateImage(movingImage, 50);

  //  using WriterType = itk::ImageFileWriter<ImageType>;
  //  WriterType::Pointer writer = WriterType::New();
  //  writer->SetFileName("input.png");
  //  writer->SetInput(input);
  //  writer->Update();

  //  using BlockMatchingImageFilterType = itk::BlockMatchingImageFilter<ImageType,␣
↪ImageType, PointSetType>;
  using BlockMatchingImageFilterType = itk::BlockMatchingImageFilter<ImageType>;
  BlockMatchingImageFilterType::Pointer blockMatchingImageFilter =␣
↪BlockMatchingImageFilterType::New();

  // Generate feature points
  //  using PointSetType = itk::PointSet< float, 2>;
  using PointSetType = BlockMatchingImageFilterType::FeaturePointsType;
  using PointType = PointSetType::PointType;
  using PointsContainerPointer = PointSetType::PointsContainerPointer;

  PointSetType::Pointer  pointSet = PointSetType::New();
  PointsContainerPointer points = pointSet->GetPoints();

  PointType p0, p1, p2, p3;

  p0[0] = 40.0;
  p0[1] = 40.0;
  p1[0] = 40.0;
  p1[1] = 60.0;
  p2[0] = 60.0;
  p2[1] = 40.0;
  p3[0] = 60.0;
  p2[1] = 60.0;
```

(continues on next page)

(continued from previous page)

```
  points->InsertElement(0, p0);
  points->InsertElement(1, p1);
  points->InsertElement(2, p2);
  points->InsertElement(3, p3);

  blockMatchingImageFilter->SetFixedImage(fixedImage);
  blockMatchingImageFilter->SetMovingImage(movingImage);
  blockMatchingImageFilter->SetFeaturePoints(pointSet);
  blockMatchingImageFilter->UpdateLargestPossibleRegion();

  typename BlockMatchingImageFilterType::DisplacementsType * displacements =
    blockMatchingImageFilter->GetDisplacements();

  std::cout << "There are " << displacements->GetNumberOfPoints() << " displacements.
↪" << std::endl;

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image, const unsigned int x)
{
  // Allocate empty image
  itk::Index<2> start;
  start.Fill(0);
  itk::Size<2> size;
  size.Fill(100);
  ImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a white square
  for (unsigned int r = x; r < x + 20; r++)
  {
    for (unsigned int c = 40; c < 60; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;
      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

## Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage** = *TFixedImage*, typename **TFeatures** = *PointSet*<*Matrix*<SpaceP
**class BlockMatchingImageFilter** : **public** itk::MeshToMeshFilter<*TFeatures*, *TDisplacements*>

Computes displacements of given points from a fixed image in a floating image.

BlockMatchingImageFilter takes fixed and moving Images as well as PointSet of feature points as inputs. Physical coordinates of feature points are stored as point coordinates. Points of the input point set must have unique identifiers within range 0..N-1, where N is the number of points. Pixels (pointData) of input point set are not used. Additionally, by default, feature points are expected to lie at least (SearchRadius + BlockRadius) voxels

from a boundary. This is usually achieved by using an appropriate mask during selection of feature points. If you are unsure whether feature points satisfy the above condition set CheckBoundary flag to true which turns on boundary checks. The default output(0) is a PointSet with displacements stored as vectors. Additional output(1) is a PointSet containing similarities. Similarities are needed to compute displacements and are always computed. The number of points in the output PointSet is equal to the number of points in the input PointSet.

The filter is templated over fixed Image, moving Image, input PointSet, output displacements PointSet and output similarities PointSet.

This filter is intended to be used in the process of Physics-Based Non-Rigid Registration. It computes displacement for selected points based on similarity [M. Bierling, Displacement estimation by hierarchical block matching, Proc. SPIE Vis. Comm. and Image Proc., vol. 1001, pp. 942-951, 1988.].

**Author** Andriy Kot, Center for Real-Time Computing, Old Dominion University, Norfolk, VA

**See** MaskFeaturePointSelectionFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Match Feature Points

See itk::BlockMatchingImageFilter for additional documentation.

### Multiresolution Pyramid From Image

### Synopsis

Construct a multiresolution pyramid from an image.

### Results

Output:

```
Writing output_0.png
Writing output_1.png
Writing output_2.png
Writing output_3.png
```

Fig. 355: output_0.png

Fig. 356: output_1.png

Fig. 357: output_2.png

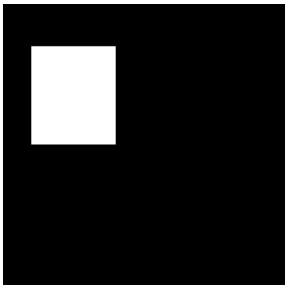


Fig. 358: output_3.png

## Code

### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkRecursiveMultiResolutionPyramidImageFilter.h"

using UnsignedCharImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);

int
main(int, char *[])
{
  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  using FloatImageType = itk::Image<float, 2>;

  unsigned int numberOfLevels = 4;

  using RecursiveMultiResolutionPyramidImageFilterType =
    itk::RecursiveMultiResolutionPyramidImageFilter<UnsignedCharImageType,␣
↪FloatImageType>;
  RecursiveMultiResolutionPyramidImageFilterType::Pointer␣
↪recursiveMultiResolutionPyramidImageFilter =
    RecursiveMultiResolutionPyramidImageFilterType::New();
  recursiveMultiResolutionPyramidImageFilter->SetInput(image);
  recursiveMultiResolutionPyramidImageFilter->SetNumberOfLevels(numberOfLevels);
  recursiveMultiResolutionPyramidImageFilter->Update();

  // This outputs the levels (0 is the lowest resolution)
```

(continues on next page)

(continued from previous page)

```
  for (unsigned int i = 0; i < numberOfLevels; ++i)
  {
    // Scale so we can write to a PNG
    using RescaleFilterType = itk::RescaleIntensityImageFilter<FloatImageType,␣
↪UnsignedCharImageType>;
    RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
    rescaleFilter->SetInput(recursiveMultiResolutionPyramidImageFilter->GetOutput(i));
    rescaleFilter->SetOutputMinimum(0);
    rescaleFilter->SetOutputMaximum(255);
    rescaleFilter->Update();

    using FileWriterType = itk::ImageFileWriter<UnsignedCharImageType>;
    FileWriterType::Pointer writer = FileWriterType::New();
    std::stringstream       ss;
    ss << "output_" << i << ".png";
    std::cout << "Writing " << ss.str() << std::endl;
    writer->SetFileName(ss.str());
    writer->SetInput(rescaleFilter->GetOutput());
    writer->Update();
  }

  return EXIT_SUCCESS;
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create a black image with a white region

  UnsignedCharImageType::IndexType start;
  start.Fill(0);

  UnsignedCharImageType::SizeType size;
  size.Fill(200);

  UnsignedCharImageType::RegionType region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      UnsignedCharImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 255);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RecursiveMultiResolutionPyramidImageFilter** : **public** itk::MultiResolutionPyramidImageFilter<*TInputIma*

Creates a multi-resolution pyramid using a recursive implementation.

RecursiveMultiResolutionPyramidImageFilter creates an image pryamid according to a user defined multi-resolution schedule.

If a schedule is downward divisible, a fast recursive implementation is used to generate the output images. If the schedule is not downward divisible the superclass (MultiResolutionPyramidImageFilter) implementation is used instead. A schedule is downward divisible if at every level, the shrink factors are divisible by the shrink factors at the next level for the same dimension.

See documentation of MultiResolutionPyramidImageFilter for information on how to specify a multi-resolution schedule.

Note that unlike the MultiResolutionPyramidImageFilter, RecursiveMultiResolutionPyramidImageFilter will not smooth the output at the finest level if the shrink factors are all one and the schedule is downward divisible.

This class is templated over the input image type and the output image type.

This filter uses multithreaded filters to perform the smoothing and downsampling.

This filter supports streaming.

**See** MultiResolutionPyramidImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Multiresolution Pyramid From Image

See itk::RecursiveMultiResolutionPyramidImageFilter for additional documentation.

### Mutual Information

### Mutual Information Metric

The `MutualInformationImageToImageMetric` class computes the mutual information between two images, i.e. the degree to which information content in one image is dependent on the other image. This example shows how `MutualInformationImageToImageMetric` can be used to map transformation parameters and register two images using a gradient ascent algorithm.

[24]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from urllib.request import urlretrieve

import itk
from itkwidgets import compare, checkerboard
```

[25]:
```
dim = 2
ImageType = itk.Image[itk.F, dim]
FixedImageType = ImageType
MovingImageType = ImageType
```

### Retrieve fixed and moving images for registration

We aim to register two slice images, one of which has an arbitrary offset.

[26]:
```
fixed_img_path = 'BrainT1SliceBorder20.png'
moving_img_path = 'BrainProtonDensitySliceShifted13x17y.png'
```

[27]:
```
if not os.path.exists(fixed_img_path):
    url = 'https://data.kitware.com/api/v1/file/5cad1ae88d777f072b18183d/download'
    urlretrieve(url, fixed_img_path)
if not os.path.exists(moving_img_path):
    url = 'https://data.kitware.com/api/v1/file/5cad1ae88d777f072b181831/download'
    urlretrieve(url, moving_img_path)
```

[28]:
```
fixed_img = itk.imread('BrainT1SliceBorder20.png', itk.F)
moving_img = itk.imread('BrainProtonDensitySliceShifted13x17y.png', itk.F)
```

[29]:
```
checkerboard(fixed_img, moving_img)
```

```
VBox(children=(Viewer(annotations=False, interpolation=False, rendered_image=<itk.
↪itkImagePython.itkImageF2; p...
```

### Prepare images for registration

[30]:
```
fixed_normalized_image = itk.normalize_image_filter(fixed_img)
fixed_smoothed_image = itk.discrete_gaussian_image_filter(fixed_normalized_image,␣
↪variance=2.0)

moving_normalized_image = itk.normalize_image_filter(moving_img)
moving_smoothed_image = itk.discrete_gaussian_image_filter(moving_normalized_image,␣
↪variance=2.0)
```

[31]:
```
compare(fixed_smoothed_image, moving_smoothed_image)
```

```
AppLayout(children=(HBox(children=(Label(value='Link:'), Checkbox(value=False,␣
↪description='cmap'), Checkbox(v...
```

### Plot the `MutualInformationImageToImageMetric` surface

For this relatively simple example we seek to adjust only the x- and y-offset of the moving image with a `TranslationTransform`. We can acquire `MutualInformationImageToImageMetric` values comparing the two images at many different possible offset pairs with `ExhaustiveOptimizer` and visualize this data set as a surface with `matplotlib`.

[32]:
```
# Move at most 20 pixels away from the initial position
window_size = [20,20]
# Collect 100 steps of data along each axis
n_steps = [100,100]
```

[33]:
```
TransformType = itk.TranslationTransform[itk.D,dim]
OptimizerType = itk.GradientDescentOptimizer;
ExhaustiveOptimizerType = itk.ExhaustiveOptimizer
```

(continues on next page)

(continued from previous page)

```
MetricType = itk.MutualInformationImageToImageMetric[ImageType, ImageType]
RegistrationType = itk.ImageRegistrationMethod[ImageType, ImageType]
InterpolatorType = itk.LinearInterpolateImageFunction[ImageType, itk.D]
```

[34]:
```
transform = TransformType.New()
metric = MetricType.New()
optimizer = ExhaustiveOptimizerType.New()
registrar = RegistrationType.New()
interpolator = InterpolatorType.New()
```

[35]:
```
metric.SetNumberOfSpatialSamples(100)
metric.SetFixedImageStandardDeviation(0.4)
metric.SetMovingImageStandardDeviation(0.4)
```

[36]:
```
optimizer.SetNumberOfSteps(n_steps)

# Initialize scales and set back to optimizer
scales = optimizer.GetScales()
scales.SetSize(2)
scales.SetElement(0,window_size[0] / n_steps[0])
scales.SetElement(1,window_size[1] / n_steps[1])
optimizer.SetScales(scales)
```

[37]:
```
registrar.SetFixedImage(fixed_smoothed_image)
registrar.SetMovingImage(moving_smoothed_image)
registrar.SetOptimizer(optimizer)
registrar.SetTransform(transform)
registrar.SetInterpolator(interpolator)
registrar.SetMetric(metric)

registrar.SetFixedImageRegion(fixed_img.GetBufferedRegion())
registrar.SetInitialTransformParameters(transform.GetParameters())
```

[38]:
```
# Collect data describing the parametric surface with an observer
surface = dict()

def print_iteration():
    surface[tuple(optimizer.GetCurrentPosition())] = optimizer.GetCurrentValue()

optimizer.AddObserver(itk.IterationEvent(), print_iteration)
```

[38]:
```
0
```

[39]:
```
registrar.Update()
```

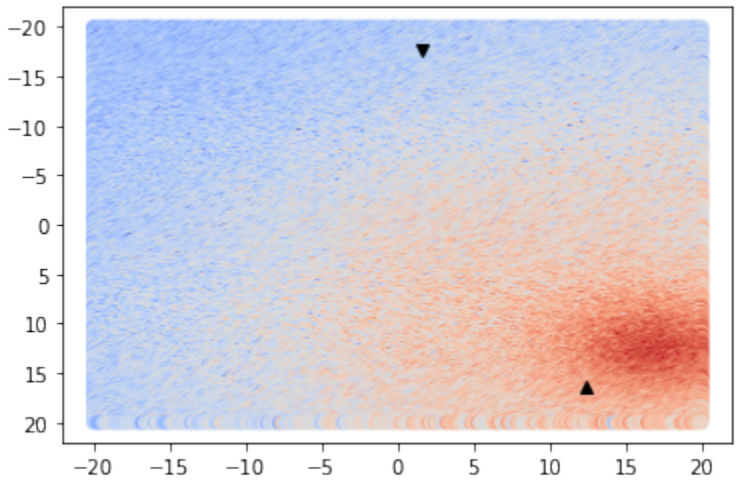
[40]:
```
# Check the extreme positions within the observed window
max_position = list(optimizer.GetMaximumMetricValuePosition())
min_position = list(optimizer.GetMinimumMetricValuePosition())

max_val = optimizer.GetMaximumMetricValue()
min_val = optimizer.GetMinimumMetricValue()

print(max_position)
print(min_position)
```

```
[12.4, 16.400000000000002]
[1.6, -17.6]
```

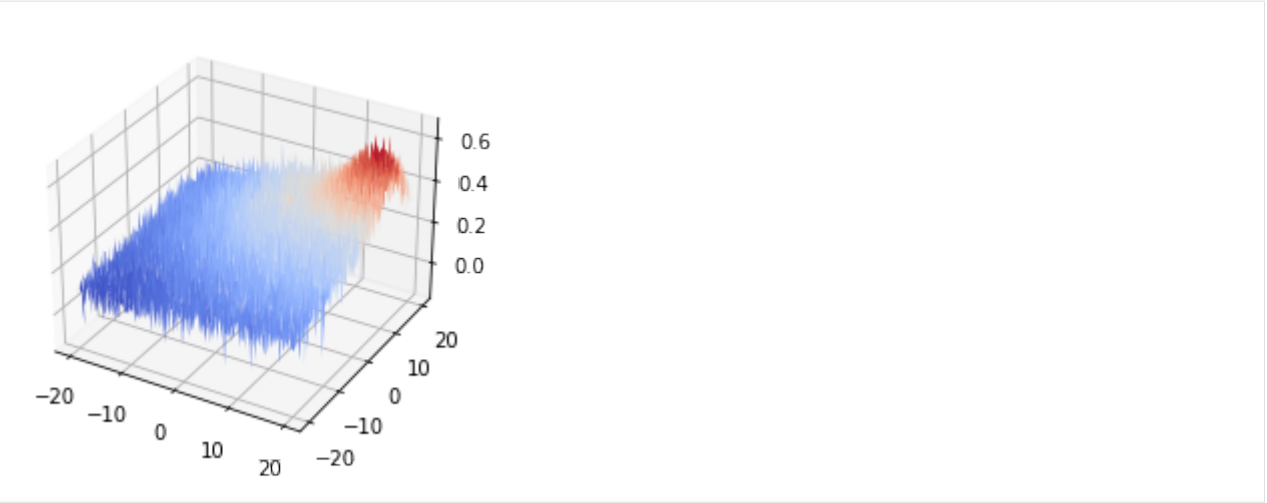
[41]:
```
# Set up values for the plot
x_vals = [list(set([x[i]
                for x in surface.keys()])) for i in range(0,2)]

for i in range(0,2):
    x_vals[i].sort()

X, Y = np.meshgrid(x_vals[0], x_vals[1])
Z = np.array([[surface[(x0,x1)] for x1 in x_vals[0]]for x0 in x_vals[1]])
```

[42]:
```
# Plot the surface as a 2D heat map
fig = plt.figure()
plt.gca().invert_yaxis()
ax = plt.gca()

surf = ax.scatter(X, Y, c=Z, cmap=cm.coolwarm)
ax.plot(max_position[0],max_position[1],'k^')
ax.plot(min_position[0],min_position[1],'kv')
```

[42]:
```
[<matplotlib.lines.Line2D at 0x2c36ecab220>]
```




[43]:
```
# Plot the surface as a 3D scatter plot
fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X,Y,Z,cmap=cm.coolwarm)
```




### Follow gradient ascent

Once we understand the shape of the parametric surface it is easier to visualize the gradient ascent algorithm. We see that there is some roughness to the surface, but it has a clear slope upwards. We want to maximize the mutual information between the two images in order to optimize registration. The results of gradient ascent optimization can be superimposed onto the `matplotlib` plot.

[44]:
```
n_iterations = 200
```

[45]:
```
transform = TransformType.New()
metric = MetricType.New()
optimizer = OptimizerType.New()
registrar = RegistrationType.New()
interpolator = InterpolatorType.New()
```

[46]:
```
registrar.SetFixedImage(fixed_smoothed_image)
registrar.SetMovingImage(moving_smoothed_image)
registrar.SetOptimizer(optimizer)
registrar.SetTransform(transform)
registrar.SetInterpolator(interpolator)
registrar.SetMetric(metric)

registrar.SetFixedImageRegion(fixed_img.GetBufferedRegion())
registrar.SetInitialTransformParameters(transform.GetParameters())
```

[47]:
```
metric.SetNumberOfSpatialSamples(100)
metric.SetFixedImageStandardDeviation(0.4)
metric.SetMovingImageStandardDeviation(0.4)

optimizer.SetLearningRate(15)
optimizer.SetNumberOfIterations(n_iterations)
optimizer.MaximizeOn()
```

[48]:
```
descent_data = dict()
descent_data[0] = (0,0)
```

(continues on next page)

(continued from previous page)

```
def log_iteration():
    descent_data[optimizer.GetCurrentIteration() + 1] = tuple(optimizer.
↪GetCurrentPosition())

optimizer.AddObserver(itk.IterationEvent(), log_iteration)
```

[48]:
```
0
```

[49]:
```
registrar.Update()
```

[50]:
```
print(f'Its: {optimizer.GetCurrentIteration()}')
print(f'Final Value: {optimizer.GetValue()}')
print(f'Final Position: {list(registrar.GetLastTransformParameters())}')
```

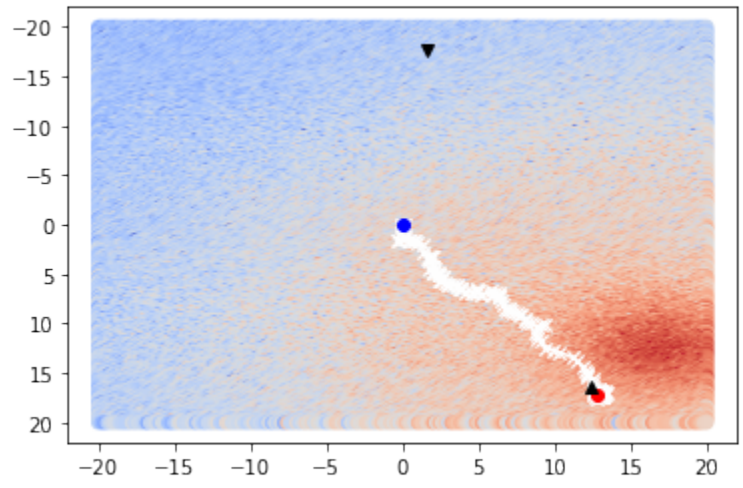
```
Its: 200
Final Value: 0.5633384063089615
Final Position: [12.97335400350759, 17.255969531154182]
```

[51]:
```
x_vals = [descent_data[i][0] for i in range(0,n_iterations)]
y_vals = [descent_data[i][1] for i in range(0,n_iterations)]
```

We see in the plot that the metric improves as transformation parameters are updated with each iteration. The value of the metric at each step generally increases, yielding a final value very close to the optimal position in the parameter space window.

[52]:
```
fig = plt.figure()
# Note: We invert the y-axis to represent the image coordinate system
plt.gca().invert_yaxis()
ax = plt.gca()

surf = ax.scatter(X, Y, c=Z, cmap=cm.coolwarm)

for i in range(0,n_iterations-1):
    plt.plot(x_vals[i:i+2],y_vals[i:i+2],'wx-')
plt.plot(descent_data[0][0], descent_data[0][1],'bo')
plt.plot(descent_data[n_iterations-1][0],descent_data[n_iterations-1][1],'ro')

plt.plot(max_position[0], max_position[1], 'k^')
plt.plot(min_position[0], min_position[1], 'kv')
```

[52]:
```
[<matplotlib.lines.Line2D at 0x2c36f234bb0>]
```




[53]:
```
max_position
```

[53]:
```
[12.4, 16.400000000000002]
```

### Resample the moving image

In order to apply the results of gradient ascent we must resample the moving image into the domain of the fixed image. The `TranslationTransform` whose parameters have been selected through gradient ascent is used to dictate how the moving image is sampled from the fixed image domain. We can compare the two images with `itkwidgets` to verify that registration is successful.

[54]:
```
ResampleFilterType = itk.ResampleImageFilter[MovingImageType,FixedImageType]
resample = ResampleFilterType.New(
    Transform=transform,
    Input=moving_img,
    Size=fixed_img.GetLargestPossibleRegion().GetSize(),
    OutputOrigin=fixed_img.GetOrigin(),
    OutputSpacing=fixed_img.GetSpacing(),
    OutputDirection=fixed_img.GetDirection(),
    DefaultPixelValue=100)
```

[55]:
```
resample.Update()
```

[56]:
```
checkerboard(fixed_img, resample.GetOutput())
```

```
VBox(children=(Viewer(annotations=False, interpolation=False, rendered_image=<itk.
↪itkImagePython.itkImageF2; p...
```

### Clean up

[57]:
```
os.remove(fixed_img_path)
os.remove(moving_img_path)
```

### Synopsis

Global registration by maximizing the mutual information and using a translation only transform.

### Results

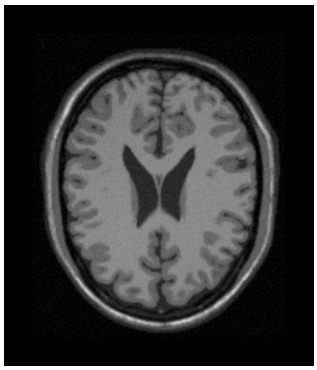


Fig. 359: fixed.png

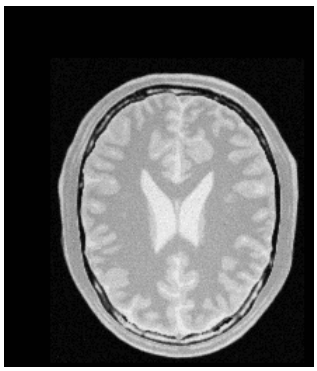


Fig. 360: moving.png

Output:

```
Optimizer stop condition: GradientDescentOptimizer: Maximum number of iterations␣
↪(200) exceeded.

  Result =
   Translation X = 12.9484
   Translation Y = 17.0856
   Iterations    = 200
   Metric value  = 0.594482
   Numb. Samples = 567
```

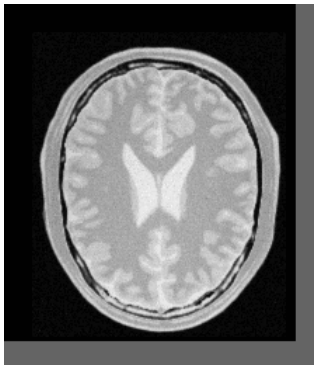


Fig. 361: OutputBaseline.png

### Jupyter Notebook

launch binder

### Code

### C++

```
#include "itkImageRegistrationMethod.h"
#include "itkTranslationTransform.h"
#include "itkMutualInformationImageToImageMetric.h"
#include "itkGradientDescentOptimizer.h"
#include "itkNormalizeImageFilter.h"
#include "itkDiscreteGaussianImageFilter.h"
#include "itkResampleImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkCheckerBoardImageFilter.h"
#include "itkEllipseSpatialObject.h"
#include "itkSpatialObjectToImageFilter.h"
#include "itkImageFileWriter.h"
#include "itkImageFileReader.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned char;

using ImageType = itk::Image<PixelType, Dimension>;

int
main(int argc, char * argv[])
{
  using ReaderType = itk::ImageFileReader<ImageType>;

  if (argc < 4)
  {
    std::cout << "Usage: " << argv[0] << " imageFile1 imageFile2 outputFile" << std::
↪endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  ReaderType::Pointer fixedReader = ReaderType::New();
  fixedReader->SetFileName(argv[1]);
  fixedReader->Update();
  ImageType::Pointer fixedImage = fixedReader->GetOutput();

  ReaderType::Pointer movingReader = ReaderType::New();
  movingReader->SetFileName(argv[2]);
  movingReader->Update();
  ImageType::Pointer movingImage = movingReader->GetOutput();

  // We use floats internally
  using InternalImageType = itk::Image<float, 2>;

  // Normalize the images
  using NormalizeFilterType = itk::NormalizeImageFilter<ImageType, InternalImageType>;

  NormalizeFilterType::Pointer fixedNormalizer = NormalizeFilterType::New();
  NormalizeFilterType::Pointer movingNormalizer = NormalizeFilterType::New();

  fixedNormalizer->SetInput(fixedImage);
  movingNormalizer->SetInput(movingImage);

  // Smooth the normalized images
  using GaussianFilterType = itk::DiscreteGaussianImageFilter<InternalImageType,␣
↪InternalImageType>;

  GaussianFilterType::Pointer fixedSmoother = GaussianFilterType::New();
  GaussianFilterType::Pointer movingSmoother = GaussianFilterType::New();

  fixedSmoother->SetVariance(2.0);
  movingSmoother->SetVariance(2.0);

  fixedSmoother->SetInput(fixedNormalizer->GetOutput());
  movingSmoother->SetInput(movingNormalizer->GetOutput());

  using TransformType = itk::TranslationTransform<double, Dimension>;
  using OptimizerType = itk::GradientDescentOptimizer;
  using InterpolatorType = itk::LinearInterpolateImageFunction<InternalImageType,␣
↪double>;
  using RegistrationType = itk::ImageRegistrationMethod<InternalImageType,␣
↪InternalImageType>;
  using MetricType = itk::MutualInformationImageToImageMetric<InternalImageType,␣
↪InternalImageType>;

  TransformType::Pointer    transform = TransformType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();

  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);

  MetricType::Pointer metric = MetricType::New();
  registration->SetMetric(metric);

  //  The metric requires a number of parameters to be selected, including
```

(continues on next page)

(continued from previous page)

```
  //  the standard deviation of the Gaussian kernel for the fixed image
  //  density estimate, the standard deviation of the kernel for the moving
  //  image density and the number of samples use to compute the densities
  //  and entropy values. Details on the concepts behind the computation of
  //  the metric can be found in Section
  //  \ref{sec:MutualInformationMetric}.  Experience has
  //  shown that a kernel standard deviation of $0.4$ works well for images
  //  which have been normalized to a mean of zero and unit variance.  We
  //  will follow this empirical rule in this example.

  metric->SetFixedImageStandardDeviation(0.4);
  metric->SetMovingImageStandardDeviation(0.4);

  registration->SetFixedImage(fixedSmoother->GetOutput());
  registration->SetMovingImage(movingSmoother->GetOutput());

  fixedNormalizer->Update();
  ImageType::RegionType fixedImageRegion = fixedNormalizer->GetOutput()->
↪GetBufferedRegion();
  registration->SetFixedImageRegion(fixedImageRegion);

  using ParametersType = RegistrationType::ParametersType;
  ParametersType initialParameters(transform->GetNumberOfParameters());

  initialParameters[0] = 0.0; // Initial offset along X
  initialParameters[1] = 0.0; // Initial offset along Y

  registration->SetInitialTransformParameters(initialParameters);

  //  Software Guide : BeginLatex
  //
  //  We should now define the number of spatial samples to be considered in
  //  the metric computation. Note that we were forced to postpone this setting
  //  until we had done the preprocessing of the images because the number of
  //  samples is usually defined as a fraction of the total number of pixels in
  //  the fixed image.
  //
  //  The number of spatial samples can usually be as low as $1\%$ of the total
  //  number of pixels in the fixed image. Increasing the number of samples
  //  improves the smoothness of the metric from one iteration to another and
  //  therefore helps when this metric is used in conjunction with optimizers
  //  that rely of the continuity of the metric values. The trade-off, of
  //  course, is that a larger number of samples result in longer computation
  //  times per every evaluation of the metric.
  //
  //  It has been demonstrated empirically that the number of samples is not a
  //  critical parameter for the registration process. When you start fine
  //  tuning your own registration process, you should start using high values
  //  of number of samples, for example in the range of $20\%$ to $50\%$ of the
  //  number of pixels in the fixed image. Once you have succeeded to register
  //  your images you can then reduce the number of samples progressively until
  //  you find a good compromise on the time it takes to compute one evaluation
  //  of the Metric. Note that it is not useful to have very fast evaluations
  //  of the Metric if the noise in their values results in more iterations
  //  being required by the optimizer to converge. You must then study the
  //  behavior of the metric values as the iterations progress.
```

(continues on next page)

(continued from previous page)

```
  const unsigned int numberOfPixels = fixedImageRegion.GetNumberOfPixels();

  const auto numberOfSamples = static_cast<unsigned int>(numberOfPixels * 0.01);

  metric->SetNumberOfSpatialSamples(numberOfSamples);

  // For consistent results when regression testing.
  metric->ReinitializeSeed(121212);

  //  Since larger values of mutual information indicate better matches than
  //  smaller values, we need to maximize the cost function in this example.
  //  By default the GradientDescentOptimizer class is set to minimize the
  //  value of the cost-function. It is therefore necessary to modify its
  //  default behavior by invoking the \code{MaximizeOn()} method.
  //  Additionally, we need to define the optimizer's step size using the
  //  \code{SetLearningRate()} method.

  optimizer->SetLearningRate(15.0);
  optimizer->SetNumberOfIterations(200);
  optimizer->MaximizeOn(); // We want to maximize mutual information (the default of␣
↪the optimizer is to minimize)

  // Note that large values of the learning rate will make the optimizer
  // unstable. Small values, on the other hand, may result in the optimizer
  // needing too many iterations in order to walk to the extrema of the cost
  // function. The easy way of fine tuning this parameter is to start with
  // small values, probably in the range of $\{5.0,10.0\}$. Once the other
  // registration parameters have been tuned for producing convergence, you
  // may want to revisit the learning rate and start increasing its value until
  // you observe that the optimization becomes unstable.  The ideal value for
  // this parameter is the one that results in a minimum number of iterations
  // while still keeping a stable path on the parametric space of the
  // optimization. Keep in mind that this parameter is a multiplicative factor
  // applied on the gradient of the Metric. Therefore, its effect on the
  // optimizer step length is proportional to the Metric values themselves.
  // Metrics with large values will require you to use smaller values for the
  // learning rate in order to maintain a similar optimizer behavior.

  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition: " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cout << "ExceptionObject caught !" << std::endl;
    std::cout << err << std::endl;
    return EXIT_FAILURE;
  }

  ParametersType finalParameters = registration->GetLastTransformParameters();

  double TranslationAlongX = finalParameters[0];
  double TranslationAlongY = finalParameters[1];
```

(continues on next page)

(continued from previous page)

```
  unsigned int numberOfIterations = optimizer->GetCurrentIteration();

  double bestValue = optimizer->GetValue();


  // Print out results
  std::cout << std::endl;
  std::cout << "Result = " << std::endl;
  std::cout << " Translation X = " << TranslationAlongX << std::endl;
  std::cout << " Translation Y = " << TranslationAlongY << std::endl;
  std::cout << " Iterations    = " << numberOfIterations << std::endl;
  std::cout << " Metric value  = " << bestValue << std::endl;
  std::cout << " Numb. Samples = " << numberOfSamples << std::endl;

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType>;

  TransformType::Pointer finalTransform = TransformType::New();

  finalTransform->SetParameters(finalParameters);
  finalTransform->SetFixedParameters(transform->GetFixedParameters());

  ResampleFilterType::Pointer resample = ResampleFilterType::New();

  resample->SetTransform(finalTransform);
  resample->SetInput(movingImage);

  resample->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resample->SetOutputOrigin(fixedImage->GetOrigin());
  resample->SetOutputSpacing(fixedImage->GetSpacing());
  resample->SetOutputDirection(fixedImage->GetDirection());
  resample->SetDefaultPixelValue(100);

  // Write transformed moving image
  using WriterType = itk::ImageFileWriter<ImageType>;

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(argv[3]);
  writer->SetInput(resample->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**>
**class MutualInformationImageToImageMetric** : **public** itk::ImageToImageMetric<*TFixedImage*, *TMovingImage*>
Computes the mutual information between two images to be registered.

MutualInformationImageToImageMetric computes the mutual information between a fixed and moving image to be registered.

This class is templated over the FixedImage type and the MovingImage type.

The fixed and moving images are set via methods SetFixedImage() and SetMovingImage(). This metric makes use of user specified Transform and Interpolator. The Transform is used to map points from the fixed image to the

moving image domain. The Interpolator is used to evaluate the image intensity at user specified geometric points in the moving image. The Transform and Interpolator are set via methods SetTransform() and SetInterpolator().

The method GetValue() computes of the mutual information while method GetValueAndDerivative() computes both the mutual information and its derivatives with respect to the transform parameters.

**Warning** This metric assumes that the moving image has already been connected to the interpolator outside of this class.

The calculations are based on the method of Viola and Wells where the probability density distributions are estimated using Parzen windows.

By default a Gaussian kernel is used in the density estimation. Other option include Cauchy and spline-based. A user can specify the kernel passing in a pointer a KernelFunctionBase using the SetKernelFunction() method.

Mutual information is estimated using two sample sets: one to calculate the singular and joint pdf's and one to calculate the entropy integral. By default 50 samples points are used in each set. Other values can be set via the SetNumberOfSpatialSamples() method.

Quality of the density estimate depends on the choice of the kernel's standard deviation. Optimal choice will depend on the images. It is can be shown that around the optimal variance, the mutual information estimate is relatively insensitive to small changes of the standard deviation. In our experiments, we have found that a standard deviation of 0.4 works well for images normalized to have a mean of zero and standard deviation of 1.0. The variance can be set via methods SetFixedImageStandardDeviation() and SetMovingImageStandardDeviation().

Implementaton of this class is based on: Viola, P. and Wells III, W. (1997). "Alignment by Maximization of Mutual Information" International Journal of Computer Vision, 24(2):137-154

**See** KernelFunctionBase

**See** GaussianKernelFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mutual Information
- Mutual Information Affine

See itk::MutualInformationImageToImageMetric for additional documentation.

template<typename **TParametersValueType** = double, unsigned int **NDimensions** = 3>
**class TranslationTransform** : **public** itk::*Transform*<*TParametersValueType*, *NDimensions*, *NDimensions*>
Translation transformation of a vector space (e.g. space coordinates)

The same functionality could be obtained by using the Affine transform, but with a large difference in performance.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Translate Vector Image
- Global Registration Of Two Images
- Mutual Information

See itk::TranslationTransform for additional documentation.

### Perform 2D Translation Registration With Mean Squares

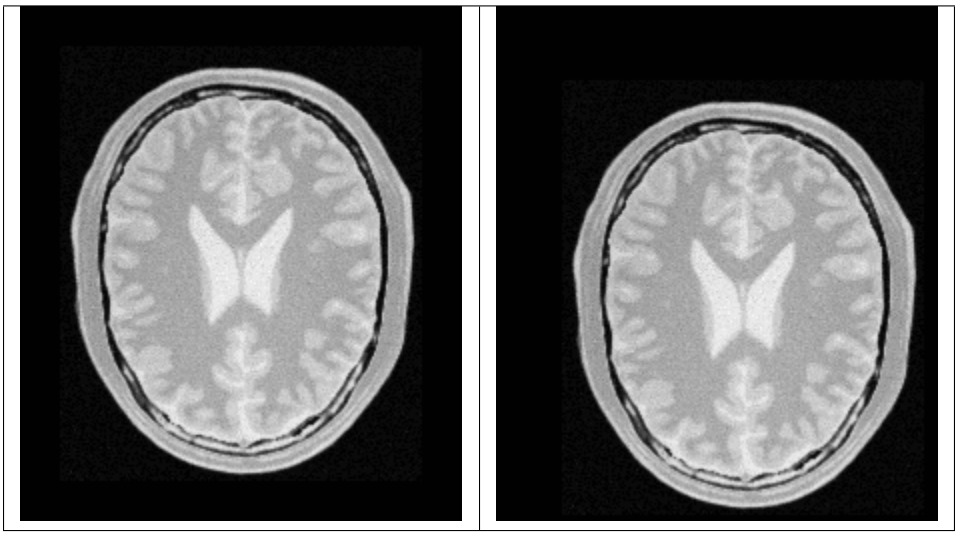
#### Synopsis

This example illustrates the use of the image registration framework in Insight. It should be read as a `Hello World` for ITK registration. Instead of means to an end, this example should be read as a basic introduction to the elements typically involved when solving a problem of image registration.

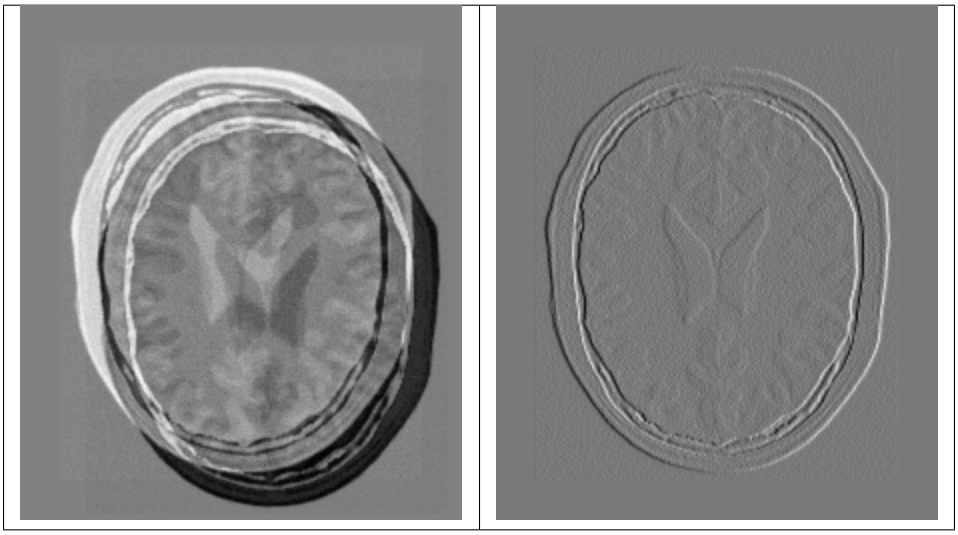
A registration method requires the following set of components: two input images, a transform, a metric and an optimizer. Some of these components are parameterized by the image type for which the registration is intended. The following header files provide declarations of common types used for these components.

This example corresponds to the ImageRegistration1.cxx example from the ITK software guide.

#### Results



Input images (fixed image left/ moving image right).



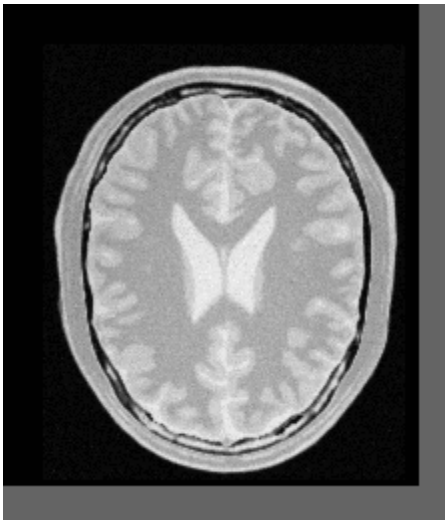


Fig. 362: Output registered image.

Image difference between moving and fixed image (left before registration, right after registration).

### Code

### Python

```
#!/usr/bin/env python

import sys
import itk
import argparse

from distutils.version import StrictVersion as VS

if VS(itk.Version.GetITKVersion()) < VS("4.9.0"):
    print("ITK 4.9.0 is required.")
    sys.exit(1)

parser = argparse.ArgumentParser(
    description="Perform 2D Translation Registration With Mean Squares."
)
parser.add_argument("fixed_input_image")
parser.add_argument("moving_input_image")
parser.add_argument("output_image")
parser.add_argument("difference_image_after")
parser.add_argument("difference_image_before")
args = parser.parse_args()

PixelType = itk.ctype("float")

fixedImage = itk.imread(args.fixed_input_image, PixelType)
movingImage = itk.imread(args.moving_input_image, PixelType)

Dimension = fixedImage.GetImageDimension()
```

(continues on next page)

(continued from previous page)

```
FixedImageType = itk.Image[PixelType, Dimension]
MovingImageType = itk.Image[PixelType, Dimension]

TransformType = itk.TranslationTransform[itk.D, Dimension]
initialTransform = TransformType.New()

optimizer = itk.RegularStepGradientDescentOptimizerv4.New(
    LearningRate=4,
    MinimumStepLength=0.001,
    RelaxationFactor=0.5,
    NumberOfIterations=200,
)

metric = itk.MeanSquaresImageToImageMetricv4[FixedImageType, MovingImageType].New()

registration = itk.ImageRegistrationMethodv4.New(
    FixedImage=fixedImage,
    MovingImage=movingImage,
    Metric=metric,
    Optimizer=optimizer,
    InitialTransform=initialTransform,
)

movingInitialTransform = TransformType.New()
initialParameters = movingInitialTransform.GetParameters()
initialParameters[0] = 0
initialParameters[1] = 0
movingInitialTransform.SetParameters(initialParameters)
registration.SetMovingInitialTransform(movingInitialTransform)

identityTransform = TransformType.New()
identityTransform.SetIdentity()
registration.SetFixedInitialTransform(identityTransform)

registration.SetNumberOfLevels(1)
registration.SetSmoothingSigmasPerLevel([0])
registration.SetShrinkFactorsPerLevel([1])

registration.Update()

transform = registration.GetTransform()
finalParameters = transform.GetParameters()
translationAlongX = finalParameters.GetElement(0)
translationAlongY = finalParameters.GetElement(1)

numberOfIterations = optimizer.GetCurrentIteration()

bestValue = optimizer.GetValue()

print("Result = ")
print(" Translation X = " + str(translationAlongX))
print(" Translation Y = " + str(translationAlongY))
print(" Iterations    = " + str(numberOfIterations))
print(" Metric value  = " + str(bestValue))

CompositeTransformType = itk.CompositeTransform[itk.D, Dimension]
outputCompositeTransform = CompositeTransformType.New()
```

(continues on next page)

(continued from previous page)

```
outputCompositeTransform.AddTransform(movingInitialTransform)
outputCompositeTransform.AddTransform(registration.GetModifiableTransform())

resampler = itk.ResampleImageFilter.New(
    Input=movingImage,
    Transform=outputCompositeTransform,
    UseReferenceImage=True,
    ReferenceImage=fixedImage,
)
resampler.SetDefaultPixelValue(100)

OutputPixelType = itk.ctype("unsigned char")
OutputImageType = itk.Image[OutputPixelType, Dimension]

caster = itk.CastImageFilter[FixedImageType, OutputImageType].New(Input=resampler)

writer = itk.ImageFileWriter.New(Input=caster, FileName=args.output_image)
writer.Update()

difference = itk.SubtractImageFilter.New(Input1=fixedImage, Input2=resampler)

intensityRescaler = itk.RescaleIntensityImageFilter[
    FixedImageType, OutputImageType
].New(
    Input=difference,
    OutputMinimum=itk.NumericTraits[OutputPixelType].min(),
    OutputMaximum=itk.NumericTraits[OutputPixelType].max(),
)

resampler.SetDefaultPixelValue(1)
writer.SetInput(intensityRescaler.GetOutput())
writer.SetFileName(args.difference_image_after)
writer.Update()

resampler.SetTransform(identityTransform)
writer.SetFileName(args.difference_image_before)
writer.Update()
```

### C++

```
#include "itkImageRegistrationMethodv4.h"
#include "itkTranslationTransform.h"
#include "itkMeanSquaresImageToImageMetricv4.h"
#include "itkRegularStepGradientDescentOptimizerv4.h"

#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkResampleImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkSubtractImageFilter.h"
```

(continues on next page)

(continued from previous page)

```
int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " fixedImageFile  movingImageFile ";
    std::cerr << "outputImagefile [differenceImageAfter]";
    std::cerr << "[differenceImageBefore]" << std::endl;
    return EXIT_FAILURE;
  }

  const char * fixedImageFile = argv[1];
  const char * movingImageFile = argv[2];
  const char * outputImageFile = argv[3];
  const char * differenceImageAfterFile = argv[4];
  const char * differenceImageBeforeFile = argv[5];

  constexpr unsigned int Dimension = 2;
  using PixelType = float;

  using FixedImageType = itk::Image<PixelType, Dimension>;
  using MovingImageType = itk::Image<PixelType, Dimension>;


  using FixedImageReaderType = itk::ImageFileReader<FixedImageType>;
  auto fixedImageReader = FixedImageReaderType::New();
  fixedImageReader->SetFileName(fixedImageFile);

  using MovingImageReaderType = itk::ImageFileReader<MovingImageType>;
  auto movingImageReader = MovingImageReaderType::New();
  movingImageReader->SetFileName(movingImageFile);


  using TransformType = itk::TranslationTransform<double, Dimension>;
  auto initialTransform = TransformType::New();

  using OptimizerType = itk::RegularStepGradientDescentOptimizerv4<double>;
  auto optimizer = OptimizerType::New();
  optimizer->SetLearningRate(4);
  optimizer->SetMinimumStepLength(0.001);
  optimizer->SetRelaxationFactor(0.5);
  optimizer->SetNumberOfIterations(200);

  using MetricType = itk::MeanSquaresImageToImageMetricv4<FixedImageType,␣
↪MovingImageType>;
  auto metric = MetricType::New();

  using RegistrationType = itk::ImageRegistrationMethodv4<FixedImageType,␣
↪MovingImageType>;
  auto registration = RegistrationType::New();
  registration->SetInitialTransform(initialTransform);
  registration->SetMetric(metric);
  registration->SetOptimizer(optimizer);
  registration->SetFixedImage(fixedImageReader->GetOutput());
  registration->SetMovingImage(movingImageReader->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  auto                          movingInitialTransform = TransformType::New();
  TransformType::ParametersType initialParameters(movingInitialTransform->
↪GetNumberOfParameters());
  initialParameters[0] = 0.0;
  initialParameters[1] = 0.0;
  movingInitialTransform->SetParameters(initialParameters);
  registration->SetMovingInitialTransform(movingInitialTransform);

  auto identityTransform = TransformType::New();
  identityTransform->SetIdentity();
  registration->SetFixedInitialTransform(identityTransform);

  constexpr unsigned int numberOfLevels = 1;
  registration->SetNumberOfLevels(numberOfLevels);

  RegistrationType::ShrinkFactorsArrayType shrinkFactorsPerLevel;
  shrinkFactorsPerLevel.SetSize(1);
  shrinkFactorsPerLevel[0] = 1;
  registration->SetShrinkFactorsPerLevel(shrinkFactorsPerLevel);

  RegistrationType::SmoothingSigmasArrayType smoothingSigmasPerLevel;
  smoothingSigmasPerLevel.SetSize(1);
  smoothingSigmasPerLevel[0] = 0;
  registration->SetSmoothingSigmasPerLevel(smoothingSigmasPerLevel);

  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition: " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cerr << "ExceptionObject caught !" << std::endl;
    std::cerr << err << std::endl;
    return EXIT_FAILURE;
  }

  auto transform = registration->GetTransform();
  auto finalParameters = transform->GetParameters();
  auto translationAlongX = finalParameters[0];
  auto translationAlongY = finalParameters[1];

  auto numberOfIterations = optimizer->GetCurrentIteration();

  auto bestValue = optimizer->GetValue();

  std::cout << "Result = " << std::endl;
  std::cout << " Translation X = " << translationAlongX << std::endl;
  std::cout << " Translation Y = " << translationAlongY << std::endl;
  std::cout << " Iterations    = " << numberOfIterations << std::endl;
  std::cout << " Metric value  = " << bestValue << std::endl;

  using CompositeTransformType = itk::CompositeTransform<double, Dimension>;
  auto outputCompositeTransform = CompositeTransformType::New();
```

(continues on next page)

(continued from previous page)

```
  outputCompositeTransform->AddTransform(movingInitialTransform);
  outputCompositeTransform->AddTransform(registration->GetModifiableTransform());

  using ResampleFilterType = itk::ResampleImageFilter<MovingImageType, FixedImageType>
↪;
  auto resampler = ResampleFilterType::New();
  resampler->SetInput(movingImageReader->GetOutput());
  resampler->SetTransform(outputCompositeTransform);
  auto fixedImage = fixedImageReader->GetOutput();
  resampler->SetUseReferenceImage(true);
  resampler->SetReferenceImage(fixedImage);
  resampler->SetDefaultPixelValue(100);

  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using CastFilterType = itk::CastImageFilter<FixedImageType, OutputImageType>;
  auto caster = CastFilterType::New();
  caster->SetInput(resampler->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  auto writer = WriterType::New();
  writer->SetFileName(outputImageFile);
  writer->SetInput(caster->GetOutput());
  writer->Update();

  using DifferenceFilterType = itk::SubtractImageFilter<FixedImageType,␣
↪FixedImageType, FixedImageType>;
  auto difference = DifferenceFilterType::New();
  difference->SetInput1(fixedImageReader->GetOutput());
  difference->SetInput2(resampler->GetOutput());

  using RescalerType = itk::RescaleIntensityImageFilter<FixedImageType,␣
↪OutputImageType>;
  auto intensityRescaler = RescalerType::New();
  intensityRescaler->SetInput(difference->GetOutput());
  intensityRescaler->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  intensityRescaler->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  resampler->SetDefaultPixelValue(1);

  writer->SetInput(intensityRescaler->GetOutput());
  writer->SetFileName(differenceImageAfterFile);
  writer->Update();

  resampler->SetTransform(identityTransform);
  writer->SetFileName(differenceImageBeforeFile);
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**, typename **TOutputTransform** = *Transform*<double, *TFixedIma*
**class ImageRegistrationMethodv4** : **public** itk::*ProcessObject*

Interface method for the current registration framework.

This interface method class encapsulates typical registration usage by incorporating all the necessary elements for performing a simple image registration between two images. This method also allows for multistage registration whereby each stage is characterize by possibly different transforms of and different image metrics. For example, many users will want to perform a linear registration followed by deformable registration where both stages are performed in multiple levels. Each level can be characterized by:

- the resolution of the virtual domain image (see below)
- smoothing of the fixed and moving images
- the coarseness of the current transform via transform adaptors (see below)

Multiple stages are handled by linking multiple instantiations of this class where the output transform is added to the optional composite transform input.

Transform adaptors: To accommodate new changes to the current ITK registration framework, we introduced the concept of transform adaptors. Whereas each stage is associated with a moving and, possibly, fixed transform, each level of each stage is defined by a transform adaptor which describes how to adapt the transform to the current level. For example, if one were to use the B-spline transform during a deformable registration stage, common practice is to increase the resolution of the B-spline mesh (or, analogously, the control point grid size) at each level. At each level, one would define the parameters of the B-spline transform adaptor at that level which increases the resolution from the previous level. For many transforms, such as affine, this concept of an adaptor may be nonsensical. For this reason, the base transform adaptor class does not do anything to the transform but merely passes it through. Each level of each stage must define a transform adaptor but, by default, the base adaptor class is assigned which, again, does not do anything to the transform. A special mention should be made of the transform adaptor at level 0 of any stage. Most likely, the user will not want to do anything to the transform as it enters into the given stage so typical use will be to assign the base adaptor class to level 0 of all stages but we leave that open to the user.

Output: The output is the updated transform.

**Author** Nick Tustison

**Author** Brian Avants

Subclassed by itk::SyNImageRegistrationMethod< TFixedImage, TMovingImage, TOutputTransform, TVirtualImage, TPointSet >, itk::TimeVaryingBSplineVelocityFieldImageRegistrationMethod< TFixedImage, TMovingImage, TOutputTransform, TVirtualImage, TPointSet >, itk::TimeVaryingVelocityFieldImageRegistrationMethodv4< TFixedImage, TMovingImage, TOutputTransform, TVirtualImage, TPointSet >

See itk::ImageRegistrationMethodv4 for additional documentation.

### Perform Multi Modality Registration With Viola Wells Mutual Information

### Synopsis

Rigid registration between two modalities with a Viola-Wells like mutual information metric.

### Results

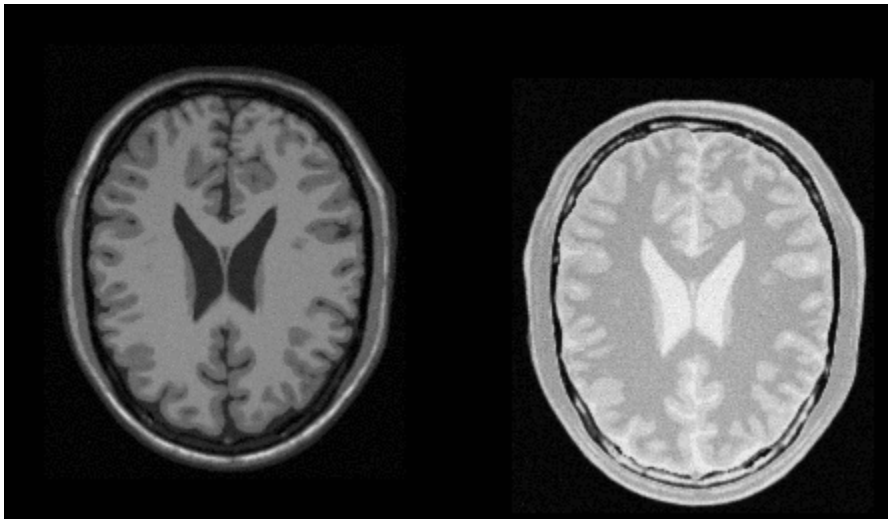


Fig. 363: Input fixed image (left) and moving image (right);

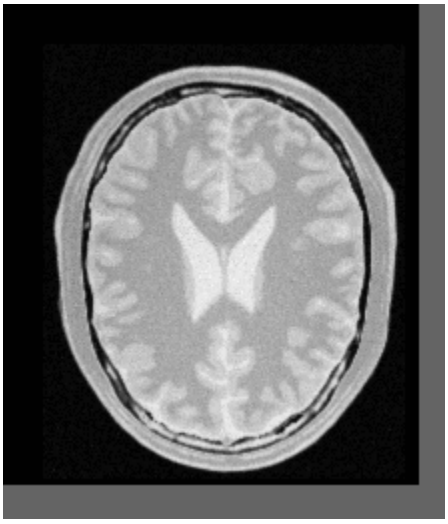


Fig. 364: Output registered image.

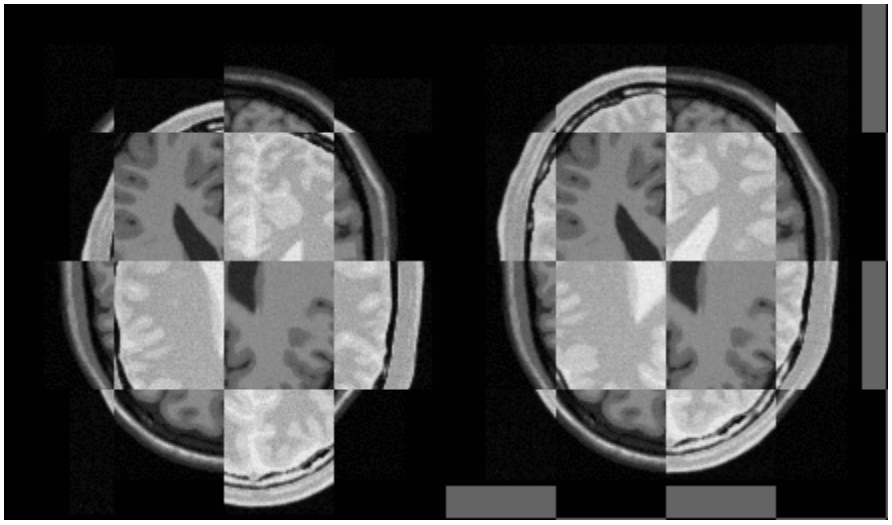


Fig. 365: Fixed/moving image checkerboard before (left) and after (right).

### Code

### C++

```
// The following simple example illustrates how multiple imaging modalities can
// be registered using the ITK registration framework. The first difference
// between this and previous examples is the use of the
// MutualInformationImageToImageMetric as the cost-function to be
// optimized. The second difference is the use of the
// GradientDescentOptimizer. Due to the stochastic nature of the
// metric computation, the values are too noisy to work successfully with the
// RegularStepGradientDescentOptimizer.  Therefore, we will use the
// simpler GradientDescentOptimizer with a user defined learning rate.  The
// following headers declare the basic components of this registration method.
#include "itkImageRegistrationMethod.h"
#include "itkTranslationTransform.h"
#include "itkMutualInformationImageToImageMetric.h"
#include "itkGradientDescentOptimizer.h"


//  One way to simplify the computation of the mutual information is
//  to normalize the statistical distribution of the two input images. The
//  NormalizeImageFilter is the perfect tool for this task.
//  It rescales the intensities of the input images in order to produce an
//  output image with zero mean and unit variance.
#include "itkNormalizeImageFilter.h"

//  Additionally, low-pass filtering of the images to be registered will also
//  increase robustness against noise. In this example, we will use the
//  DiscreteGaussianImageFilter for that purpose.
#include "itkDiscreteGaussianImageFilter.h"


#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
```

(continues on next page)

(continued from previous page)

```
#include "itkResampleImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkCheckerBoardImageFilter.h"


//  The following section of code implements a Command observer
//  that will monitor the evolution of the registration process.
//
#include "itkCommand.h"
class CommandIterationUpdate : public itk::Command
{
public:
  using Self = CommandIterationUpdate;
  using Superclass = itk::Command;
  using Pointer = itk::SmartPointer<Self>;
  itkNewMacro(Self);

protected:
  CommandIterationUpdate() = default;

public:
  using OptimizerType = itk::GradientDescentOptimizer;
  using OptimizerPointer = const OptimizerType *;

  void
  Execute(itk::Object * caller, const itk::EventObject & event) override
  {
    Execute((const itk::Object *)caller, event);
  }

  void
  Execute(const itk::Object * object, const itk::EventObject & event) override
  {
    auto optimizer = dynamic_cast<OptimizerPointer>(object);
    if (!itk::IterationEvent().CheckEvent(&event))
    {
      return;
    }
    std::cout << optimizer->GetCurrentIteration() << "   ";
    std::cout << optimizer->GetValue() << "   ";
    std::cout << optimizer->GetCurrentPosition() << std::endl;
  }
};


int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " fixedImageFile";
    std::cerr << " movingImageFile";
    std::cerr << " outputImageFile ";
    std::cerr << " [checkerBoardBefore]";
```

(continues on next page)

(continued from previous page)

```
    std::cerr << " [checkerBoardAfter]" << std::endl;
    return EXIT_FAILURE;
  }
  const char * fixedImageFile = argv[1];
  const char * movingImageFile = argv[2];
  const char * outputImageFile = argv[3];
  const char * checkerBoardBefore = argv[4];
  const char * checkerBoardAfter = argv[5];

  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned short;

  using FixedImageType = itk::Image<PixelType, Dimension>;
  using MovingImageType = itk::Image<PixelType, Dimension>;

  //  It is convenient to work with an internal image type because mutual
  //  information will perform better on images with a normalized statistical
  //  distribution. The fixed and moving images will be normalized and
  //  converted to this internal type.
  using InternalPixelType = float;
  using InternalImageType = itk::Image<InternalPixelType, Dimension>;

  using TransformType = itk::TranslationTransform<double, Dimension>;
  using OptimizerType = itk::GradientDescentOptimizer;
  using InterpolatorType = itk::LinearInterpolateImageFunction<InternalImageType,␣
↪double>;
  using RegistrationType = itk::ImageRegistrationMethod<InternalImageType,␣
↪InternalImageType>;

  using MetricType = itk::MutualInformationImageToImageMetric<InternalImageType,␣
↪InternalImageType>;


  TransformType::Pointer    transform = TransformType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();
  MetricType::Pointer       metric = MetricType::New();

  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);
  registration->SetMetric(metric);

  //  The metric requires a number of parameters to be selected, including
  //  the standard deviation of the Gaussian kernel for the fixed image
  //  density estimate, the standard deviation of the kernel for the moving
  //  image density and the number of samples use to compute the densities
  //  and entropy values. Experience has
  //  shown that a kernel standard deviation of 0.4 works well for images
  //  which have been normalized to a mean of zero and unit variance.  We
  //  will follow this empirical rule in this example.
  metric->SetFixedImageStandardDeviation(0.4);
  metric->SetMovingImageStandardDeviation(0.4);

  using FixedImageReaderType = itk::ImageFileReader<FixedImageType>;
  using MovingImageReaderType = itk::ImageFileReader<MovingImageType>;
```

(continues on next page)

(continued from previous page)

```
  FixedImageReaderType::Pointer  fixedImageReader = FixedImageReaderType::New();
  MovingImageReaderType::Pointer movingImageReader = MovingImageReaderType::New();

  fixedImageReader->SetFileName(fixedImageFile);
  movingImageReader->SetFileName(movingImageFile);

  using FixedNormalizeFilterType = itk::NormalizeImageFilter<FixedImageType,␣
↪InternalImageType>;

  using MovingNormalizeFilterType = itk::NormalizeImageFilter<MovingImageType,␣
↪InternalImageType>;

  FixedNormalizeFilterType::Pointer fixedNormalizer = FixedNormalizeFilterType::New();

  MovingNormalizeFilterType::Pointer movingNormalizer = MovingNormalizeFilterType::
↪New();

  using GaussianFilterType = itk::DiscreteGaussianImageFilter<InternalImageType,␣
↪InternalImageType>;

  GaussianFilterType::Pointer fixedSmoother = GaussianFilterType::New();
  GaussianFilterType::Pointer movingSmoother = GaussianFilterType::New();

  fixedSmoother->SetVariance(2.0);
  movingSmoother->SetVariance(2.0);

  fixedNormalizer->SetInput(fixedImageReader->GetOutput());
  movingNormalizer->SetInput(movingImageReader->GetOutput());

  fixedSmoother->SetInput(fixedNormalizer->GetOutput());
  movingSmoother->SetInput(movingNormalizer->GetOutput());

  registration->SetFixedImage(fixedSmoother->GetOutput());
  registration->SetMovingImage(movingSmoother->GetOutput());


  fixedNormalizer->Update();
  FixedImageType::RegionType fixedImageRegion = fixedNormalizer->GetOutput()->
↪GetBufferedRegion();
  registration->SetFixedImageRegion(fixedImageRegion);

  using ParametersType = RegistrationType::ParametersType;
  ParametersType initialParameters(transform->GetNumberOfParameters());

  initialParameters[0] = 0.0; // Initial offset in mm along X
  initialParameters[1] = 0.0; // Initial offset in mm along Y

  registration->SetInitialTransformParameters(initialParameters);

  //  We should now define the number of spatial samples to be considered in
  //  the metric computation. Note that we were forced to postpone this setting
  //  until we had done the preprocessing of the images because the number of
  //  samples is usually defined as a fraction of the total number of pixels in
  //  the fixed image.
  //
  //  The number of spatial samples can usually be as low as $1\%$ of the total
```

(continues on next page)

(continued from previous page)

```
//  number of pixels in the fixed image. Increasing the number of samples
//  improves the smoothness of the metric from one iteration to another and
//  therefore helps when this metric is used in conjunction with optimizers
//  that rely of the continuity of the metric values. The trade-off, of
//  course, is that a larger number of samples result in longer computation
//  times per every evaluation of the metric.
//
//  It has been demonstrated empirically that the number of samples is not a
//  critical parameter for the registration process. When you start fine
//  tuning your own registration process, you should start using high values
//  of number of samples, for example in the range of 20% to 50% of the
//  number of pixels in the fixed image. Once you have succeeded to register
//  your images you can then reduce the number of samples progressively until
//  you find a good compromise on the time it takes to compute one evaluation
//  of the Metric. Note that it is not useful to have very fast evaluations
//  of the Metric if the noise in their values results in more iterations
//  being required by the optimizer to converge.
//  behavior of the metric values as the iterations progress.
const unsigned int numberOfPixels = fixedImageRegion.GetNumberOfPixels();

const auto numberOfSamples = static_cast<unsigned int>(numberOfPixels * 0.01);

metric->SetNumberOfSpatialSamples(numberOfSamples);

// For consistent results when regression testing.
metric->ReinitializeSeed(121212);


//  Since larger values of mutual information indicate better matches than
//  smaller values, we need to maximize the cost function in this example.
//  By default the GradientDescentOptimizer class is set to minimize the
//  value of the cost-function. It is therefore necessary to modify its
//  default behavior by invoking the MaximizeOn() method.
//  Additionally, we need to define the optimizer's step size using the
//  SetLearningRate() method.
optimizer->SetNumberOfIterations(200);
optimizer->MaximizeOn();

// Note that large values of the learning rate will make the optimizer
// unstable. Small values, on the other hand, may result in the optimizer
// needing too many iterations in order to walk to the extrema of the cost
// function. The easy way of fine tuning this parameter is to start with
// small values, probably in the range of {5.0, 10.0}. Once the other
// registration parameters have been tuned for producing convergence, you
// may want to revisit the learning rate and start increasing its value until
// you observe that the optimization becomes unstable.  The ideal value for
// this parameter is the one that results in a minimum number of iterations
// while still keeping a stable path on the parametric space of the
// optimization. Keep in mind that this parameter is a multiplicative factor
// applied on the gradient of the Metric. Therefore, its effect on the
// optimizer step length is proportional to the Metric values themselves.
// Metrics with large values will require you to use smaller values for the
// learning rate in order to maintain a similar optimizer behavior.
optimizer->SetLearningRate(15.0);

CommandIterationUpdate::Pointer observer = CommandIterationUpdate::New();
optimizer->AddObserver(itk::IterationEvent(), observer);
```

(continues on next page)

(continued from previous page)

```
  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition: " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
  catch (itk::ExceptionObject & err)
  {
    std::cout << "ExceptionObject caught !" << std::endl;
    std::cout << err << std::endl;
    return EXIT_FAILURE;
  }

  ParametersType finalParameters = registration->GetLastTransformParameters();

  double TranslationAlongX = finalParameters[0];
  double TranslationAlongY = finalParameters[1];

  unsigned int numberOfIterations = optimizer->GetCurrentIteration();

  double bestValue = optimizer->GetValue();

  // Print out results
  std::cout << std::endl;
  std::cout << "Result = " << std::endl;
  std::cout << " Translation X = " << TranslationAlongX << std::endl;
  std::cout << " Translation Y = " << TranslationAlongY << std::endl;
  std::cout << " Iterations    = " << numberOfIterations << std::endl;
  std::cout << " Metric value  = " << bestValue << std::endl;
  std::cout << " Numb. Samples = " << numberOfSamples << std::endl;


  using ResampleFilterType = itk::ResampleImageFilter<MovingImageType, FixedImageType>
↪;

  TransformType::Pointer finalTransform = TransformType::New();

  finalTransform->SetParameters(finalParameters);
  finalTransform->SetFixedParameters(transform->GetFixedParameters());

  ResampleFilterType::Pointer resample = ResampleFilterType::New();

  resample->SetTransform(finalTransform);
  resample->SetInput(movingImageReader->GetOutput());

  FixedImageType::Pointer fixedImage = fixedImageReader->GetOutput();

  resample->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resample->SetOutputOrigin(fixedImage->GetOrigin());
  resample->SetOutputSpacing(fixedImage->GetSpacing());
  resample->SetOutputDirection(fixedImage->GetDirection());
  resample->SetDefaultPixelValue(100);


  using OutputPixelType = unsigned char;
```

(continues on next page)

(continued from previous page)

```
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using CastFilterType = itk::CastImageFilter<FixedImageType, OutputImageType>;

  using WriterType = itk::ImageFileWriter<OutputImageType>;

  WriterType::Pointer     writer = WriterType::New();
  CastFilterType::Pointer caster = CastFilterType::New();

  writer->SetFileName(outputImageFile);
  caster->SetInput(resample->GetOutput());
  writer->SetInput(caster->GetOutput());
  writer->Update();


  // Generate checkerboards before and after registration
  using CheckerBoardFilterType = itk::CheckerBoardImageFilter<FixedImageType>;

  CheckerBoardFilterType::Pointer checker = CheckerBoardFilterType::New();

  checker->SetInput1(fixedImage);
  checker->SetInput2(resample->GetOutput());

  caster->SetInput(checker->GetOutput());
  writer->SetInput(caster->GetOutput());

  // Before registration
  TransformType::Pointer identityTransform = TransformType::New();
  identityTransform->SetIdentity();
  resample->SetTransform(identityTransform);

  if (argc > 4)
  {
    writer->SetFileName(checkerBoardBefore);
  }

  // After registration
  resample->SetTransform(finalTransform);
  if (argc > 5)
  {
    writer->SetFileName(checkerBoardAfter);
  }

  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**>
**class MutualInformationImageToImageMetric** : **public** itk::ImageToImageMetric<*TFixedImage*, *TMovingImage*>
Computes the mutual information between two images to be registered.

MutualInformationImageToImageMetric computes the mutual information between a fixed and moving image to be registered.

This class is templated over the FixedImage type and the MovingImage type.

The fixed and moving images are set via methods SetFixedImage() and SetMovingImage(). This metric makes use of user specified Transform and Interpolator. The Transform is used to map points from the fixed image to the moving image domain. The Interpolator is used to evaluate the image intensity at user specified geometric points in the moving image. The Transform and Interpolator are set via methods SetTransform() and SetInterpolator().

The method GetValue() computes of the mutual information while method GetValueAndDerivative() computes both the mutual information and its derivatives with respect to the transform parameters.

**Warning** This metric assumes that the moving image has already been connected to the interpolator outside of this class.

The calculations are based on the method of Viola and Wells where the probability density distributions are estimated using Parzen windows.

By default a Gaussian kernel is used in the density estimation. Other option include Cauchy and spline-based. A user can specify the kernel passing in a pointer a KernelFunctionBase using the SetKernelFunction() method.

Mutual information is estimated using two sample sets: one to calculate the singular and joint pdf's and one to calculate the entropy integral. By default 50 samples points are used in each set. Other values can be set via the SetNumberOfSpatialSamples() method.

Quality of the density estimate depends on the choice of the kernel's standard deviation. Optimal choice will depend on the images. It is can be shown that around the optimal variance, the mutual information estimate is relatively insensitive to small changes of the standard deviation. In our experiments, we have found that a standard deviation of 0.4 works well for images normalized to have a mean of zero and standard deviation of 1.0. The variance can be set via methods SetFixedImageStandardDeviation() and SetMovingImageStandardDeviation().

Implementaton of this class is based on: Viola, P. and Wells III, W. (1997). "Alignment by Maximization of Mutual Information" International Journal of Computer Vision, 24(2):137-154

**See** KernelFunctionBase

**See** GaussianKernelFunction

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Mutual Information
- Mutual Information Affine

See itk::MutualInformationImageToImageMetric for additional documentation.

### Register Image to Another Using Landmarks

### Synopsis

Rigidly register one image to another using manually specified landmarks.

### Results

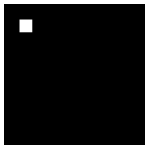


Fig. 366: fixed.png

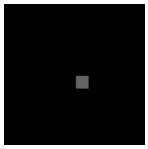


Fig. 367: moving.png

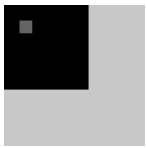


Fig. 368: output.png

### Code

### C++

```
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkVector.h"
#include "itkResampleImageFilter.h"
#include "itkLandmarkBasedTransformInitializer.h"
#include "itkRigid2DTransform.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned char;
using ImageType = itk::Image<PixelType, Dimension>;

static void
CreateFixedImage(ImageType::Pointer image);
static void
CreateMovingImage(ImageType::Pointer image);
```

(continues on next page)

(continued from previous page)

```
int
main(int itkNotUsed(argc), char * itkNotUsed(argv)[])
{
  ImageType::Pointer fixedImage = ImageType::New();
  CreateFixedImage(fixedImage);

  ImageType::Pointer movingImage = ImageType::New();
  CreateMovingImage(movingImage);

  using Rigid2DTransformType = itk::Rigid2DTransform<double>;
  using LandmarkBasedTransformInitializerType =
    itk::LandmarkBasedTransformInitializer<Rigid2DTransformType, ImageType, ImageType>
↪;

  LandmarkBasedTransformInitializerType::Pointer landmarkBasedTransformInitializer =
    LandmarkBasedTransformInitializerType::New();
  //  Create source and target landmarks.
  using LandmarkContainerType = LandmarkBasedTransformInitializerType::
↪LandmarkPointContainer;
  using LandmarkPointType = LandmarkBasedTransformInitializerType::LandmarkPointType;

  LandmarkContainerType fixedLandmarks;
  LandmarkContainerType movingLandmarks;

  LandmarkPointType fixedPoint;
  LandmarkPointType movingPoint;

  fixedPoint[0] = 10;
  fixedPoint[1] = 10;
  movingPoint[0] = 50;
  movingPoint[1] = 50;
  fixedLandmarks.push_back(fixedPoint);
  movingLandmarks.push_back(movingPoint);

  fixedPoint[0] = 10;
  fixedPoint[1] = 20;
  movingPoint[0] = 50;
  movingPoint[1] = 60;
  fixedLandmarks.push_back(fixedPoint);
  movingLandmarks.push_back(movingPoint);

  fixedPoint[0] = 20;
  fixedPoint[1] = 10;
  movingPoint[0] = 60;
  movingPoint[1] = 50;
  fixedLandmarks.push_back(fixedPoint);
  movingLandmarks.push_back(movingPoint);

  fixedPoint[0] = 20;
  fixedPoint[1] = 20;
  movingPoint[0] = 60;
  movingPoint[1] = 60;
  fixedLandmarks.push_back(fixedPoint);
  movingLandmarks.push_back(movingPoint);

  landmarkBasedTransformInitializer->SetFixedLandmarks(fixedLandmarks);
```

(continues on next page)

(continued from previous page)

```
  landmarkBasedTransformInitializer->SetMovingLandmarks(movingLandmarks);

  Rigid2DTransformType::Pointer transform = Rigid2DTransformType::New();

  transform->SetIdentity();
  landmarkBasedTransformInitializer->SetTransform(transform);
  landmarkBasedTransformInitializer->InitializeTransform();

  using ResampleFilterType = itk::ResampleImageFilter<ImageType, ImageType, double>;
  ResampleFilterType::Pointer resampleFilter = ResampleFilterType::New();
  resampleFilter->SetInput(movingImage);
  resampleFilter->SetTransform(transform);
  resampleFilter->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resampleFilter->SetOutputOrigin(fixedImage->GetOrigin());
  resampleFilter->SetOutputSpacing(fixedImage->GetSpacing());
  resampleFilter->SetOutputDirection(fixedImage->GetDirection());
  resampleFilter->SetDefaultPixelValue(200);
  resampleFilter->GetOutput();

  // Write the output
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(resampleFilter->GetOutput());
  writer->SetFileName("output.png");
  writer->Update();

  return EXIT_SUCCESS;
}

void
CreateFixedImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 10 && imageIterator.GetIndex()[0] < 20 &&␣
↪imageIterator.GetIndex()[1] > 10 &&
        imageIterator.GetIndex()[1] < 20)
    {
      imageIterator.Set(255);
    }
```

(continues on next page)

(continued from previous page)

```
    ++imageIterator;
  }

  // Write the deformation field
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(image);
  writer->SetFileName("fixed.png");
  writer->Update();
}


void
CreateMovingImage(ImageType::Pointer image)
{
  // Create a black image with a white square
  ImageType::IndexType start;
  start.Fill(0);

  ImageType::SizeType size;
  size.Fill(100);

  ImageType::RegionType region;
  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(0);

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 60 &&␣
↪imageIterator.GetIndex()[1] > 50 &&
        imageIterator.GetIndex()[1] < 60)
    {
      imageIterator.Set(100);
    }
    ++imageIterator;
  }

  // Write the deformation field
  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetInput(image);
  writer->SetFileName("moving.png");
  writer->Update();
}
```

### Classes demonstrated

template<typename **TTransform**, typename **TFixedImage** = itk::ImageBase<*TTransform*::InputSpaceDimension>, typename **TMo**
**class LandmarkBasedTransformInitializer** : **public** itk::*Object*

This class computes the transform that aligns the fixed and moving images given a set of pair landmarks. The class is templated over the Transform type as well as fixed image and moving image types. The transform computed gives the best fit transform that maps the fixed and moving images in a least squares sense. The indices are taken to correspond, so point 1 in the first set will get mapped close to point 1 in the second set, etc.

Currently, the following transforms are supported by the class: VersorRigid3DTransform Rigid2DTransform AffineTransform BSplineTransform

An equal number of fixed and moving landmarks need to be specified using SetFixedLandmarks() and SetMovingLandmarks(). Any number of landmarks may be specified. In the case of the Affine transformation the number of landmarks must be greater than the landmark dimensionality. If this is not the case an exception is thrown. In the case of the VersorRigid3DTransform and Rigid2DTransform the number of landmarks must be equal or greater than the landmark dimensionality. If this is not the case, only the translational component of the transformation is computed and the rotation is the identity. In the case of using Affine or BSpline transforms, each landmark pair can contribute in the final transform based on its defined weight. Number of weights should be equal to the number of landmarks and can be specified using SetLandmarkWeight(). By defaults are weights are set to one. Call InitializeTransform() to initialize the transform.

The class is based in part on Hybrid/vtkLandmarkTransform originally implemented in python by David G. Gobbi.

The solution is based on Berthold K. P. Horn (1987), "Closed-form solution of absolute orientation

using unit quaternions," http://people.csail.mit.edu/bkph/papers/Absolute_Orientation.pdf

The Affine Transform initializer is based on an algorithm by H Spaeth, and is described in the Insight Journal Article "Affine Transformation for Landmark Based Registration Initializer

in ITK" by Kim E.Y., Johnson H., Williams N. available at http://midasjournal.com/browse/publication/825

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Register Image To Another Using Landmarks

See itk::LandmarkBasedTransformInitializer for additional documentation.

### Watch Registration

### Synopsis

Watch the iterations of a registration using VTK.

### Results

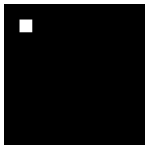


Fig. 369: fixed.png

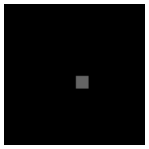


Fig. 370: moving.png

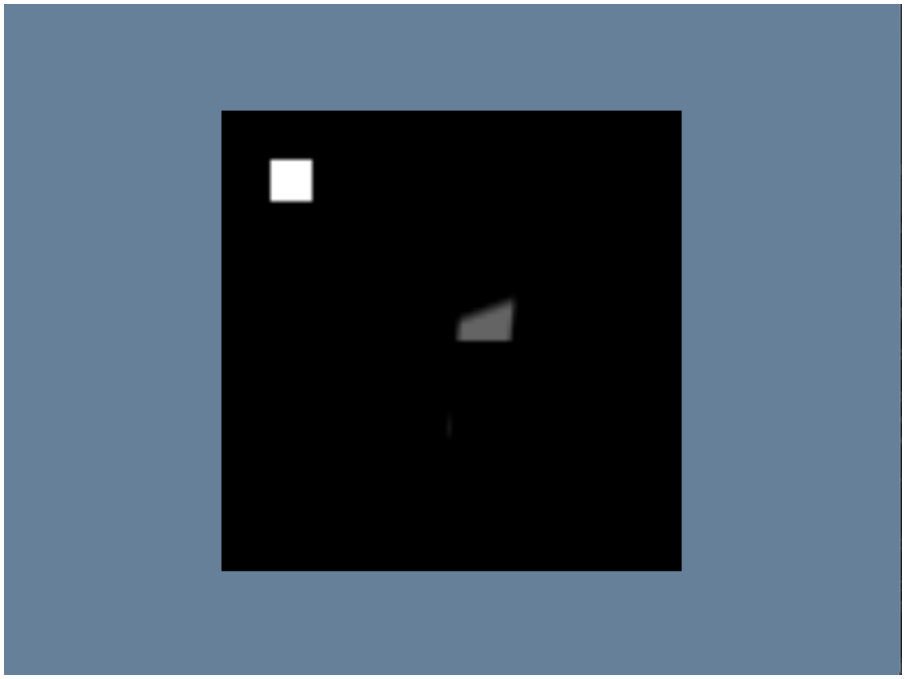


Fig. 371: Output In VTK Window

Output:

```
Optimizer stop condition: RegularStepGradientDescentOptimizer: Maximum number of␣
↪iterations (1000) exceeded.
Final Transform: AffineTransform (0x7f9129d00120)
RTTI typeinfo:   itk::AffineTransform<double, 2u>
Reference Count: 4
Modified Time: 215110
Debug: Off
Object Name:
Observers:
  none
```

(continues on next page)

(continued from previous page)

```
Matrix:
  0.789218 0.062097
  0.139299 0.37936
Offset: [7.3411, 26.2747]
Center: [49.5, 49.5]
Translation: [-0.0188083, 2.44832]
Inverse:
  1.30477 -0.213577
  -0.479108 2.71444
Singular: 0

Final Parameters: [0.7892180100803599, 0.06209699937125816, 0.13929938284463836, 0.
↪37935986885403217, -0.018808339095318795, 2.448232274873905]

Result =
Iterations    = 1000
Metric value  = -0.00195192
Numb. Samples = 500
```

### Code

### C++

```
#include "itkImageRegistrationMethod.h"
#include "itkAffineTransform.h"
#include "itkMattesMutualInformationImageToImageMetric.h"
#include "itkRegularStepGradientDescentOptimizer.h"
#include "itkNormalizeImageFilter.h"
#include "itkDiscreteGaussianImageFilter.h"
#include "itkResampleImageFilter.h"
#include "itkCheckerBoardImageFilter.h"
#include "itkFlipImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageToVTKImageFilter.h"
#include "itkCenteredTransformInitializer.h"

#include "vtkVersion.h"

#include "vtkSmartPointer.h"
#include "vtkRenderWindow.h"
#include "vtkRenderer.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkImageActor.h"
#include "vtkImageMapper3D.h"

constexpr unsigned int Dimension = 2;
using PixelType = unsigned char;

using InputImageType = itk::Image<PixelType, Dimension>;
using OutputImageType = itk::Image<unsigned char, Dimension>;

//  Command observer to visualize the evolution of the registration process.
//
```

(continues on next page)

(continued from previous page)

```
#include "itkCommand.h"
template <typename TImage>
class IterationUpdate : public itk::Command
{
public:
  using Self = IterationUpdate;
  using Superclass = itk::Command;
  using Pointer = itk::SmartPointer<Self>;
  itkNewMacro(Self);

protected:
  IterationUpdate() = default;

public:
  using InternalImageType = itk::Image<float, 2>;
  using OptimizerType = itk::RegularStepGradientDescentOptimizer;
  using OptimizerPointer = const OptimizerType *;
  using ResampleFilterType = itk::ResampleImageFilter<TImage, TImage>;
  using TransformType = itk::AffineTransform<double, 2>;
  using ConnectorType = itk::ImageToVTKImageFilter<TImage>;
  using FilterType = itk::FlipImageFilter<TImage>;
  void
  Execute(itk::Object * caller, const itk::EventObject & event) override
  {
    Execute((const itk::Object *)caller, event);
  }

  void
  Execute(const itk::Object * object, const itk::EventObject & event) override
  {
    auto optimizer = static_cast<OptimizerPointer>(object);
    if (!(itk::IterationEvent().CheckEvent(&event)))
    {
      return;
    }

    m_Transform->SetParameters(optimizer->GetCurrentPosition());
    m_Filter->Update();
    m_Connector->SetInput(m_Filter->GetOutput());
    m_Connector->Update();

#if VTK_MAJOR_VERSION <= 5
    m_ImageActor->SetInput(m_Connector->GetOutput());
#else
    m_Connector->Update();
    m_ImageActor->GetMapper()->SetInputData(m_Connector->GetOutput());
#endif
    m_RenderWindow->Render();
  }
  void
  SetTransform(TransformType::Pointer & transform)
  {
    m_Transform = transform;
  }
  void
  SetFilter(typename FilterType::Pointer & filter)
  {
```

(continues on next page)

(continued from previous page)

```
    m_Filter = filter;
  }
  void
  SetConnector(typename ConnectorType::Pointer & connector)
  {
    m_Connector = connector;
  }
  void
  SetImageActor(vtkImageActor * actor)
  {
    m_ImageActor = actor;
  }
  void
  SetRenderWindow(vtkRenderWindow * renderWindow)
  {
    m_RenderWindow = renderWindow;
  }
  TransformType::Pointer          m_Transform;
  typename FilterType::Pointer    m_Filter;
  typename ConnectorType::Pointer m_Connector;
  vtkImageActor *                 m_ImageActor;
  vtkRenderWindow *               m_RenderWindow;
};

int
main(int argc, char * argv[])
{
  InputImageType::Pointer fixedImage = InputImageType::New();
  InputImageType::Pointer movingImage = InputImageType::New();

  if (argc > 2)
  {
    using ReaderType = itk::ImageFileReader<InputImageType>;
    ReaderType::Pointer fixedReader = ReaderType::New();
    fixedReader->SetFileName(argv[1]);
    fixedReader->Update();
    fixedImage = fixedReader->GetOutput();

    ReaderType::Pointer movingReader = ReaderType::New();
    movingReader->SetFileName(argv[2]);
    movingReader->Update();
    movingImage = movingReader->GetOutput();
  }
  else
  {
    std::cout << "Usage: " << argv[0] << " fixedImage movingImage" << std::endl;
    return EXIT_FAILURE;
  }

  // Use floats internally
  using InternalImageType = itk::Image<float, Dimension>;

  // Normalize the images
  using NormalizeFilterType = itk::NormalizeImageFilter<InputImageType,␣
↪InternalImageType>;
  NormalizeFilterType::Pointer fixedNormalizer = NormalizeFilterType::New();
  NormalizeFilterType::Pointer movingNormalizer = NormalizeFilterType::New();
```

(continues on next page)

(continued from previous page)

```
  fixedNormalizer->SetInput(fixedImage);
  movingNormalizer->SetInput(movingImage);

  // Smooth the normalized images
  using GaussianFilterType = itk::DiscreteGaussianImageFilter<InternalImageType,␣
↪InternalImageType>;
  GaussianFilterType::Pointer fixedSmoother = GaussianFilterType::New();
  GaussianFilterType::Pointer movingSmoother = GaussianFilterType::New();

  fixedSmoother->SetVariance(3.0);
  fixedSmoother->SetInput(fixedNormalizer->GetOutput());
  movingSmoother->SetVariance(3.0);
  movingSmoother->SetInput(movingNormalizer->GetOutput());

  // Set up registration
  using TransformType = itk::AffineTransform<double, Dimension>;
  using OptimizerType = itk::RegularStepGradientDescentOptimizer;
  using InterpolatorType = itk::LinearInterpolateImageFunction<InternalImageType,␣
↪double>;
  using MetricType = itk::MattesMutualInformationImageToImageMetric<InternalImageType,
↪ InternalImageType>;
  using RegistrationType = itk::ImageRegistrationMethod<InternalImageType,␣
↪InternalImageType>;
  using InitializerType = itk::CenteredTransformInitializer<TransformType,␣
↪InputImageType, InputImageType>;

  InitializerType::Pointer  initializer = InitializerType::New();
  TransformType::Pointer    transform = TransformType::New();
  OptimizerType::Pointer    optimizer = OptimizerType::New();
  InterpolatorType::Pointer interpolator = InterpolatorType::New();
  RegistrationType::Pointer registration = RegistrationType::New();

  // Set up the registration framework
  initializer->SetFixedImage(fixedImage);
  initializer->SetMovingImage(movingImage);
  initializer->SetTransform(transform);

  transform->SetIdentity();
  initializer->GeometryOn();
  initializer->InitializeTransform();

  registration->SetOptimizer(optimizer);
  registration->SetTransform(transform);
  registration->SetInterpolator(interpolator);

  MetricType::Pointer metric = MetricType::New();

  registration->SetMetric(metric);
  registration->SetFixedImage(fixedSmoother->GetOutput());
  registration->SetMovingImage(movingSmoother->GetOutput());

  // Update to get the size of the region
  fixedNormalizer->Update();
  InputImageType::RegionType fixedImageRegion = fixedNormalizer->GetOutput()->
↪GetBufferedRegion();
  registration->SetFixedImageRegion(fixedImageRegion);
```

(continues on next page)

(continued from previous page)

```
  using ParametersType = RegistrationType::ParametersType;
  ParametersType initialParameters(transform->GetNumberOfParameters());

  // rotation matrix (identity)
  initialParameters[0] = 1.0; // R(0,0)
  initialParameters[1] = 0.0; // R(0,1)
  initialParameters[2] = 0.0; // R(1,0)
  initialParameters[3] = 1.0; // R(1,1)

  // translation vector
  initialParameters[4] = 0.0;
  initialParameters[5] = 0.0;

  registration->SetInitialTransformParameters(initialParameters);

  const unsigned int numberOfPixels = fixedImageRegion.GetNumberOfPixels();
  const auto         numberOfSamples = static_cast<unsigned int>(numberOfPixels * 0.
↪05);

  metric->SetNumberOfHistogramBins(26);
  metric->SetNumberOfSpatialSamples(numberOfSamples);

  optimizer->MinimizeOn();
  optimizer->SetMaximumStepLength(0.500);
  optimizer->SetMinimumStepLength(0.001);
  optimizer->SetNumberOfIterations(1000);

  const unsigned int numberOfParameters = transform->GetNumberOfParameters();
  using OptimizerScalesType = OptimizerType::ScalesType;
  OptimizerScalesType optimizerScales(numberOfParameters);
  double              translationScale = 1.0 / 1000.0;
  optimizerScales[0] = 1.0;
  optimizerScales[1] = 1.0;
  optimizerScales[2] = 1.0;
  optimizerScales[3] = 1.0;
  optimizerScales[4] = translationScale;
  optimizerScales[5] = translationScale;

  optimizer->SetScales(optimizerScales);

  TransformType::Pointer finalTransform = TransformType::New();
  finalTransform->SetParameters(initialParameters);
  finalTransform->SetFixedParameters(transform->GetFixedParameters());

  using ResampleFilterType = itk::ResampleImageFilter<InputImageType, InputImageType>;
  ResampleFilterType::Pointer resample = ResampleFilterType::New();
  resample->SetTransform(finalTransform);
  resample->SetInput(movingImage);
  resample->SetSize(fixedImage->GetLargestPossibleRegion().GetSize());
  resample->SetOutputOrigin(fixedImage->GetOrigin());
  resample->SetOutputSpacing(fixedImage->GetSpacing());
  resample->SetOutputDirection(fixedImage->GetDirection());
  resample->SetDefaultPixelValue(100);
  resample->Update();

  // Set up the visualization pipeline
```

(continues on next page)

(continued from previous page)

```
  using CheckerBoardFilterType = itk::CheckerBoardImageFilter<InputImageType>;
  CheckerBoardFilterType::Pointer          checkerboard = CheckerBoardFilterType::
↪New();
  CheckerBoardFilterType::PatternArrayType pattern;
  pattern[0] = 4;
  pattern[1] = 4;

  checkerboard->SetCheckerPattern(pattern);
  checkerboard->SetInput1(fixedImage);
  checkerboard->SetInput2(resample->GetOutput());

  using FlipFilterType = itk::FlipImageFilter<InputImageType>;
  FlipFilterType::Pointer flip = FlipFilterType::New();
  bool                    flipAxes[3] = { false, true, false };
  flip->SetFlipAxes(flipAxes);
  flip->SetInput(checkerboard->GetOutput());
  flip->Update();

  // VTK visualization pipeline
  using ConnectorType = itk::ImageToVTKImageFilter<InputImageType>;
  ConnectorType::Pointer connector = ConnectorType::New();
  connector->SetInput(flip->GetOutput());

  vtkSmartPointer<vtkImageActor> actor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor->SetInput(connector->GetOutput());
#else
  connector->Update();
  actor->GetMapper()->SetInputData(connector->GetOutput());
#endif
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  vtkSmartPointer<vtkRenderer>     renderer = vtkSmartPointer<vtkRenderer>::New();
  renderer->SetBackground(.4, .5, .6);
  renderer->AddActor(actor);
  renderWindow->SetSize(640, 480);
  ;
  renderWindow->AddRenderer(renderer);
  renderWindow->Render();

  // Set up the iteration event observer
  IterationUpdate<InputImageType>::Pointer observer = IterationUpdate<InputImageType>:
↪:New();
  optimizer->AddObserver(itk::IterationEvent(), observer);

  observer->SetTransform(finalTransform);
  observer->SetFilter(flip);
  observer->SetConnector(connector);
  observer->SetImageActor(actor);
  observer->SetRenderWindow(renderWindow);
  try
  {
    registration->Update();
    std::cout << "Optimizer stop condition: " << registration->GetOptimizer()->
↪GetStopConditionDescription()
              << std::endl;
  }
```

(continues on next page)

(continued from previous page)

```
  catch (itk::ExceptionObject & err)
  {
    std::cout << "ExceptionObject caught !" << std::endl;
    std::cout << err << std::endl;
    return EXIT_FAILURE;
  }
  std::cout << "Final Transform: " << finalTransform << std::endl;

  ParametersType finalParameters = registration->GetLastTransformParameters();
  std::cout << "Final Parameters: " << finalParameters << std::endl;

  unsigned int numberOfIterations = optimizer->GetCurrentIteration();
  double       bestValue = optimizer->GetValue();

  // Print out results
  std::cout << std::endl;
  std::cout << "Result = " << std::endl;
  std::cout << " Iterations    = " << numberOfIterations << std::endl;
  std::cout << " Metric value  = " << bestValue << std::endl;
  std::cout << " Numb. Samples = " << numberOfSamples << std::endl;

  // Interact with the image
  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  vtkSmartPointer<vtkInteractorStyleImage>   style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);
  interactor->SetRenderWindow(renderWindow);
  interactor->Start();

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TFixedImage**, typename **TMovingImage**>
**class MattesMutualInformationImageToImageMetric** : **public** itk::ImageToImageMetric<*TFixedImage*, *TMovingIm*

Computes the mutual information between two images to be registered using the method of Mattes et al.

MattesMutualInformationImageToImageMetric computes the mutual information between a fixed and moving image to be registered.

This class is templated over the FixedImage type and the MovingImage type.

The fixed and moving images are set via methods SetFixedImage() and SetMovingImage(). This metric makes use of user specified Transform and Interpolator. The Transform is used to map points from the fixed image to the moving image domain. The Interpolator is used to evaluate the image intensity at user specified geometric points in the moving image. The Transform and Interpolator are set via methods SetTransform() and SetInterpolator().

If a BSplineInterpolationFunction is used, this class obtain image derivatives from the BSpline interpolator. Otherwise, image derivatives are computed using central differencing.

The method GetValue() computes of the mutual information while method GetValueAndDerivative() computes both the mutual information and its derivatives with respect to the transform parameters.

**Warning** This metric assumes that the moving image has already been connected to the interpolator outside of this class.

The calculations are based on the method of Mattes et al [1,2] where the probability density distribution are estimated using Parzen histograms. Since the fixed image PDF does not contribute to the derivatives, it does not need to be smooth. Hence, a zero order (box car) BSpline kernel is used for the fixed image intensity PDF. On the other hand, to ensure smoothness a third order BSpline kernel is used for the moving image intensity PDF.

On Initialize(), the FixedImage is uniformly sampled within the FixedImageRegion. The number of samples used can be set via SetNumberOfSpatialSamples(). Typically, the number of spatial samples used should increase with the image size.

The option UseAllPixelOn() disables the random sampling and uses all the pixels of the FixedImageRegion in order to estimate the joint intensity PDF.

During each call of GetValue(), GetDerivatives(), GetValueAndDerivatives(), marginal and joint intensity PDF's values are estimated at discrete position or bins. The number of bins used can be set via SetNumberOfHistogramBins(). To handle data with arbitrary magnitude and dynamic range, the image intensity is scale such that any contribution to the histogram will fall into a valid bin.

One the PDF's have been constructed, the mutual information is obtained by doubling summing over the discrete PDF values.

Notes:

a. This class returns the negative mutual information value.

References: [1] "Nonrigid multimodality image registration" D. Mattes, D. R. Haynor, H. Vesselle, T. Lewellen and W. Eubank Medical Imaging 2001: Image Processing, 2001, pp. 1609-1620. [2] "PET-CT Image Registration in the Chest Using Free-form Deformations" D. Mattes, D. R. Haynor, H. Vesselle, T. Lewellen and W. Eubank IEEE Transactions in Medical Imaging. Vol.22, No.1, January 2003. pp.120-128. [3] "Optimization of Mutual Information for MultiResolution Image

Registration" P. Thevenaz and M. Unser IEEE Transactions in Image Processing, 9(12) December 2000.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Watch Registration

See itk::MattesMutualInformationImageToImageMetric for additional documentation.

### 3.9.2 Metricsv4

#### Perform Registration on Vector Images

#### Synopsis

Register images that have multi-component pixels like an itk::VectorImage or an itk::Image< itk::Vector, Dimension >.

### Results

| | | |
|---|---|---|
|  Fixed image. |  Moving image. |  Resampled, registered moving image. |

### Code

### C++

```
#include "itkMeanSquaresImageToImageMetricv4.h"
#include "itkGradientDescentOptimizerv4.h"
#include "itkRegistrationParameterScalesFromPhysicalShift.h"
#include "itkVectorImageToImageMetricTraitsv4.h"

#include "itkGaussianSmoothingOnUpdateDisplacementFieldTransform.h"

#include "itkCastImageFilter.h"

#include "itkCommand.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include <iomanip>

int
main(int argc, char * argv[])
{

  if (argc < 4)
  {
    std::cerr << "Missing Parameters " << std::endl;
    std::cerr << "Usage: " << argv[0];
    std::cerr << " fixedImageFile movingImageFile ";
    std::cerr << " resampledMovingImageFile ";
    std::cerr << " [numberOfAffineIterations numberOfDisplacementIterations] ";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
  const char * fixedImageFile = argv[1];
  const char * movingImageFile = argv[2];
  const char * resampledMovingImageFile = argv[3];

  unsigned int numberOfAffineIterations = 2;
  unsigned int numberOfDisplacementIterations = 2;
  if (argc >= 5)
  {
    numberOfAffineIterations = std::stoi(argv[4]);
```

(continues on next page)

(continued from previous page)

```
  }
  if (argc >= 6)
  {
    numberOfDisplacementIterations = std::stoi(argv[5]);
  }

  constexpr unsigned int Dimension = 2;

  // The input images have red, blue, and green pixel components.
  constexpr unsigned int NumberOfPixelComponents = 3;
  using PixelComponentType = float;
  using InputPixelType = itk::Vector<PixelComponentType, NumberOfPixelComponents>;
  using InputImageType = itk::Image<InputPixelType, Dimension>;

  using ParametersValueType = double;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  ReaderType::Pointer fixedImageReader = ReaderType::New();
  fixedImageReader->SetFileName(fixedImageFile);
  ReaderType::Pointer movingImageReader = ReaderType::New();
  movingImageReader->SetFileName(movingImageFile);

  // Get the input images
  try
  {
    fixedImageReader->Update();
    movingImageReader->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error while reading images: " << error << std::endl;
    return EXIT_FAILURE;
  }
  InputImageType::Pointer fixedImage = fixedImageReader->GetOutput();
  InputImageType::Pointer movingImage = movingImageReader->GetOutput();

  using AffineTransformType = itk::AffineTransform<ParametersValueType, Dimension>;
  AffineTransformType::Pointer affineTransform = AffineTransformType::New();

  using DisplacementTransformType =
    itk::GaussianSmoothingOnUpdateDisplacementFieldTransform<ParametersValueType,␣
↪Dimension>;
  DisplacementTransformType::Pointer displacementTransform =␣
↪DisplacementTransformType::New();

  using DisplacementFieldType = DisplacementTransformType::DisplacementFieldType;
  DisplacementFieldType::Pointer displacementField = DisplacementFieldType::New();

  // Set the displacement field to be the same as the fixed image region, which will
  // act by default as the virtual domain in this example.
  displacementField->SetRegions(fixedImage->GetLargestPossibleRegion());
  // make sure the displacement field has the same spatial information as the image
  displacementField->CopyInformation(fixedImage);
  std::cout << "fixedImage->GetLargestPossibleRegion(): " << fixedImage->
↪GetLargestPossibleRegion() << std::endl;
  displacementField->Allocate();
  // Fill it with 0's
```

(continues on next page)

(continued from previous page)

```
  DisplacementTransformType::OutputVectorType zeroVector;
  zeroVector.Fill(0.0);
  displacementField->FillBuffer(zeroVector);
  // Assign to transform
  displacementTransform->SetDisplacementField(displacementField);
  displacementTransform->SetGaussianSmoothingVarianceForTheUpdateField(5.0);
  displacementTransform->SetGaussianSmoothingVarianceForTheTotalField(6.0);

  // Identity transform for fixed image
  using IdentityTransformType = itk::IdentityTransform<ParametersValueType, Dimension>
↪;
  IdentityTransformType::Pointer identityTransform = IdentityTransformType::New();

  // The metric
  using VirtualImageType = itk::Image<double, Dimension>;
  using MetricTraitsType =
    itk::VectorImageToImageMetricTraitsv4<InputImageType, InputImageType,␣
↪VirtualImageType, InputPixelType::Length>;
  using MetricType =
    itk::MeanSquaresImageToImageMetricv4<InputImageType, InputImageType,␣
↪VirtualImageType, double, MetricTraitsType>;
  using PointSetType = MetricType::FixedSampledPointSetType;
  MetricType::Pointer metric = MetricType::New();

  using PointType = PointSetType::PointType;
  PointSetType::Pointer                                pointSet = PointSetType::New();
  itk::ImageRegionIteratorWithIndex<InputImageType> fixedImageIt(fixedImage,␣
↪fixedImage->GetLargestPossibleRegion());
  itk::SizeValueType                                   index = 0;
  itk::SizeValueType                                   count = 0;
  for (fixedImageIt.GoToBegin(); !fixedImageIt.IsAtEnd(); ++fixedImageIt)
  {
    // take every N^th point
    if (count % 2 == 0)
    {
      PointType point;
      fixedImage->TransformIndexToPhysicalPoint(fixedImageIt.GetIndex(), point);
      pointSet->SetPoint(index, point);
      ++index;
    }
    ++count;
  }
  metric->SetFixedSampledPointSet(pointSet);
  metric->SetUseSampledPointSet(true);


  // Assign images and transforms.
  // By not setting a virtual domain image or virtual domain settings,
  // the metric will use the fixed image for the virtual domain.
  metric->SetFixedImage(fixedImage);
  metric->SetMovingImage(movingImage);
  metric->SetFixedTransform(identityTransform);
  metric->SetMovingTransform(affineTransform);
  const bool gaussian = false;
  metric->SetUseMovingImageGradientFilter(gaussian);
  metric->SetUseFixedImageGradientFilter(gaussian);
  metric->Initialize();
```

(continues on next page)

(continued from previous page)

```
  using RegistrationParameterScalesFromShiftType = itk::
↪RegistrationParameterScalesFromPhysicalShift<MetricType>;
  RegistrationParameterScalesFromShiftType::Pointer shiftScaleEstimator =
    RegistrationParameterScalesFromShiftType::New();
  shiftScaleEstimator->SetMetric(metric);


  //
  // Affine registration
  //
  std::cout << "First do an affine registration..." << std::endl;
  using OptimizerType = itk::GradientDescentOptimizerv4;
  OptimizerType::Pointer optimizer = OptimizerType::New();
  optimizer->SetMetric(metric);
  optimizer->SetNumberOfIterations(numberOfAffineIterations);
  optimizer->SetScalesEstimator(shiftScaleEstimator);
  optimizer->StartOptimization();
  std::cout << "After optimization affine parameters are: " << affineTransform->
↪GetParameters() << std::endl;

  //
  // Deformable registration
  //
  std::cout << "Now, add the displacement field to the composite transform..." << std:
↪:endl;
  using CompositeType = itk::CompositeTransform<ParametersValueType, Dimension>;
  CompositeType::Pointer compositeTransform = CompositeType::New();
  compositeTransform->AddTransform(affineTransform);
  compositeTransform->AddTransform(displacementTransform);
  // Optimize only the displacement field, but still using the
  // previously computed affine transformation.
  compositeTransform->SetOnlyMostRecentTransformToOptimizeOn();
  metric->SetMovingTransform(compositeTransform);
  metric->SetUseSampledPointSet(false);
  metric->Initialize();

  // Optimizer
  optimizer->SetScalesEstimator(shiftScaleEstimator);
  optimizer->SetMetric(metric);
  optimizer->SetNumberOfIterations(numberOfDisplacementIterations);
  try
  {
    if (numberOfDisplacementIterations > 0)
    {
      optimizer->StartOptimization();
    }
    else
    {
      std::cout << "*** Skipping displacement field optimization.\n";
    }
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  // Warp the image with the displacement field
  using ResampleFilterType = itk::ResampleImageFilter<InputImageType, InputImageType>;
  ResampleFilterType::Pointer resampler = ResampleFilterType::New();
  resampler->SetTransform(compositeTransform);
  resampler->SetInput(movingImageReader->GetOutput());
  resampler->SetReferenceImage(fixedImage);
  resampler->SetUseReferenceImage(true);

  // Write the warped image into a file
  using OutputPixelType = itk::Vector<unsigned char, NumberOfPixelComponents>;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;
  using CastFilterType = itk::CastImageFilter<InputImageType, OutputImageType>;
  CastFilterType::Pointer caster = CastFilterType::New();
  caster->SetInput(resampler->GetOutput());

  using WriterType = itk::ImageFileWriter<OutputImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(resampledMovingImageFile);
  writer->SetInput(caster->GetOutput());
  try
  {
    writer->UpdateLargestPossibleRegion();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TFixedImageType**, typename **TMovingImageType**, typename **TVirtualImageType**, unsigned int **Numb**
**class VectorImageToImageMetricTraitsv4**

A simple structure holding type information for ImageToImageMetricv4 classes.

This class provides type information for class members and methods used in gradient calculation. This class is used for images with vector pixel types, including VectorImage. For images with scalar pixel types, see itkDefaultImageToImageMetricTraitsv4.

**See** itkDefaultImageToImageMetricTraitsv4

See itk::VectorImageToImageMetricTraitsv4 for additional documentation.

## Register Two Point Sets

## Index

**See also:**

registration; transformation; pointset

## Register Two Point Sets

Similar to image registration, an n-dimensional "moving" point set may be resampled to align with a "fixed" point set. An ITK point set metric may be employed with an ITK optimizer in order to register the two sets.

In this example we create two `itk.PointSet` representations with an arbitrary offset and select parameters to align them with a `TranslationTransform`. We use the `JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4` class to quantify the difference between point sets and the `GradientDescentOptimizerv4` class to iteratively reduce this difference by modifying transform parameters. Our example also includes sample code visualizing the parameter surface with `matplotlib` and `itkwidgets` as well as a sample hyperparameter search to optimize gradient descent performance.

[1]:
```
import os
import sys
import itertools
from math import pi, sin, cos, sqrt

import matplotlib.pyplot as plt
import numpy as np

import itk
from itkwidgets import view
```

## Construct Two Point Sets

[2]:
```
# Generate two circles with a small offset
def make_circles(dimension:int=2,offset:list=None):

    PointSetType = itk.PointSet[itk.F, dimension]

    RADIUS = 100

    if not offset or len(offset) != dimension:
        offset = [2.0] * dimension

    fixed_points = PointSetType.New()
    moving_points = PointSetType.New()
    fixed_points.Initialize()
    moving_points.Initialize()

    count = 0
    step = 0.1
    for count in range(0, int(2 * pi / step) + 1):

        theta = count * step
```

(continues on next page)

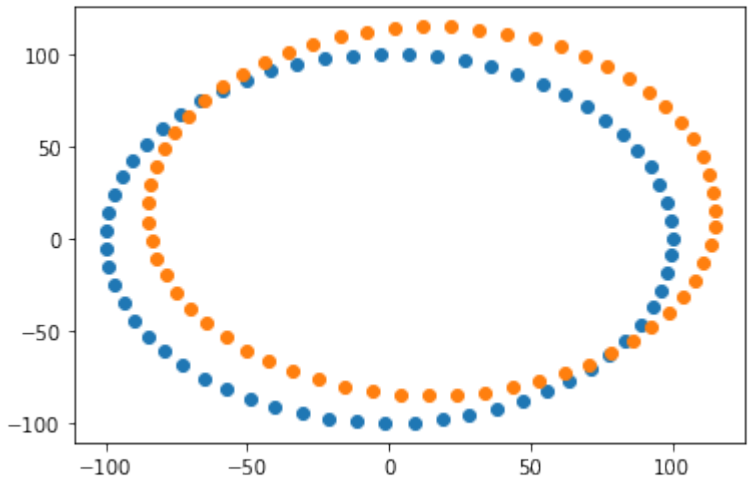
(continued from previous page)

```
        fixed_point = list()
        fixed_point.append(RADIUS * cos(theta))
        for dim in range(1,dimension):
            fixed_point.append(RADIUS * sin(theta))
        fixed_points.SetPoint(count, fixed_point)

        moving_point = [fixed_point[dim] + offset[dim]
                        for dim in range(0,dimension)]
        moving_points.SetPoint(count, moving_point)

    return fixed_points, moving_points
```

[3]:
```
POINT_SET_OFFSET = [15.0, 15.0]
fixed_set, moving_set = make_circles(offset=POINT_SET_OFFSET)
```

[4]:
```
# Visualize point sets with matplotlib

fig = plt.figure()
ax = plt.axes()

n_points = fixed_set.GetNumberOfPoints()
ax.scatter([fixed_set.GetPoint(i)[0] for i in range(0,n_points)],
           [fixed_set.GetPoint(i)[1] for i in range(0,n_points)])
ax.scatter([moving_set.GetPoint(i)[0] for i in range(0,n_points)],
           [moving_set.GetPoint(i)[1] for i in range(0,n_points)])
```

[4]:
```
<matplotlib.collections.PathCollection at 0x1d48559e790>
```



### Run Gradient Descent Optimization

We will quantify the point set offset with `JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4` and minimize the metric value over 10 gradient descent iterations.

[5]:
```
ExhaustiveOptimizerType = itk.ExhaustiveOptimizerv4[itk.D]
```

[6]:
```
dim = 2

# Define translation parameters to update iteratively
TransformType = itk.TranslationTransform[itk.D, dim]
transform = TransformType.New()
transform.SetIdentity()
```

[7]:
```
PointSetType = type(fixed_set)

# Define a metric to reflect the difference between point sets
PointSetMetricType = itk.
↪JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4[PointSetType]
metric = PointSetMetricType.New(
        FixedPointSet=fixed_set,
        MovingPointSet=moving_set,
        MovingTransform=transform,
        PointSetSigma=5.0,
        KernelSigma=10.0,
        UseAnisotropicCovariances=False,
        CovarianceKNeighborhood=5,
        EvaluationKNeighborhood=10,
        Alpha=1.1)
metric.Initialize()
```

[8]:
```
# Define an estimator to help determine step sizes along each transform parameter2
ShiftScalesType = itk.RegistrationParameterScalesFromPhysicalShift[PointSetMetricType]
shift_scale_estimator = ShiftScalesType.New(
    Metric=metric,
    VirtualDomainPointSet=metric.GetVirtualTransformedPointSet(),
    TransformForward=True
    )
```

[9]:
```
max_iterations = 10

# Define the gradient descent optimzer
OptimizerType = itk.GradientDescentOptimizerv4Template[itk.D]
optimizer = OptimizerType.New(
        Metric=metric,
        NumberOfIterations=max_iterations,
        ScalesEstimator=shift_scale_estimator,
        MaximumStepSizeInPhysicalUnits=8.0,
        MinimumConvergenceValue=-1,
        DoEstimateLearningRateAtEachIteration=False,
        DoEstimateLearningRateOnce=True,
        ReturnBestParametersAndValue=True)
```

[10]:
```
iteration_data = dict()
```

(continues on next page)

(continued from previous page)

```
# Track gradient descent iterations with observers
def print_iteration():
    print(f'It: {optimizer.GetCurrentIteration()}'
          f' metric value: {optimizer.GetCurrentMetricValue():.6f} '
          #f' transform position: {list(optimizer.GetCurrentPosition())}'
          f' learning rate: {optimizer.GetLearningRate()}')

def log_iteration():
    iteration_data[optimizer.GetCurrentIteration() + 1] = list(optimizer.
↪GetCurrentPosition())

optimizer.AddObserver(itk.AnyEvent(), print_iteration)
optimizer.AddObserver(itk.IterationEvent(), log_iteration)

# Set first value to default transform position
iteration_data[0] = list(optimizer.GetCurrentPosition())
```

[11]:
```
# Run optimization and print out results
optimizer.StartOptimization()

print(f'Number of iterations: {optimizer.GetCurrentIteration() - 1}')
print(f'Moving-source final value: {optimizer.GetCurrentMetricValue()}')
print(f'Moving-source final position: {list(optimizer.GetCurrentPosition())}')
print(f'Optimizer scales: {list(optimizer.GetScales())}')
print(f'Optimizer learning rate: {optimizer.GetLearningRate()}')
print(f'Stop reason: {optimizer.GetStopConditionDescription()}')
```

```
It: 0 metric value: 0.000000  learning rate: 1.0
It: 0 metric value: -0.043464  learning rate: 5594.753388446298
It: 1 metric value: -0.054787  learning rate: 5594.753388446298
It: 2 metric value: -0.062597  learning rate: 5594.753388446298
It: 3 metric value: -0.064588  learning rate: 5594.753388446298
It: 4 metric value: -0.064807  learning rate: 5594.753388446298
It: 5 metric value: -0.064815  learning rate: 5594.753388446298
It: 6 metric value: -0.064815  learning rate: 5594.753388446298
It: 7 metric value: -0.064815  learning rate: 5594.753388446298
It: 8 metric value: -0.064815  learning rate: 5594.753388446298
It: 9 metric value: -0.064815  learning rate: 5594.753388446298
It: 10 metric value: -0.064815  learning rate: 5594.753388446298
Number of iterations: 9
Moving-source final value: -0.06481531061643396
Moving-source final position: [15.000412861069881, 14.99997463945473]
Optimizer scales: [1.0000000000010232, 1.0000000000010232]
Optimizer learning rate: 5594.753388446298
Stop reason: GradientDescentOptimizerv4Template: Maximum number of iterations (10)␣
↪exceeded.
```

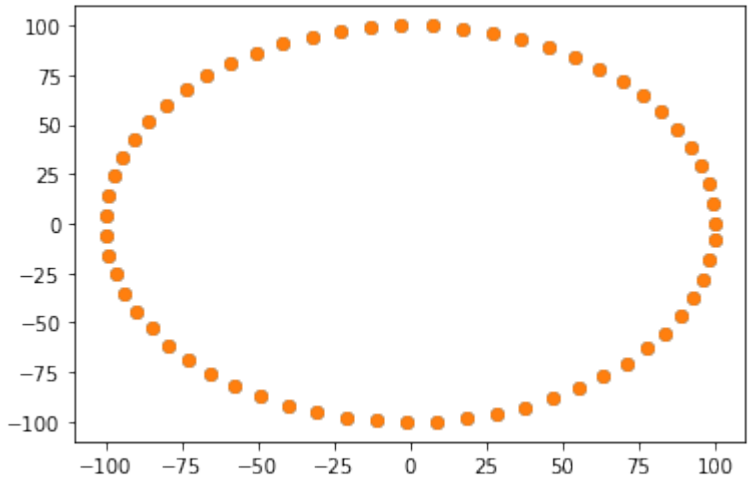
### Resample Moving Point Set

[12]:
```
moving_inverse = metric.GetMovingTransform().GetInverseTransform()
fixed_inverse = metric.GetFixedTransform().GetInverseTransform()
```

[13]:
```
transformed_fixed_set = PointSetType.New()
transformed_moving_set = PointSetType.New()

for n in range(0,metric.GetNumberOfComponents()):
    transformed_moving_point = moving_inverse.TransformPoint(moving_set.GetPoint(n))
    transformed_moving_set.SetPoint(n,transformed_moving_point)

    transformed_fixed_point = fixed_inverse.TransformPoint(fixed_set.GetPoint(n))
    transformed_fixed_set.SetPoint(n,transformed_fixed_point)
```

[14]:
```
# Compare fixed point set with resampled moving point set to see alignment

fig = plt.figure()
ax = plt.axes()

n_points = fixed_set.GetNumberOfPoints()
ax.scatter([fixed_set.GetPoint(i)[0] for i in range(0,n_points)],
           [fixed_set.GetPoint(i)[1] for i in range(0,n_points)])
ax.scatter([transformed_moving_set.GetPoint(i)[0] for i in range(0,n_points)],
           [transformed_moving_set.GetPoint(i)[1] for i in range(0,n_points)])
```

[14]:
```
<matplotlib.collections.PathCollection at 0x1d488213850>
```




### Visualize Gradient Descent

We can use the ITK `ExhaustiveOptimizerv4` class to view how the optimizer moved along the surface defined by the transform parameters and metric.

[15]:
```
# Set up the new optimizer

# Create a new transform and metric for analysis
transform = TransformType.New()
```

(continues on next page)

(continued from previous page)

```
transform.SetIdentity()

metric = PointSetMetricType.New(
        FixedPointSet=fixed_set,
        MovingPointSet=moving_set,
        MovingTransform=transform,
        PointSetSigma=5,
        KernelSigma=10.0,
        UseAnisotropicCovariances=False,
        CovarianceKNeighborhood=5,
        EvaluationKNeighborhood=10,
        Alpha=1.1)
metric.Initialize()

# Create a new observer to map out the parameter surface
optimizer.RemoveAllObservers()
optimizer = ExhaustiveOptimizerType.New(
    Metric=metric)

# Use observers to collect points on the surface
param_space = dict()
def log_exhaustive_iteration():
    param_space[tuple(optimizer.GetCurrentPosition())] = optimizer.GetCurrentValue()

optimizer.AddObserver(itk.IterationEvent(), log_exhaustive_iteration)

# Collect a moderate number of steps along each transform parameter
step_count = 25
optimizer.SetNumberOfSteps([step_count,step_count])

# Step a reasonable distance along each transform parameter
scales = optimizer.GetScales()
scales.SetSize(2)

scale_size = 1.0
scales.SetElement(0, scale_size)
scales.SetElement(1, scale_size)

optimizer.SetScales(scales)
```

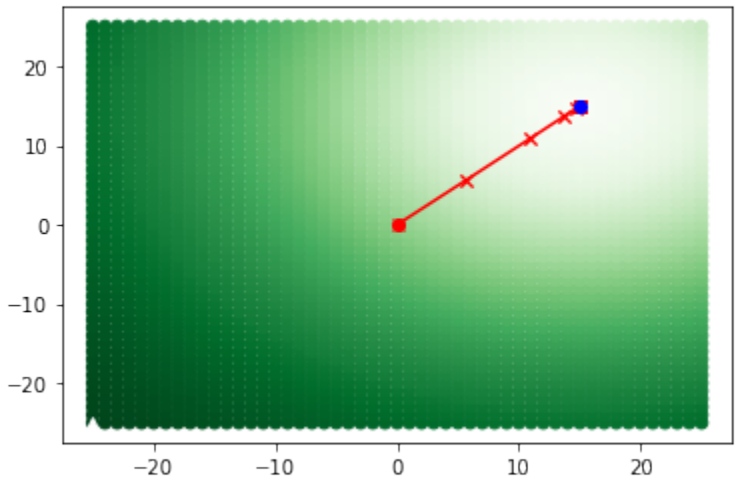
[16]:
```
optimizer.StartOptimization()
print(f'MinimumMetricValue: {optimizer.GetMinimumMetricValue():.4f}\t'
      f'MaximumMetricValue: {optimizer.GetMaximumMetricValue():.4f}\n'
      f'MinimumMetricValuePosition: {list(optimizer.GetMinimumMetricValuePosition())}\
↪t'
      f'MaximumMetricValuePosition: {list(optimizer.GetMaximumMetricValuePosition())}\
↪n'
      f'StopConditionDescription: {optimizer.GetStopConditionDescription()}\t')
```

```
MinimumMetricValue: -0.0648     MaximumMetricValue: -0.0153
MinimumMetricValuePosition: [15.0, 15.0]        MaximumMetricValuePosition: [-25.0, -
↪25.0]
StopConditionDescription: ExhaustiveOptimizerv4: Completed sampling of parametric␣
↪space of size 2
```

[17]:
```
# Reformat gradient descent data to overlay on the plot
descent_x_vals = [iteration_data[i][0] for i in range(0,len(iteration_data))]
descent_y_vals = [iteration_data[i][1] for i in range(0,len(iteration_data))]
```

[18]:
```
# Plot the surface, extrema, and gradient descent data in a matplotlib scatter plot

fig = plt.figure()
ax = plt.axes()

ax.scatter([x for (x,y) in param_space.keys()],
           [y for (x,y) in param_space.keys()],
           c=list(param_space.values()),
           cmap='Greens',zorder=1);
ax.plot(optimizer.GetMinimumMetricValuePosition()[0],
       optimizer.GetMinimumMetricValuePosition()[1],
       'kv')
ax.plot(optimizer.GetMaximumMetricValuePosition()[0],
       optimizer.GetMaximumMetricValuePosition()[1],
       'w^')

for i in range(0,len(iteration_data)):
    ax.plot(descent_x_vals[i:i+2],descent_y_vals[i:i+2],'rx-')
ax.plot(descent_x_vals[0],descent_y_vals[0],'ro')
ax.plot(descent_x_vals[len(iteration_data) - 1],
        descent_y_vals[len(iteration_data) - 1],'bo')
```

[18]:
```
[<matplotlib.lines.Line2D at 0x1d48834c940>]
```




We can also view and export the surface as an image using `itkwidgets`.

[19]:
```
x_vals = list(set(x for (x,y) in param_space.keys()))
y_vals = list(set(y for (x,y) in param_space.keys()))

x_vals.sort()
y_vals.sort(reverse=True)
array = np.array([[param_space[(x,y)]
                   for x in x_vals]
                   for y in y_vals])

image_view = itk.GetImageViewFromArray(array)
```

[20]:
```
view(image_view)
```

```
Viewer(geometries=[], gradient_opacity=0.22, point_sets=[], rendered_image=<itk.
↪itkImagePython.itkImageD2; pro...
```

### Hyperparameter Search

In order to find adequate results with different transforms, metrics, and optimizers it is often useful to compare results across variations in hyperparameters. In the case of this example it was necessary to evaluate performance for different values of the `JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4.PointSetSigma` parameter and `GradientDescentOptimizerv4.DoEstimateLearningRate` parameters. Gradient descent iteration data was plotted along 2D scatter plots to compare and select the hyperparameter combination yielding the best performance.

[21]:
```
# Index values for gradient descent logging

FINAL_OPT_INDEX = 0
DESCENT_DATA_INDEX = 1
```

[22]:
```
hyper_data = dict()
final_optimizers = dict()

# sigma must be sufficiently large to avoid negative entropy results
point_set_sigmas = (1.0,2.5,5.0,10.0,20.0,50.0)

# Compare performance with repeated or one-time learning rate estimation
estimate_rates = [(False,False), (False, True), (True, False)]
```

[23]:
```
# Run gradient descent optimization for each combination of hyperparameters

for trial_values in itertools.product(point_set_sigmas, estimate_rates):
    hyper_data[trial_values] = dict()

    (point_set_sigma, est_rate) = trial_values

    fixed_set, moving_set = \
        make_circles(offset=POINT_SET_OFFSET)

    transform = TransformType.New()
    transform.SetIdentity()

    metric = PointSetMetricType.New(
            FixedPointSet=fixed_set,
            MovingPointSet=moving_set,
            PointSetSigma=point_set_sigma,
            KernelSigma=10.0,
            UseAnisotropicCovariances=False,
            CovarianceKNeighborhood=5,
            EvaluationKNeighborhood=10,
            MovingTransform=transform,
            Alpha=1.1)
    shift_scale_estimator = ShiftScalesType.New(
        Metric=metric,
        VirtualDomainPointSet=metric.GetVirtualTransformedPointSet())
```

(continues on next page)

(continued from previous page)

```
    metric.Initialize()
    optimizer = OptimizerType.New(
        Metric=metric,
        NumberOfIterations=100,
        MaximumStepSizeInPhysicalUnits=3.0,
        MinimumConvergenceValue=-1,
        DoEstimateLearningRateOnce=est_rate[0],
        DoEstimateLearningRateAtEachIteration=est_rate[1],
        LearningRate=1e6,  # Ignored if either est_rate argument is True
        ReturnBestParametersAndValue=False)

    optimizer.SetScalesEstimator(shift_scale_estimator)

    def log_hyper_iteration_data():
        hyper_data[trial_values][optimizer.GetCurrentIteration()] =\
                 round(optimizer.GetCurrentMetricValue(),8)

    optimizer.AddObserver(itk.IterationEvent(), log_hyper_iteration_data)

    optimizer.StartOptimization()

    final_optimizers[trial_values] = optimizer
```

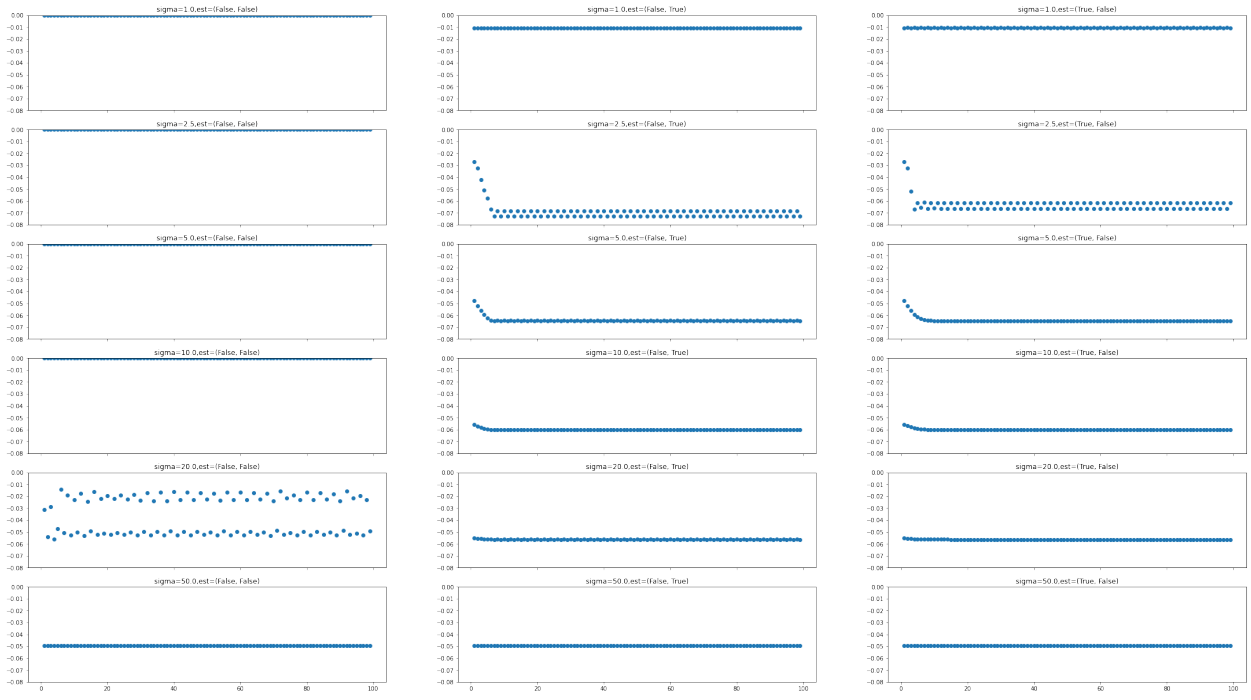
[24]:
```
# Print results for each set of hyperparameters

print('PS_sigma\test once/each:\tfinal index\tfinal metric')
for trial_values in itertools.product(point_set_sigmas,estimate_rates):

    print(f'{trial_values[0]}\t\t{trial_values[1]}:\t\t'
          f'{final_optimizers[trial_values].GetCurrentIteration()}\t'
          f'{final_optimizers[trial_values].GetCurrentMetricValue():10.8f}')
```

```
PS_sigma       est once/each:  final index     final metric
1.0             (False, False):         100     0.00000000
1.0             (False, True):          100     -0.01074471
1.0             (True, False):          100     -0.01061929
2.5             (False, False):         100     0.00000000
2.5             (False, True):          100     -0.07264293
2.5             (True, False):          100     -0.06137661
5.0             (False, False):         100     0.00000000
5.0             (False, True):          100     -0.06481184
5.0             (True, False):          100     -0.06481531
10.0            (False, False):         100     0.00000000
10.0            (False, True):          100     -0.06039311
10.0            (True, False):          100     -0.06039382
20.0            (False, False):         100     -0.04939961
20.0            (False, True):          100     -0.05628143
20.0            (True, False):          100     -0.05628171
50.0            (False, False):         100     -0.04947632
50.0            (False, True):          100     -0.04947578
50.0            (True, False):          100     -0.04947632
```

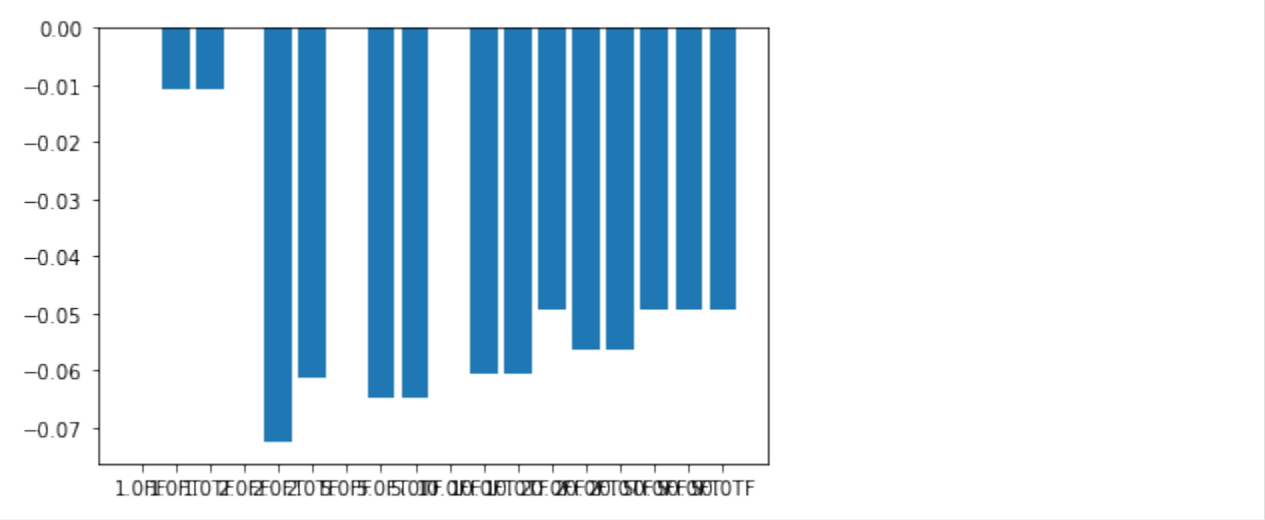
We can use `matplotlib` subplots and bar graphs to compare gradient descent performance and final metric values for each set of hyperparameters. In this example we see that estimating the learning rate once typically gives the best performance over time, while estimating the learning rate at each iteration can prevent gradient descent from converging efficiently. The hyperparameter set giving the best and most consistent metric value is that with `PointSetSigma=5.0` and `DoEstimateLearningRateOnce=True`, which are the values we have used in

this notebook.

[25]:
```
# Visualize metric over gradient descent iterations as matplotlib subplots.

f, axn = plt.subplots(len(point_set_sigmas),len(estimate_rates),sharex=True)

for (i,j) in [(i,j)
              for i in range(0,len(point_set_sigmas))
              for j in range(0,len(estimate_rates))]:
    axn[i,j].scatter(x=list(hyper_data[point_set_sigmas[i],estimate_rates[j]].
↪keys())[1:],
                     y=list(hyper_data[point_set_sigmas[i],estimate_rates[j]].
↪values())[1:])
    axn[i,j].set_title(f'sigma={point_set_sigmas[i]},est={estimate_rates[j]}')
    axn[i,j].set_ylim(-0.08,0)

plt.subplots_adjust(top=5,bottom=1,right=5)
plt.show()
```




[26]:
```
# Compare final metric magnitudes

fig = plt.figure()
ax = fig.gca()

labels = [f'{round(sig,0)}{"T" if est0 else "F"}{"T" if est1 else "F"}'
          for (sig,(est0,est1)) in itertools.product(point_set_sigmas,estimate_rates)]

vals = [final_optimizers[trial_values].GetCurrentMetricValue()
        for trial_values in itertools.product(point_set_sigmas,estimate_rates)]

ax.bar(labels,vals)
plt.show()
```




## Synopsis

Register two point sets with the Jensen-Havrda-Charvat-Tsallis point set metric.

## Results

```
It: 0    metric value: -0.0641355
It: 1    metric value: -0.0955508
It: 2    metric value: -0.0956365
It: 3    metric value: -0.0956648
It: 4    metric value: -0.0956742
It: 5    metric value: -0.0956774
It: 6    metric value: -0.0956784
It: 7    metric value: -0.0956788
It: 8    metric value: -0.0956789
It: 9    metric value: -0.0956789
numberOfIterations: 10
Moving-source final value: -0.0956789
Moving-source final position: [0.9999999481711049, 3.6956646020565954e-9,
                               -3.9958144654776e-8, 1.0000000021678688,
                               1.9991709056354183, 1.999169080146488]
Optimizer scales: [10000, 9998.46502472856,
                   10000, 9998.46502472848,
                   1.0000000000010232, 1.0000000000010232]
Optimizer learning rate: 164.938
```

### Jupyter Notebook

launch binder

### Code

### Python

```python
#!/usr/bin/env python3
import sys
from math import pi, sin, cos

import itk

# Generate two circles with a small offset
def make_circles(dimension: int = 2):

    PointSetType = itk.PointSet[itk.F, dimension]

    RADIUS = 100
    offset = [2.0] * dimension

    fixed_points = PointSetType.New()
    moving_points = PointSetType.New()
    fixed_points.Initialize()
    moving_points.Initialize()

    count = 0
    step = 0.1
    for count in range(0, int(2 * pi / step) + 1):

        theta = count * step

        fixed_point = list()
        fixed_point.append(RADIUS * cos(theta))
        for dim in range(1, dimension):
            fixed_point.append(RADIUS * sin(theta))
        fixed_points.SetPoint(count, fixed_point)

        moving_point = [fixed_point[dim] + offset[dim] for dim in range(0, dimension)]
        moving_points.SetPoint(count, moving_point)

    return fixed_points, moving_points


def test_registration(dimension: int = 2):
    # Define test parameters

    num_iterations = 10

    passed = True
    tolerance = 0.05

    # Define types
```

(continues on next page)

(continued from previous page)

```
    PointType = itk.Point[itk.F, dimension]
    PointSetType = itk.PointSet[itk.F, dimension]
    AffineTransformType = itk.AffineTransform[itk.D, dimension]
    PointSetMetricType = itk.JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4[
        PointSetType
    ]
    ShiftScalesType = itk.RegistrationParameterScalesFromPhysicalShift[
        PointSetMetricType
    ]
    OptimizerType = itk.RegularStepGradientDescentOptimizerv4[itk.D]

    # Make point sets
    fixed_set, moving_set = make_circles(dimension)

    transform = AffineTransformType.New()
    transform.SetIdentity()

    metric = PointSetMetricType.New(
        FixedPointSet=fixed_set,
        MovingPointSet=moving_set,
        PointSetSigma=1.0,
        KernelSigma=10.0,
        UseAnisotropicCovariances=False,
        CovarianceKNeighborhood=5,
        EvaluationKNeighborhood=10,
        MovingTransform=transform,
        Alpha=1.1,
    )
    metric.Initialize()

    shift_scale_estimator = ShiftScalesType.New(
        Metric=metric, VirtualDomainPointSet=metric.GetVirtualTransformedPointSet()
    )

    optimizer = OptimizerType.New(
        Metric=metric,
        NumberOfIterations=num_iterations,
        ScalesEstimator=shift_scale_estimator,
        MaximumStepSizeInPhysicalUnits=3.0,
        MinimumConvergenceValue=0.0,
        ConvergenceWindowSize=10,
    )

    def print_iteration():
        print(
            f"It: {optimizer.GetCurrentIteration()}"
            f" metric value: {optimizer.GetCurrentMetricValue():.6f} "
        )

    optimizer.AddObserver(itk.IterationEvent(), print_iteration)

    # Run optimization to align the point sets
    optimizer.StartOptimization()

    print(f"Number of iterations: {num_iterations}")
    print(f"Moving-source final value: {optimizer.GetCurrentMetricValue()}")
    print(f"Moving-source final position: {list(optimizer.GetCurrentPosition())}")
```

(continues on next page)

(continued from previous page)

```
    print(f"Optimizer scales: {list(optimizer.GetScales())}")
    print(f"Optimizer learning rate: {optimizer.GetLearningRate()}")

    # applying the resultant transform to moving points and verify result
    print("Fixed\tMoving\tMovingTransformed\tFixedTransformed\tDiff")

    moving_inverse = metric.GetMovingTransform().GetInverseTransform()
    fixed_inverse = metric.GetFixedTransform().GetInverseTransform()

    def print_point(vals: list) -> str:
        return f'[{",".join(f"{x:.4f}" for x in vals)}]'

    for n in range(0, metric.GetNumberOfComponents()):
        transformed_moving_point = moving_inverse.TransformPoint(moving_set.
↪GetPoint(n))
        transformed_fixed_point = fixed_inverse.TransformPoint(fixed_set.GetPoint(n))

        difference = [
            transformed_moving_point[dim] - transformed_fixed_point[dim]
            for dim in range(0, dimension)
        ]

        print(
            f"{print_point(fixed_set.GetPoint(n))}"
            f"\t{print_point(moving_set.GetPoint(n))}"
            f"\t{print_point(transformed_moving_point)}"
            f"\t{print_point(transformed_fixed_point)}"
            f"\t{print_point(difference)}"
        )

        if any(abs(difference[dim]) > tolerance for dim in range(0, dimension)):
            passed = False

    if not passed:
        raise Exception("Transform outside of allowable tolerance")
    else:
        print("Transform is within allowable tolerance.")


if __name__ == "__main__":
    if len(sys.argv) == 2:
        dimension = int(sys.argv[1])
        test_registration(dimension)
    else:
        test_registration()
```

### C++

```
#include "itkJensenHavrdaCharvatTsallisPointSetToPointSetMetricv4.h"
#include "itkGradientDescentOptimizerv4.h"
#include "itkTransform.h"
#include "itkAffineTransform.h"
#include "itkRegistrationParameterScalesFromPhysicalShift.h"
#include "itkCommand.h"

#include <fstream>

template <typename TFilter>
class␣
↪itkJensenHavrdaCharvatTsallisPointSetMetricRegistrationTestCommandIterationUpdate :␣
↪public itk::Command
{
public:
  using Self =␣
↪itkJensenHavrdaCharvatTsallisPointSetMetricRegistrationTestCommandIterationUpdate;

  using Superclass = itk::Command;
  using Pointer = itk::SmartPointer<Self>;
  itkNewMacro(Self);

protected:
  itkJensenHavrdaCharvatTsallisPointSetMetricRegistrationTestCommandIterationUpdate()␣
↪= default;

public:
  void
  Execute(itk::Object * caller, const itk::EventObject & event) override
  {
    Execute((const itk::Object *)caller, event);
  }

  void
  Execute(const itk::Object * object, const itk::EventObject & event) override
  {
    if (typeid(event) != typeid(itk::IterationEvent))
    {
      return;
    }
    const auto * optimizer = dynamic_cast<const TFilter *>(object);

    if (!optimizer)
    {
      itkGenericExceptionMacro("Error dynamic_cast failed");
    }
    std::cout << "It: " << optimizer->GetCurrentIteration() << " metric value: " <<␣
↪optimizer->GetCurrentMetricValue();
    std::cout << std::endl;
  }
};


int
main(int argc, char * argv[])
```

(continues on next page)

(continued from previous page)

```
{
  constexpr unsigned int Dimension = 2;

  unsigned int numberOfIterations = 10;
  if (argc > 1)
  {
    numberOfIterations = std::stoi(argv[1]);
  }

  using PointSetType = itk::PointSet<unsigned char, Dimension>;

  using PointType = PointSetType::PointType;

  PointSetType::Pointer fixedPoints = PointSetType::New();
  fixedPoints->Initialize();

  PointSetType::Pointer movingPoints = PointSetType::New();
  movingPoints->Initialize();


  // two ellipses, one rotated slightly
  /*
    // Having trouble with these, as soon as there's a slight rotation added.
    unsigned long count = 0;
    for( float theta = 0; theta < 2.0 * itk::Math::pi; theta += 0.1 )
      {
      float radius = 100.0;
      PointType fixedPoint;
      fixedPoint[0] = 2 * radius * std::cos( theta );
      fixedPoint[1] = radius * std::sin( theta );
      fixedPoints->SetPoint( count, fixedPoint );

      PointType movingPoint;
      movingPoint[0] = 2 * radius * std::cos( theta + (0.02 * itk::Math::pi) ) + 2.0;
      movingPoint[1] = radius * std::sin( theta + (0.02 * itk::Math::pi) ) + 2.0;
      movingPoints->SetPoint( count, movingPoint );

      count++;
      }
  */

  // two circles with a small offset
  PointType offset;
  for (unsigned int d = 0; d < Dimension; d++)
  {
    offset[d] = 2.0;
  }
  unsigned long count = 0;
  for (float theta = 0; theta < 2.0 * itk::Math::pi; theta += 0.1)
  {
    PointType fixedPoint;
    float     radius = 100.0;
    fixedPoint[0] = radius * std::cos(theta);
    fixedPoint[1] = radius * std::sin(theta);
    if (Dimension > 2)
    {
      fixedPoint[2] = radius * std::sin(theta);
```

(continues on next page)

(continued from previous page)

```
    }
    fixedPoints->SetPoint(count, fixedPoint);

    PointType movingPoint;
    movingPoint[0] = fixedPoint[0] + offset[0];
    movingPoint[1] = fixedPoint[1] + offset[1];
    if (Dimension > 2)
    {
      movingPoint[2] = fixedPoint[2] + offset[2];
    }
    movingPoints->SetPoint(count, movingPoint);

    count++;
  }

  using AffineTransformType = itk::AffineTransform<double, Dimension>;
  AffineTransformType::Pointer transform = AffineTransformType::New();
  transform->SetIdentity();

  // Instantiate the metric
  using PointSetMetricType = itk::JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4
↪<PointSetType>;
  PointSetMetricType::Pointer metric = PointSetMetricType::New();
  metric->SetFixedPointSet(fixedPoints);
  metric->SetMovingPointSet(movingPoints);
  metric->SetPointSetSigma(1.0);
  metric->SetKernelSigma(10.0);
  metric->SetUseAnisotropicCovariances(false);
  metric->SetCovarianceKNeighborhood(5);
  metric->SetEvaluationKNeighborhood(10);
  metric->SetMovingTransform(transform);
  metric->SetAlpha(1.1);
  metric->Initialize();

  // scales estimator
  using RegistrationParameterScalesFromShiftType =
    itk::RegistrationParameterScalesFromPhysicalShift<PointSetMetricType>;
  RegistrationParameterScalesFromShiftType::Pointer shiftScaleEstimator =
    RegistrationParameterScalesFromShiftType::New();
  shiftScaleEstimator->SetMetric(metric);
  // needed with pointset metrics
  shiftScaleEstimator->SetVirtualDomainPointSet(metric->
↪GetVirtualTransformedPointSet());

  // optimizer
  using OptimizerType = itk::GradientDescentOptimizerv4;
  OptimizerType::Pointer optimizer = OptimizerType::New();
  optimizer->SetMetric(metric);
  optimizer->SetNumberOfIterations(numberOfIterations);
  optimizer->SetScalesEstimator(shiftScaleEstimator);
  optimizer->SetMaximumStepSizeInPhysicalUnits(3.0);

  using CommandType =␣
↪itkJensenHavrdaCharvatTsallisPointSetMetricRegistrationTestCommandIterationUpdate
↪<OptimizerType>;
  CommandType::Pointer observer = CommandType::New();
  optimizer->AddObserver(itk::IterationEvent(), observer);
```

(continues on next page)

(continued from previous page)

```
  optimizer->SetMinimumConvergenceValue(0.0);
  optimizer->SetConvergenceWindowSize(10);
  optimizer->StartOptimization();

  std::cout << "numberOfIterations: " << numberOfIterations << std::endl;
  std::cout << "Moving-source final value: " << optimizer->GetCurrentMetricValue() <<␣
↪std::endl;
  std::cout << "Moving-source final position: " << optimizer->GetCurrentPosition() <<␣
↪std::endl;
  std::cout << "Optimizer scales: " << optimizer->GetScales() << std::endl;
  std::cout << "Optimizer learning rate: " << optimizer->GetLearningRate() << std::
↪endl;

  // applying the resultant transform to moving points and verify result
  std::cout << "Fixed\tMoving\tMovingTransformed\tFixedTransformed\tDiff" << std::
↪endl;
  bool                                          passed = true;
  PointType::ValueType                          tolerance = 1e-2;
  AffineTransformType::InverseTransformBasePointer movingInverse = metric->
↪GetMovingTransform()->GetInverseTransform();
  AffineTransformType::InverseTransformBasePointer fixedInverse = metric->
↪GetFixedTransform()->GetInverseTransform();
  for (unsigned int n = 0; n < metric->GetNumberOfComponents(); n++)
  {
    // compare the points in virtual domain
    PointType transformedMovingPoint = movingInverse->TransformPoint(movingPoints->
↪GetPoint(n));
    PointType transformedFixedPoint = fixedInverse->TransformPoint(fixedPoints->
↪GetPoint(n));
    PointType difference;
    difference[0] = transformedMovingPoint[0] - transformedFixedPoint[0];
    difference[1] = transformedMovingPoint[1] - transformedFixedPoint[1];
    std::cout << fixedPoints->GetPoint(n) << "\t" << movingPoints->GetPoint(n) << "\t
↪" << transformedMovingPoint << "\t"
              << transformedFixedPoint << "\t" << difference << std::endl;
    if (fabs(difference[0]) > tolerance || fabs(difference[1]) > tolerance)
    {
      passed = false;
    }
  }
  if (!passed)
  {
    std::cerr << "Results do not match truth within tolerance." << std::endl;
    return EXIT_FAILURE;
  }


  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TPointSet**, class **TInternalComputationValueType** = double>
**class JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4** : **public** itk::PointSetToPointSetMetricv4<*TP*
Implementation of the Jensen Havrda Charvat Tsallis Point Set metric.

Given a specified transform and direction, this class calculates the value and derivative between a "fixed" and "moving" point set pair using the Havrda-Charvat-Tsallis entropy family, a generalization of the well-known Shannon entropy, and the Jensen divergence. Another way to look at the family of information-theoretic measures is that the points are used to construct the corresponding probably density functions.

In addition, we allow the user to invoke a manifold parzen windowing of the data. Instead of an isotropic Gaussian being associated with each point, we can actually calculate the covariance matrix for each point such that it reflects the locate point set structure.

To speed up the metric calculation, we use ITK's K-d tree to query the metric value only for a given neighborhood. Considering that probably only a small subset of points is needed to get a good approximation of the metric value for a single point, this is probably warranted. So what we do is transform each point (with the specified transform) and construct the k-d tree from the transformed points.

Contributed by Nicholas J. Tustison, James C. Gee in the Insight Journal paper: https://www.insight-journal.org/browse/publication/317

N.J. Tustison, S. P. Awate, G. Song, T. S. Cook, and J. C. Gee. "Point set registration using Havrda-Charvat-Tsallis entropy measures" IEEE Transactions on Medical Imaging, 30(2):451-60, 2011.

**Note** The original work reported in Tustison et al. 2011 optionally employed a regularization term to prevent the moving point set(s) from coalescing to a single point location. However, within the registration framework, this term is of limited utility as such regularization is dictated by the transform and any explicit regularization terms. Also note that the published work applies to multiple points sets each of which could be considered "moving" but this is also not applicable for this particular implementation.

**REFERENCE**

See itk::JensenHavrdaCharvatTsallisPointSetToPointSetMetricv4 for additional documentation.

template<typename **TInternalComputationValueType**>
**class GradientDescentOptimizerv4Template** : **public** itk::GradientDescentOptimizerBasev4Template<*TInternalComp*
Gradient descent optimizer.

GradientDescentOptimizer implements a simple gradient descent optimizer. At each iteration the current position is updated according to

$$p_{n+1} = p_n + \text{learningRate}\,\frac{\partial f(p_n)}{\partial p_n}$$

Optionally, the best metric value and matching parameters can be stored and retried via GetValue() and GetCurrentPosition(). See SetReturnBestParametersAndValue().

Gradient scales can be manually set or automatically estimated, as documented in the base class. The learing rate defaults to 1.0, and can be set in two ways: 1) manually, via `SetLearningRate()`. Or, 2) automatically, either at each iteration or only at the first iteration, by assigning a ScalesEstimator via SetScalesEstimator(). When a ScalesEstimator is assigned, the optimizer is enabled by default to estimate learning rate only once, during the first iteration. This behavior can be changed via SetDoEstimateLearningRateAtEveryIteration() and SetDoEstimateLearningRateOnce(). For learning rate to be estimated at each iteration, the user must call SetDoEstimateLearningRateAtEveryIteration(true) and SetDoEstimateLearningRateOnce(false). When enabled, the optimizer computes learning rate(s) such that at each step, each voxel's change in physical space will be less than m_MaximumStepSizeInPhysicalUnits.

```
m_LearningRate =
  m_MaximumStepSizeInPhysicalUnits /
  m_ScalesEstimator->EstimateStepScale(scaledGradient)
```

where m_MaximumStepSizeInPhysicalUnits defaults to the voxel spacing returned by m_ScalesEstimator->EstimateMaximumStepSize() (which is typically 1 voxel), and can be set by the user via SetMaximumStepSizeInPhysicalUnits(). When SetDoEstimateLearningRateOnce is enabled, the voxel change may become being greater than m_MaximumStepSizeInPhysicalUnits in later iterations.

**Note** Unlike the previous version of GradientDescentOptimizer, this version does not have a "maximize/minimize" option to modify the effect of the metric derivative. The assigned metric is assumed to return a parameter derivative result that "improves" the optimization when *added* to the current parameters via the metric::UpdateTransformParameters method, after the optimizer applies scales and a learning rate.

Subclassed by itk::GradientDescentLineSearchOptimizerv4Template< TInternalComputationValueType >, itk::MultiGradientOptimizerv4Template< TInternalComputationValueType >, itk::QuasiNewtonOptimizerv4Template< TInternalComputationValueType >, itk::RegularStepGradientDescentOptimizerv4< TInternalComputationValueType >

See itk::GradientDescentOptimizerv4Template for additional documentation.

## 3.10 Segmentation

### 3.10.1 Classifiers

#### Cluster Pixels in Grayscale Image

#### Synopsis

Cluster the pixels in a greyscale image.

#### Results

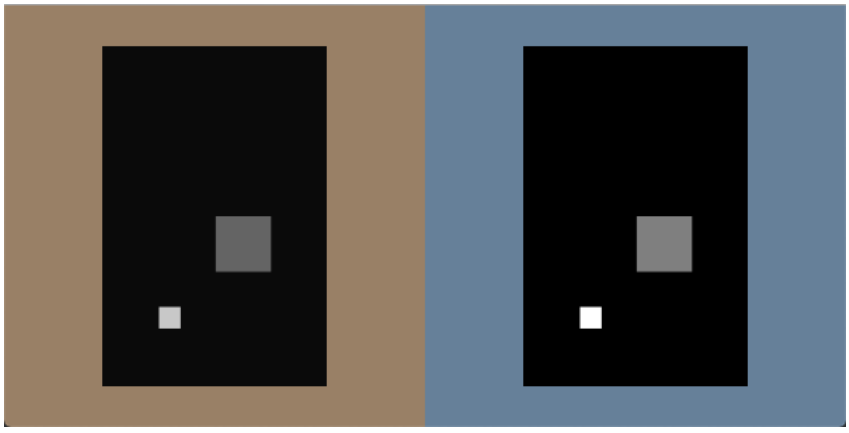


Fig. 372: Output In VTK Window

Output:

```
cluster[0]     estimated mean : 10
cluster[1]     estimated mean : 100
cluster[2]     estimated mean : 200
Number of pixels per class
```

## Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkScalarImageKmeansImageFilter.h"
#include "itkRelabelComponentImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkImageRegionIterator.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkVersion.h"
#include "vtkImageViewer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkImageMapper3D.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using ImageType = itk::Image<unsigned char, 2>;

static void
CreateImage(ImageType::Pointer image);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using KMeansFilterType = itk::ScalarImageKmeansImageFilter<ImageType>;

  KMeansFilterType::Pointer kmeansFilter = KMeansFilterType::New();

  kmeansFilter->SetInput(image);
  kmeansFilter->SetUseNonContiguousLabels(true);
  kmeansFilter->AddClassWithInitialMean(8);
  kmeansFilter->AddClassWithInitialMean(110);
  kmeansFilter->AddClassWithInitialMean(210);
  kmeansFilter->Update();

  KMeansFilterType::ParametersType estimatedMeans = kmeansFilter->GetFinalMeans();

  const unsigned int numberOfClasses = estimatedMeans.Size();
```

(continues on next page)

(continued from previous page)

```
  for (unsigned int i = 0; i < numberOfClasses; ++i)
  {
    std::cout << "cluster[" << i << "] ";
    std::cout << "    estimated mean : " << estimatedMeans[i] << std::endl;
  }

  using OutputImageType = KMeansFilterType::OutputImageType;

  using RelabelFilterType = itk::RelabelComponentImageFilter<OutputImageType,␣
↪OutputImageType>;

  RelabelFilterType::Pointer relabeler = RelabelFilterType::New();

  relabeler->SetInput(kmeansFilter->GetOutput());

  using RescaleFilterType = itk::RescaleIntensityImageFilter<ImageType, ImageType>;
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetInput(relabeler->GetOutput());
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(255);

  using SizesType = RelabelFilterType::ObjectSizeInPixelsContainerType;

  const SizesType & sizes = relabeler->GetSizeOfObjectsInPixels();

  auto sizeItr = sizes.begin();
  auto sizeEnd = sizes.end();

  std::cout << "Number of pixels per class " << std::endl;
  unsigned int kclass = 0;
  while (sizeItr != sizeEnd)
  {
    std::cout << "Class " << kclass << " = " << *sizeItr << std::endl;
    ++kclass;
    ++sizeItr;
  }

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer originalConnector = ConnectorType::New();
  originalConnector->SetInput(image);
  vtkSmartPointer<vtkImageActor> originalActor = vtkSmartPointer<vtkImageActor>::
↪New();
#if VTK_MAJOR_VERSION <= 5
  originalActor->SetInput(originalConnector->GetOutput());
#else
  originalConnector->Update();
  originalActor->SetInputData(originalConnector->GetOutput());
#endif

  ConnectorType::Pointer outputConnector = ConnectorType::New();
  outputConnector->SetInput(rescaleFilter->GetOutput());

  vtkSmartPointer<vtkImageActor> outputActor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  outputActor->SetInput(outputConnector->GetOutput());
#else
```

(continues on next page)

(continued from previous page)

```
  outputConnector->Update();
  outputActor->SetInputData(outputConnector->GetOutput());
#endif

  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(600, 300);

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.5, 1.0 };
  double rightViewport[4] = { 0.5, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);
  leftRenderer->SetBackground(.6, .5, .4);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(.4, .5, .6);


  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(originalActor);
  rightRenderer->AddActor(outputActor);

  leftRenderer->ResetCamera();
  rightRenderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();

  interactor->SetInteractorStyle(style);

  interactor->Start();

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;
```

(continues on next page)

(continued from previous page)

```
  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 150 &&␣
↪imageIterator.GetIndex()[1] > 100 &&
        imageIterator.GetIndex()[1] < 150)
    {
      imageIterator.Set(100);
    }
    else if (imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70 &&␣
↪imageIterator.GetIndex()[1] > 50 &&
             imageIterator.GetIndex()[1] < 70)
    {
      imageIterator.Set(200);
    }
    else
    {
      imageIterator.Set(10);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<unsigned char, *TInputImage*::ImageDimension>>
**class ScalarImageKmeansImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Classifies the intensity values of a scalar image using the K-Means algorithm.

Given an input image with scalar values, it uses the K-Means statistical classifier in order to define labels for every pixel in the image. The filter is templated over the type of the input image. The output image is predefined as having the same dimension of the input image and pixel type unsigned char, under the assumption that the classifier will generate less than 256 classes.

You may want to look also at the RelabelImageFilter that may be used as a postprocessing stage, in particular if you are interested in ordering the labels by their relative size in number of pixels.

**See** Image

**See** ImageKmeansModelEstimator

**See** KdTreeBasedKmeansEstimator, WeightedCentroidKdTreeGenerator, KdTree

**See** RelabelImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cluster Pixels In Grayscale Image
- K-Means Clustering

See itk::ScalarImageKmeansImageFilter for additional documentation.

### K-Means Clustering

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

KMeans clustering.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include <itkImage.h>
#include <itkImageFileReader.h>
#include <itkImageFileWriter.h>
#include <itkScalarImageKmeansImageFilter.h>

int
main(int argc, char * argv[])
{
  // sample usage
  //./kMeansClustering input.jpg output.jpg 1 3 0 100 200

  // verify command line arguments
  if (argc < 5)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " inputScalarImage outputLabeledImage contiguousLabels";
    std::cerr << " numberOfClasses mean1 mean2... meanN " << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  // parse command line arguments
  const char *       inputImageFileName = argv[1];
  const char *       outputImageFileName = argv[2];
  const unsigned int useNonContiguousLabels = std::stoi(argv[3]);
  const unsigned int numberOfInitialClasses = std::stoi(argv[4]);

  constexpr unsigned int argoffset = 5;

  if (static_cast<unsigned int>(argc) < numberOfInitialClasses + argoffset)
  {
    std::cerr << "Error: " << std::endl;
    std::cerr << numberOfInitialClasses << " classes has been specified ";
    std::cerr << "but no enough means have been provided in the command ";
    std::cerr << "line arguments " << std::endl;
    return EXIT_FAILURE;
  }

  std::vector<double> userMeans;
  for (unsigned k = 0; k < numberOfInitialClasses; k++)
  {
    const double userProvidedInitialMean = std::stod(argv[k + argoffset]);
    userMeans.push_back(userProvidedInitialMean);
  }

  // Define the pixel type and dimension of the image that we intend to
  // classify.

  using PixelType = signed short;
  constexpr unsigned int Dimension = 2;

  using ImageType = itk::Image<PixelType, Dimension>;

  // create a reader
  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputImageFileName);

  // Instantiate the ScalarImageKmeansImageFilter
  using KMeansFilterType = itk::ScalarImageKmeansImageFilter<ImageType>;

  KMeansFilterType::Pointer kmeansFilter = KMeansFilterType::New();

  kmeansFilter->SetInput(reader->GetOutput());

  // Make the output image intellegable by expanding the range of output image values,
↪ if desired

  kmeansFilter->SetUseNonContiguousLabels(useNonContiguousLabels);

  // initialize using the user input means

  for (unsigned k = 0; k < numberOfInitialClasses; k++)
  {
    kmeansFilter->AddClassWithInitialMean(userMeans[k]);
  }
```

(continues on next page)

(continued from previous page)

```
  // Create and setup a writer

  using OutputImageType = KMeansFilterType::OutputImageType;

  using WriterType = itk::ImageFileWriter<OutputImageType>;

  WriterType::Pointer writer = WriterType::New();

  writer->SetInput(kmeansFilter->GetOutput());

  writer->SetFileName(outputImageFileName);

  // execut the pipeline
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & excp)
  {
    std::cerr << "Problem encountered while writing ";
    std::cerr << " image file : " << outputImageFileName << std::endl;
    std::cerr << excp << std::endl;
    return EXIT_FAILURE;
  }

  // inspect the means
  KMeansFilterType::ParametersType estimatedMeans = kmeansFilter->GetFinalMeans();

  const unsigned int numberOfClasses = estimatedMeans.Size();

  for (unsigned int i = 0; i < numberOfClasses; ++i)
  {
    std::cout << "cluster[" << i << "] ";
    std::cout << "    estimated mean : " << estimatedMeans[i] << std::endl;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage** = *Image*<unsigned char, *TInputImage*::ImageDimension>>
**class ScalarImageKmeansImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Classifies the intensity values of a scalar image using the K-Means algorithm.

Given an input image with scalar values, it uses the K-Means statistical classifier in order to define labels for every pixel in the image. The filter is templated over the type of the input image. The output image is predefined as having the same dimension of the input image and pixel type unsigned char, under the assumption that the classifier will generate less than 256 classes.

You may want to look also at the RelabelImageFilter that may be used as a postprocessing stage, in particular if you are interested in ordering the labels by their relative size in number of pixels.

**See** Image

**See** ImageKmeansModelEstimator

**See** KdTreeBasedKmeansEstimator, WeightedCentroidKdTreeGenerator, KdTree

**See** RelabelImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Cluster Pixels In Grayscale Image
- K-Means Clustering

See itk::ScalarImageKmeansImageFilter for additional documentation.

### K Means Cluster of Pixels in Image

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

### Synopsis

Compute kmeans clusters of pixels in an image

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

### C++

```
#include "itkImage.h"
#include "itkListSample.h"
#include "itkVector.h"
#include "itkImageKmeansModelEstimator.h"
#include "itkImageRegionIteratorWithIndex.h"
#include "itkImageToListSampleAdaptor.h"
#include "itkDistanceToCentroidMembershipFunction.h"
#include "itkSampleClassifierFilter.h"
#include "itkMinimumDecisionRule.h"
#include "itkImageFileWriter.h"

using MeasurementVectorType = itk::Vector<unsigned char, 3>;
using ColorImageType = itk::Image<MeasurementVectorType, 2>;
using ScalarImageType = itk::Image<unsigned char, 2>;

static void
```

(continues on next page)

(continued from previous page)

```
CreateImage(ColorImageType::Pointer image);

int
main(int, char *[])
{
  // Create a demo image
  ColorImageType::Pointer image = ColorImageType::New();
  CreateImage(image);

  // Compute pixel clusters using KMeans
  using MembershipFunctionType = itk::Statistics::DistanceToCentroidMembershipFunction
↪<MeasurementVectorType>;
  using MembershipFunctionPointer = MembershipFunctionType::Pointer;
  using MembershipFunctionPointerVector = std::vector<MembershipFunctionPointer>;

  using ImageKmeansModelEstimatorType = itk::ImageKmeansModelEstimator<ColorImageType,
↪ MembershipFunctionType>;

  ImageKmeansModelEstimatorType::Pointer kmeansEstimator =␣
↪ImageKmeansModelEstimatorType::New();
  kmeansEstimator->SetInputImage(image);
  kmeansEstimator->SetNumberOfModels(3);
  kmeansEstimator->SetThreshold(0.01);
  kmeansEstimator->SetOffsetAdd(0.01);
  kmeansEstimator->SetOffsetMultiply(0.01);
  kmeansEstimator->SetMaxSplitAttempts(10);
  kmeansEstimator->Update();

  // Classify each pixel
  using SampleType = itk::Statistics::ListSample<MeasurementVectorType>;
  using ClassifierType = itk::Statistics::SampleClassifierFilter<SampleType>;
  ClassifierType::Pointer classifier = ClassifierType::New();

  using DecisionRuleType = itk::Statistics::MinimumDecisionRule;
  DecisionRuleType::Pointer decisionRule = DecisionRuleType::New();

  classifier->SetDecisionRule(decisionRule);
  classifier->SetNumberOfClasses(3);

  using ClassLabelVectorObjectType = ClassifierType::ClassLabelVectorObjectType;
  using ClassLabelVectorType = ClassifierType::ClassLabelVectorType;
  using MembershipFunctionVectorObjectType = ClassifierType::
↪MembershipFunctionVectorObjectType;
  using MembershipFunctionVectorType = ClassifierType::MembershipFunctionVectorType;

  // Setup membership functions
  MembershipFunctionPointerVector kmeansMembershipFunctions = kmeansEstimator->
↪GetMembershipFunctions();

  MembershipFunctionVectorObjectType::Pointer membershipFunctionsVectorObject =
    MembershipFunctionVectorObjectType::New();
  classifier->SetMembershipFunctions(membershipFunctionsVectorObject);

  MembershipFunctionVectorType & membershipFunctionsVector =␣
↪membershipFunctionsVectorObject->Get();

  for (auto & kmeansMembershipFunction : kmeansMembershipFunctions)
```

(continues on next page)

(continued from previous page)

```
  {
    membershipFunctionsVector.push_back(kmeansMembershipFunction.GetPointer());
  }

  // Setup class labels
  ClassLabelVectorObjectType::Pointer classLabelsObject = ClassLabelVectorObjectType::
↪New();
  classifier->SetClassLabels(classLabelsObject);

  ClassLabelVectorType & classLabelsVector = classLabelsObject->Get();
  classLabelsVector.push_back(50);
  classLabelsVector.push_back(150);
  classLabelsVector.push_back(250);

  // Perform the classification
  using SampleAdaptorType = itk::Statistics::ImageToListSampleAdaptor<ColorImageType>;
  SampleAdaptorType::Pointer sample = SampleAdaptorType::New();
  sample->SetImage(image);

  classifier->SetInput(sample);
  classifier->Update();

  // Prepare the output image
  ScalarImageType::Pointer outputImage = ScalarImageType::New();
  outputImage->SetRegions(image->GetLargestPossibleRegion());
  outputImage->Allocate();
  outputImage->FillBuffer(0);

  // Setup the membership iterator
  const ClassifierType::MembershipSampleType *        membershipSample = classifier->
↪GetOutput();
  ClassifierType::MembershipSampleType::ConstIterator membershipIterator =␣
↪membershipSample->Begin();

  // Setup the output image iterator - this is automatically synchronized with the␣
↪membership iterator since the sample
  // is an adaptor
  itk::ImageRegionIteratorWithIndex<ScalarImageType> outputIterator(outputImage,
                                                                    outputImage->
↪GetLargestPossibleRegion());
  outputIterator.GoToBegin();

  while (membershipIterator != membershipSample->End())
  {
    int classLabel = membershipIterator.GetClassLabel();
    // std::cout << "Class label: " << classLabel << std::endl;
    outputIterator.Set(classLabel);
    ++membershipIterator;
    ++outputIterator;
  }

  using WriterType = itk::ImageFileWriter<ColorImageType>;
  WriterType::Pointer inputWriter = WriterType::New();
  inputWriter->SetFileName("input.mha");
  inputWriter->SetInput(image);
  inputWriter->Update();
```

(continues on next page)

(continued from previous page)

```
  using ScalarWriterType = itk::ImageFileWriter<ScalarImageType>;
  ScalarWriterType::Pointer outputWriter = ScalarWriterType::New();
  outputWriter->SetFileName("output.mha");
  outputWriter->SetInput(outputImage);
  outputWriter->Update();


  return EXIT_SUCCESS;
}

void
CreateImage(ColorImageType::Pointer image)
{
  // Create a black image with a red square and a green square
  ColorImageType::RegionType region;
  ColorImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ColorImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ColorImageType> imageIterator(image, region);

  itk::Vector<unsigned char, 3> redPixel;
  redPixel[0] = 255;
  redPixel[1] = 0;
  redPixel[2] = 0;

  itk::Vector<unsigned char, 3> greenPixel;
  greenPixel[0] = 0;
  greenPixel[1] = 255;
  greenPixel[2] = 0;

  itk::Vector<unsigned char, 3> blackPixel;
  blackPixel[0] = 0;
  blackPixel[1] = 0;
  blackPixel[2] = 0;

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 150 &&␣
↪imageIterator.GetIndex()[1] > 100 &&
        imageIterator.GetIndex()[1] < 150)
    {
      imageIterator.Set(redPixel);
    }
    else if (imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70 &&␣
↪imageIterator.GetIndex()[1] > 50 &&
             imageIterator.GetIndex()[1] < 70)
```

(continues on next page)

(continued from previous page)

```
    {
      imageIterator.Set(greenPixel);
    }
    else
    {
      imageIterator.Set(blackPixel);
    }

    ++imageIterator;
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TMembershipFunction**>
**class ImageKmeansModelEstimator** : **public** itk::ImageModelEstimatorBase<*TInputImage*, *TMembershipFunction*>
Base class for ImageKmeansModelEstimator object.

itkImageKmeansModelEstimator generates the kmeans model (cluster centers). This object performs clustering of data sets into different clusters, either using user-provided seed points as an initial guess or generating the clusters using a recursive approach when the user provides the number of desired clusters. Each cluster is represented by its cluster center. The two algorithms used are the Generalized Lloyd algorithm (GLA) and the Linde-Buzo-Gray algorithm (LBG). The cluster centers are also referred to as codewords and a table of cluster centers is referred as a codebook.

As required by the GLA algorithm, the initial seed cluster should contain approximate centers of clusters. The GLA algorithm genrates updated cluster centers that result in a lower distortion than the input seed cluster when the input vectors are mapped/classified/labelled using the given codebooks.

If no codebook is provided, the Linde-Buzo-Gray algorithm is used. This algorithm uses the GLA algorithm at its core to generate the centroids of the input vectors (data). However, since there is no initial codebook, LBG first creates a one word codebook (or centroid of one cluster comprising of all the input training vectors). The LBG uses codeword or centroid splitting to create an increasing number of clusters. Each new set of clusters are optimized using the GLA algorithm. The number of clusters increases as $2^{n}$ n= 0, 1, . . . The codebook is expected to be in the form of a vnl matrix, where there are N rows, each row representing the cluster mean of a given cluster. The number of columns in the codebook should be equal to the input image vector dimension.

The threshold parameter controls the ‘’optimality” of the returned codebook, where optimality is related to the least possible mean-squared error distortion that can be found by the algorithm. For larger thresholds, the result will be less optimal. For smaller thresholds, the result will be more optimal. If a more optimal result is desired, then the algorithm will take longer to complete. A reasonable threshold value is 0.01.

If, during the operation of the algorithm, there are any unused clusters or cells, the m_OffsetAdd and m_OffsetMultiply parameters are used to split the cells with the highest distortion. This function will attempt to fill empty cells up to 10 times (unless the overall distortion is zero). Using 0.01 is a reasonable default values for the m_OffsetAdd and m_OffsetMultiply parameters.

If the GLA is unable to resolve the data into the desired number of clusters or cells, only the codewords which were used will be returned.

In terms of clustering, codewords are cluster centers, and a codebook is a table containing all cluster centers. The GLA produces results that are equivalent to the K-means clustering algorithm.

For more information about the algorithms, see A. Gersho and R. M. Gray, {*Vector* Quantization and Signal Compression}, Kluwer Academic Publishers, Boston, MA, 1992.

This object supports data handling of multiband images. The object accepts the input image in vector format only, where each pixel is a vector and each element of the vector corresponds to an entry from 1 particular band of a multiband dataset. A single band image is treated as a vector image with a single element for every vector.

This function is templated over the type of input image. In addition, a second parameter for the MembershipFunction needs to be specified. In this case a Membership function that store cluster centroids models needs to be specified.

The Update() function enables the calculation of the various models, creates the membership function objects and populates them.

Note: There is a second implementation of k-means algorithm in ITK under the itk::Statistics namespace. While this algorithm (GLA/LBG based algorithm) is memory efficient, the other algorithm is time efficient.

**See** KdTreeBasedKmeansEstimator, WeightedCentroidKdTreeGenerator, KdTree

**See** ScalarImageKmeansImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- K Means Cluster Of Pixels In Image

See itk::ImageKmeansModelEstimator for additional documentation.

## 3.10.2 ConnectedComponents

### Assign Contiguous Labels to Connected Regions in an Image

### Synopsis

Assign contiguous labels to connected regions of an image.

### Results

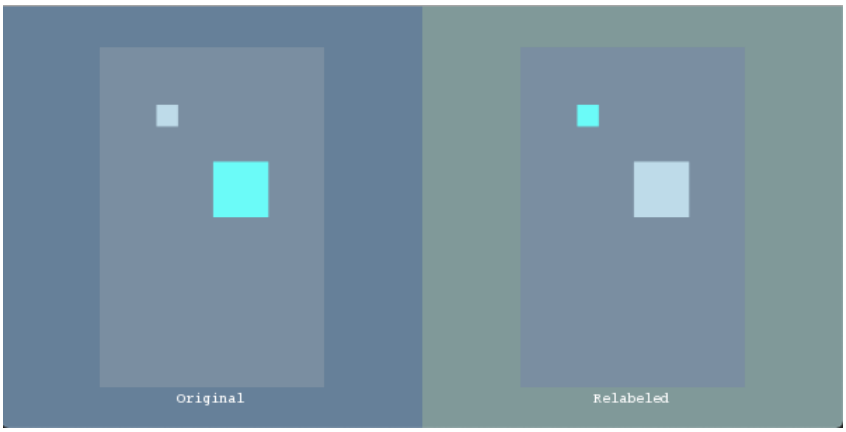



Fig. 373: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkCustomColormapFunction.h"
#include "itkScalarToRGBColormapImageFilter.h"
#include "itkRGBPixel.h"
#include "itkMersenneTwisterRandomVariateGenerator.h"

#include "itkRelabelComponentImageFilter.h"

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using RGBPixelType = itk::RGBPixel<unsigned char>;
using RGBImageType = itk::Image<RGBPixelType, 2>;
using ImageType = itk::Image<unsigned char, 2>;
using ColormapType = itk::Function::CustomColormapFunction<ImageType::PixelType,␣
↪RGBImageType::PixelType>;

static void
CreateImage(ImageType::Pointer image);
static void
CreateRandomColormap(unsigned int size, ColormapType::Pointer colormap);

int
main(int, char *[])
{
  ImageType::Pointer image = ImageType::New();

  CreateImage(image);

  using FilterType = itk::RelabelComponentImageFilter<ImageType, ImageType>;
  FilterType::Pointer relabelFilter = FilterType::New();
  relabelFilter->SetInput(image);

  using ColormapFilterType = itk::ScalarToRGBColormapImageFilter<ImageType,␣
↪RGBImageType>;
  ColormapFilterType::Pointer colormapFilter1 = ColormapFilterType::New();

  ColormapType::Pointer largeColormap = ColormapType::New();
  CreateRandomColormap(255, largeColormap);

  colormapFilter1->SetInput(image);
  colormapFilter1->SetColormap(largeColormap);

  ColormapFilterType::Pointer colormapFilter2 = ColormapFilterType::New();
  colormapFilter2->SetInput(relabelFilter->GetOutput());
  colormapFilter2->SetColormap(largeColormap);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddRGBImage(colormapFilter1->GetOutput(), true, "Original");
```

(continues on next page)

(continued from previous page)

```
  viewer.AddRGBImage(colormapFilter2->GetOutput(), true, "Relabeled");

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  itk::ImageRegionIterator<ImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 150 &&␣
↪imageIterator.GetIndex()[1] > 100 &&
        imageIterator.GetIndex()[1] < 150)
    {
      imageIterator.Set(200);
    }
    else if (imageIterator.GetIndex()[0] > 50 && imageIterator.GetIndex()[0] < 70 &&␣
↪imageIterator.GetIndex()[1] > 50 &&
             imageIterator.GetIndex()[1] < 70)
    {
      imageIterator.Set(100);
    }
    else
    {
      imageIterator.Set(0);
    }

    ++imageIterator;
  }
}

void
CreateRandomColormap(unsigned int size, ColormapType::Pointer colormap)
{
  ColormapType::ChannelType                                    redChannel;
  ColormapType::ChannelType                                    greenChannel;
```

(continues on next page)

(continued from previous page)

```
  ColormapType::ChannelType                                    blueChannel;
  itk::Statistics::MersenneTwisterRandomVariateGenerator::Pointer random =
    itk::Statistics::MersenneTwisterRandomVariateGenerator::New();

  random->SetSeed(8775070);
  for (unsigned int i = 0; i < size; ++i)
  {
    redChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(.3, 1.0)));
    greenChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(.3, 1.0)));
    blueChannel.push_back(static_cast<ColormapType::RealType>(random->
↪GetUniformVariate(.3, 1.0)));
  }
  colormap->SetRedChannel(redChannel);
  colormap->SetGreenChannel(greenChannel);
  colormap->SetBlueChannel(blueChannel);
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RelabelComponentImageFilter** : **public** itk::*InPlaceImageFilter*<*TInputImage*, *TOutputImage*>

Relabel the components in an image such that consecutive labels are used.

RelabelComponentImageFilter remaps the labels associated with the objects in an image (as from the output of ConnectedComponentImageFilter) such that the label numbers are consecutive with no gaps between the label numbers used. By default, the relabeling will also sort the labels based on the size of the object: the largest object will have label #1, the second largest will have label #2, etc. If two labels have the same size their initial order is kept. The sorting by size can be disabled using SetSortByObjectSize.

Label #0 is assumed to be the background and is left unaltered by the relabeling.

RelabelComponentImageFilter is typically used on the output of the ConnectedComponentImageFilter for those applications that want to extract the largest object or the "k" largest objects. Any particular object can be extracted from the relabeled output using a BinaryThresholdImageFilter. A group of objects can be extracted from the relabled output using a ThresholdImageFilter.

Once all the objects are relabeled, the application can query the number of objects and the size of each object. Object sizes are returned in a vector. The size of the background is not calculated. So the size of object #1 is GetSizeOfObjectsInPixels()[0], the size of object #2 is GetSizeOfObjectsInPixels()[1], etc.

If user sets a minimum object size, all objects with fewer pixels than the minimum will be discarded, so that the number of objects reported will be only those remaining. The GetOriginalNumberOfObjects method can be called to find out how many objects were present before the small ones were discarded.

RelabelComponentImageFilter can be run as an "in place" filter, where it will overwrite its output. The default is run out of place (or generate a separate output). "In place" operation can be controlled via methods in the superclass, InPlaceImageFilter::InPlaceOn() and InPlaceImageFilter::InPlaceOff().

**See** ConnectedComponentImageFilter, BinaryThresholdImageFilter, ThresholdImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Assign Contiguous Labels To Connected Regions In An Image

See itk::RelabelComponentImageFilter for additional documentation.

### Extra Largest Connect Component From Binary Image

### Synopsis

Extract the largest connected component from a Binary Image.

### Results

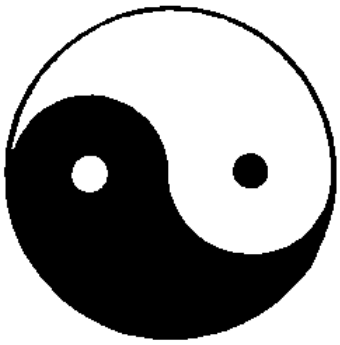


Fig. 374: Input image.

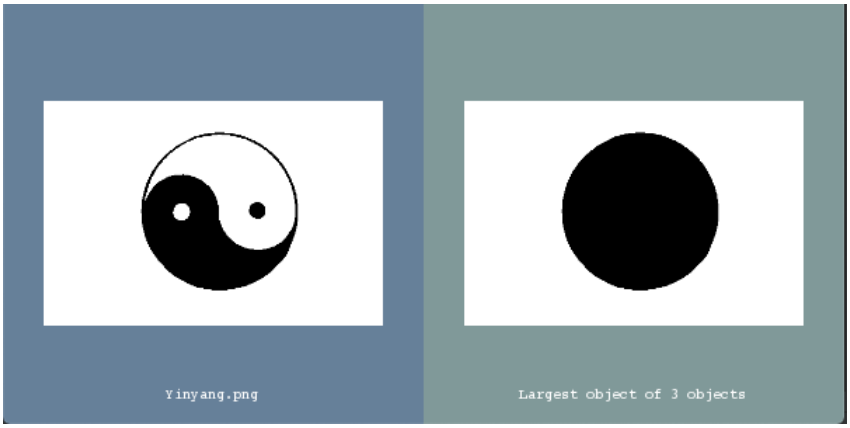



Fig. 375: Output In VTK Window

Output:

```
Number of objects: 3
```

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConnectedComponentImageFilter.h"
#include "itkLabelShapeKeepNObjectsImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cout << "Usage:" << std::endl;
    std::cout << argv[0] << " InputFileName" << std::endl;
  }
  constexpr unsigned int Dimension = 2;
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  using OutputImageType = itk::Image<unsigned short, Dimension>;

  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<ImageType, OutputImageType>;

  ConnectedComponentImageFilterType::Pointer connected =␣
↪ConnectedComponentImageFilterType::New();
  connected->SetInput(reader->GetOutput());
  connected->Update();

  std::cout << "Number of objects: " << connected->GetObjectCount() << std::endl;

  using LabelShapeKeepNObjectsImageFilterType = itk::LabelShapeKeepNObjectsImageFilter
↪<OutputImageType>;
  LabelShapeKeepNObjectsImageFilterType::Pointer labelShapeKeepNObjectsImageFilter =
    LabelShapeKeepNObjectsImageFilterType::New();
  labelShapeKeepNObjectsImageFilter->SetInput(connected->GetOutput());
  labelShapeKeepNObjectsImageFilter->SetBackgroundValue(0);
  labelShapeKeepNObjectsImageFilter->SetNumberOfObjects(1);
  labelShapeKeepNObjectsImageFilter->SetAttribute(
    LabelShapeKeepNObjectsImageFilterType::LabelObjectType::NUMBER_OF_PIXELS);

  using RescaleFilterType = itk::RescaleIntensityImageFilter<OutputImageType,␣
↪ImageType>;
```

(continues on next page)

(continued from previous page)

```
  RescaleFilterType::Pointer rescaleFilter = RescaleFilterType::New();
  rescaleFilter->SetOutputMinimum(0);
  rescaleFilter->SetOutputMaximum(itk::NumericTraits<PixelType>::max());
  rescaleFilter->SetInput(labelShapeKeepNObjectsImageFilter->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  std::stringstream desc;
  desc << "Largest object of " << connected->GetObjectCount() << " objects";
  viewer.AddImage(rescaleFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TInputImage*>
**class ConnectedComponentImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>, **protected**

Label the objects in a binary image.

ConnectedComponentImageFilter labels the objects in a binary image (non-zero pixels are considered to be objects, zero-valued pixels are considered to be background). Each distinct object is assigned a unique label. The filter experiments with some improvements to the existing implementation, and is based on run length encoding along raster lines. If the output background value is set to zero (the default), the final object labels start with 1 and are consecutive. If the output background is set to a non-zero value (by calling the SetBackgroundValue() routine of the filter), the final labels start at 0, and remain consecutive except for skipping the background value as needed. Objects that are reached earlier by a raster order scan have a lower label. This is different to the behaviour of the original connected component image filter which did not produce consecutive labels or impose any particular ordering.

After the filter is executed, ObjectCount holds the number of connected components.

**See** ImageToImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Label Connect Components In Binary Image
- Extra Largest Connect Component From Binary Image

Subclassed by itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, Functor::SimilarPixelsFunctor< TInputImage::ValueType >, TMaskImage >, itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, Functor::SimilarVectorsFunctor< TInputImage::ValueType >, TMaskImage >, itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, TFunctor, TMaskImage >

See itk::ConnectedComponentImageFilter for additional documentation.

### Label Connect Components in Binary Image

### Synopsis

Label connected components in a binary image.

### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

### Code

### C++

```
#include "itkLiThresholdImageFilter.h"
#include "itkHuangThresholdImageFilter.h"
#include "itkIntermodesThresholdImageFilter.h"
#include "itkIsoDataThresholdImageFilter.h"
#include "itkKittlerIllingworthThresholdImageFilter.h"
#include "itkMaximumEntropyThresholdImageFilter.h"
#include "itkMomentsThresholdImageFilter.h"
#include "itkOtsuThresholdImageFilter.h"
#include "itkRenyiEntropyThresholdImageFilter.h"
#include "itkShanbhagThresholdImageFilter.h"
#include "itkTriangleThresholdImageFilter.h"
#include "itkYenThresholdImageFilter.h"

#include "itkConnectedComponentImageFilter.h"
#include "itkLabelToRGBImageFilter.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>
#include <map>
#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

int
main(int argc, char * argv[])
{
  if (argc < 2)
  {
    std::cout << "Usage: " << argv[0];
    std::cout << " inputImageFile";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  using InputPixelType = short;
  using OutputPixelType = int;
```

(continues on next page)

(continued from previous page)

```
  using RGBPixelType = itk::RGBPixel<unsigned char>;

  using InputImageType = itk::Image<InputPixelType, 2>;
  using OutputImageType = itk::Image<OutputPixelType, 2>;
  using RGBImageType = itk::Image<RGBPixelType, 2>;

  using LiFilterType = itk::LiThresholdImageFilter<InputImageType, OutputImageType>;
  using HuangFilterType = itk::HuangThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using IntermodesFilterType = itk::IntermodesThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using IsoDataFilterType = itk::IsoDataThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using KittlerIllingworthFilterType = itk::KittlerIllingworthThresholdImageFilter
↪<InputImageType, OutputImageType>;
  using LiFilterType = itk::LiThresholdImageFilter<InputImageType, OutputImageType>;
  using MaximumEntropyFilterType = itk::MaximumEntropyThresholdImageFilter
↪<InputImageType, OutputImageType>;
  using MomentsFilterType = itk::MomentsThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using OtsuFilterType = itk::OtsuThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using RenyiEntropyFilterType = itk::RenyiEntropyThresholdImageFilter<InputImageType,
↪ OutputImageType>;
  using ShanbhagFilterType = itk::ShanbhagThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using TriangleFilterType = itk::TriangleThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  using YenFilterType = itk::YenThresholdImageFilter<InputImageType, OutputImageType>;

  using ConnectedComponentImageFilterType = itk::ConnectedComponentImageFilter
↪<OutputImageType, OutputImageType>;
  using RGBFilterType = itk::LabelToRGBImageFilter<OutputImageType, RGBImageType>;

  using ReaderType = itk::ImageFileReader<InputImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(argv[1]));

  using FilterContainerType =
    std::map<std::string, itk::HistogramThresholdImageFilter<InputImageType,␣
↪OutputImageType>::Pointer>;
  FilterContainerType filterContainer;

  filterContainer["Huang"] = HuangFilterType::New();
  filterContainer["Intermodes"] = IntermodesFilterType::New();
  filterContainer["IsoData"] = IsoDataFilterType::New();
  filterContainer["KittlerIllingworth"] = KittlerIllingworthFilterType::New();
  filterContainer["Li"] = LiFilterType::New();
  filterContainer["MaximumEntropy"] = MaximumEntropyFilterType::New();
  filterContainer["Moments"] = MomentsFilterType::New();
  filterContainer["Otsu"] = OtsuFilterType::New();
```

(continues on next page)

(continued from previous page)

```
  filterContainer["RenyiEntropy"] = RenyiEntropyFilterType::New();
  filterContainer["Shanbhag"] = ShanbhagFilterType::New();
  filterContainer["Triangle"] = TriangleFilterType::New();
  filterContainer["Yen"] = YenFilterType::New();

  auto it = filterContainer.begin();
  for (it = filterContainer.begin(); it != filterContainer.end(); ++it)
  {
    (*it).second->SetInsideValue(255);
    (*it).second->SetOutsideValue(0);
    (*it).second->SetNumberOfHistogramBins(25);
    (*it).second->SetInput(reader->GetOutput());
    try
    {
      (*it).second->Update();
    }
    catch (itk::ExceptionObject & err)
    {
      std::cout << "Caught exception" << std::endl;
      std::cout << err << std::endl;
      continue;
    }
    ConnectedComponentImageFilterType::Pointer connected =␣
↪ConnectedComponentImageFilterType::New();
    connected->SetInput((*it).second->GetOutput());

    RGBFilterType::Pointer rgbFilter = RGBFilterType::New();
    rgbFilter->SetInput(connected->GetOutput());
    std::stringstream desc;
    desc << (*it).first << " threshold = " << (*it).second->GetThreshold();
    viewer.AddRGBImage(rgbFilter->GetOutput(), true, desc.str());
  }

  viewer.Visualize();
#endif
  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TInputImage*>
**class ConnectedComponentImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>, **protected**

Label the objects in a binary image.

ConnectedComponentImageFilter labels the objects in a binary image (non-zero pixels are considered to be objects, zero-valued pixels are considered to be background). Each distinct object is assigned a unique label. The filter experiments with some improvements to the existing implementation, and is based on run length encoding along raster lines. If the output background value is set to zero (the default), the final object labels start with 1 and are consecutive. If the output background is set to a non-zero value (by calling the SetBackgroundValue() routine of the filter), the final labels start at 0, and remain consecutive except for skipping the background value as needed. Objects that are reached earlier by a raster order scan have a lower label. This is different to the behaviour of the original connected component image filter which did not produce consecutive labels or impose any particular ordering.

After the filter is executed, ObjectCount holds the number of connected components.

**See** ImageToImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Label Connect Components In Binary Image
- Extra Largest Connect Component From Binary Image

Subclassed by itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, Functor::SimilarPixelsFunctor< TInputImage::ValueType >, TMaskImage >, itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, Functor::SimilarVectorsFunctor< TInputImage::ValueType >, TMaskImage >, itk::ConnectedComponentFunctorImageFilter< TInputImage, TOutputImage, TFunctor, TMaskImage >

See itk::ConnectedComponentImageFilter for additional documentation.

### Label Connect Components in Grayscale Image

### Synopsis

Label connected components in a grayscale image.

### Results

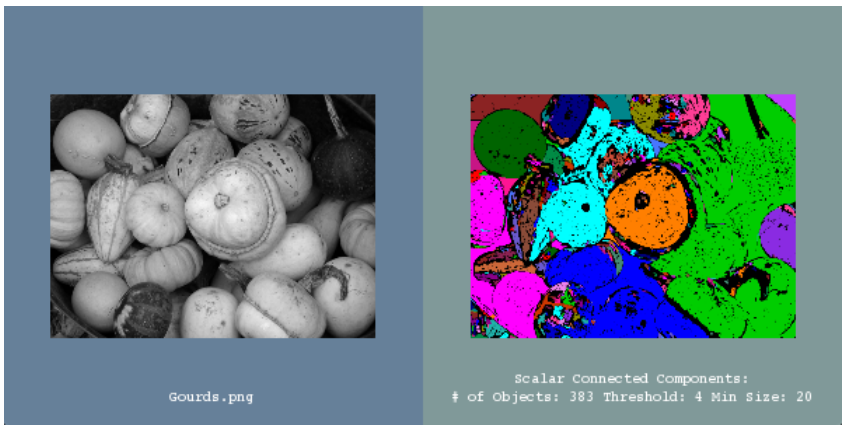



Fig. 376: Output In VTK Window

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkLabelToRGBImageFilter.h"
#include "itkRelabelComponentImageFilter.h"
#include "itkLabelStatisticsImageFilter.h"
```

(continues on next page)

(continued from previous page)

```
#include "itkScalarConnectedComponentImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

template <typename TImage>
static void
CreateImage(TImage * const image);

template <typename TImage, typename TLabelImage>
static void
SummarizeLabelStatistics(TImage * image, TLabelImage * labelImage);

int
main(int argc, char * argv[])
{
  constexpr unsigned int Dimension = 2;
  using PixelType = short;
  using ImageType = itk::Image<PixelType, Dimension>;

  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, Dimension>;

  using LabelPixelType = unsigned int;
  using LabelImageType = itk::Image<LabelPixelType, Dimension>;

  ImageType::Pointer image;
  PixelType          distanceThreshold = 4;
  if (argc < 2)
  {
    image = ImageType::New();
    CreateImage(image.GetPointer());
  }
  else
  {
    using ReaderType = itk::ImageFileReader<ImageType>;
    ReaderType::Pointer reader = ReaderType::New();
    reader->SetFileName(argv[1]);
    reader->Update();

    if (argc > 2)
    {
      distanceThreshold = static_cast<PixelType>(atoi(argv[2]));
    }
    image = reader->GetOutput();
  }


  using ConnectedComponentImageFilterType = itk::ScalarConnectedComponentImageFilter
↪<ImageType, LabelImageType>;

  ConnectedComponentImageFilterType::Pointer connected =␣
↪ConnectedComponentImageFilterType::New();
```

(continues on next page)

(continued from previous page)

```
  connected->SetInput(image);
  connected->SetDistanceThreshold(distanceThreshold);

  using RelabelFilterType = itk::RelabelComponentImageFilter<LabelImageType,␣
↪LabelImageType>;
  RelabelFilterType::Pointer        relabel = RelabelFilterType::New();
  RelabelFilterType::ObjectSizeType minSize = 20;
  if (argc > 3)
  {
    minSize = std::stoi(argv[3]);
  }
  relabel->SetInput(connected->GetOutput());
  relabel->SetMinimumObjectSize(minSize);
  relabel->Update();

  SummarizeLabelStatistics(image.GetPointer(), relabel->GetOutput());

  using RGBFilterType = itk::LabelToRGBImageFilter<LabelImageType, RGBImageType>;
  RGBFilterType::Pointer rgbFilter = RGBFilterType::New();
  rgbFilter->SetInput(relabel->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(
    image.GetPointer(), true, argc > 1 ? itksys::SystemTools::
↪GetFilenameName(argv[1]) : "Generated image");

  std::stringstream desc;
  desc << "Scalar Connected Components:\n# of Objects: " << relabel->
↪GetNumberOfObjects() << " Threshold: "
       << itk::NumericTraits<ConnectedComponentImageFilterType::OutputPixelType>::
↪PrintType(
            connected->GetDistanceThreshold())
       << " Min Size: " << relabel->GetMinimumObjectSize();
  viewer.AddRGBImage(rgbFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}

template <typename TImage>
void
CreateImage(TImage * const image)
{
  // Create an image with 2 connected components
  typename TImage::IndexType start = { { 0, 0 } };
  start[0] = 0;
  start[1] = 0;

  typename TImage::SizeType size;
  unsigned int              NumRows = 200;
  unsigned int              NumCols = 300;
  size[0] = NumRows;
  size[1] = NumCols;
```

(continues on next page)

(continued from previous page)

```
  typename TImage::RegionType region(start, size);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (typename TImage::IndexValueType r = 20; r < 80; r++)
  {
    for (typename TImage::IndexValueType c = 30; c < 100; c++)
    {
      typename TImage::IndexType pixelIndex = { { r, c } };

      image->SetPixel(pixelIndex, 255);
    }
  }

  // Make another square
  for (typename TImage::IndexValueType r = 100; r < 130; r++)
  {
    for (typename TImage::IndexValueType c = 115; c < 160; c++)
    {
      typename TImage::IndexType pixelIndex = { { r, c } };

      image->SetPixel(pixelIndex, 255);
    }
  }
}

template <typename TImage, typename TLabelImage>
void
SummarizeLabelStatistics(TImage * image, TLabelImage * labelImage)
{
  using LabelStatisticsImageFilterType = itk::LabelStatisticsImageFilter<TImage,␣
↪TLabelImage>;
  typename LabelStatisticsImageFilterType::Pointer labelStatisticsImageFilter =␣
↪LabelStatisticsImageFilterType::New();
  labelStatisticsImageFilter->SetLabelInput(labelImage);
  labelStatisticsImageFilter->SetInput(image);
  labelStatisticsImageFilter->UseHistogramsOn(); // needed to compute median
  labelStatisticsImageFilter->Update();

  std::cout << "Number of labels: " << labelStatisticsImageFilter->
↪GetNumberOfLabels() << std::endl;
  std::cout << std::endl;

  using LabelPixelType = typename LabelStatisticsImageFilterType::LabelPixelType;

  for (auto vIt = labelStatisticsImageFilter->GetValidLabelValues().begin();
       vIt != labelStatisticsImageFilter->GetValidLabelValues().end();
       ++vIt)
  {
    if (labelStatisticsImageFilter->HasLabel(*vIt))
    {
      LabelPixelType labelValue = *vIt;
      std::cout << "Label: " << *vIt << std::endl;
      std::cout << "\tmin: " << labelStatisticsImageFilter->GetMinimum(labelValue) <<␣
↪std::endl;
```

(continues on next page)

(continued from previous page)

```
      std::cout << "\tmax: " << labelStatisticsImageFilter->GetMaximum(labelValue) <<␣
↪std::endl;
      std::cout << "\tmedian: " << labelStatisticsImageFilter->GetMedian(labelValue) <
↪< std::endl;
      std::cout << "\tmean: " << labelStatisticsImageFilter->GetMean(labelValue) <<␣
↪std::endl;
      std::cout << "\tsigma: " << labelStatisticsImageFilter->GetSigma(labelValue) <<␣
↪std::endl;
      std::cout << "\tvariance: " << labelStatisticsImageFilter->
↪GetVariance(labelValue) << std::endl;
      std::cout << "\tsum: " << labelStatisticsImageFilter->GetSum(labelValue) << std:
↪:endl;
      std::cout << "\tcount: " << labelStatisticsImageFilter->GetCount(labelValue) <<␣
↪std::endl;
      std::cout << "\tregion: " << labelStatisticsImageFilter->GetRegion(labelValue) <
↪< std::endl;
    }
  }
}
```

#### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**, typename **TMaskImage** = *TInputImage*>
**class ScalarConnectedComponentImageFilter** : **public** itk::ConnectedComponentFunctorImageFilter<*TInputImage*, *T*

A connected components filter that labels the objects in an arbitrary image. Two pixels are similar if they are within threshold of each other. Uses ConnectedComponentFunctorImageFilter.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Label Connect Components In Grayscale Image

See itk::ScalarConnectedComponentImageFilter for additional documentation.

### 3.10.3 KLMRegionGrowing

#### Basic Region Growing

**Warning:** **Fix Problem** Contains problems not fixed from original wiki.

#### Synopsis

Basic region growing.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkRegionGrowImageFilter.h"
#include "itkCastImageFilter.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkImageViewer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using PixelType = int;
constexpr size_t Dimension = 3;
using ImageType = itk::Image<PixelType, Dimension>;

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using RegionGrowImageFilterType = itk::RegionGrowImageFilter<ImageType, ImageType>;
  RegionGrowImageFilterType::Pointer regionGrow = RegionGrowImageFilterType::New();
  float                              lower = 95.0;
  float                              upper = 105.0;
  regionGrow->SetLower(lower);
  regionGrow->SetUpper(upper);

  regionGrow->SetReplaceValue(255);

  // Seed 1: (25, 35)
  ImageType::IndexType seed1;
  seed1[0] = 25;
  seed1[1] = 35;
  regionGrow->SetSeed(seed1);
  regionGrow->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  // Seed 2: (110, 120)
  ImageType::IndexType seed2;
  seed2[0] = 110;
  seed2[1] = 120;
  regionGrow->SetSeed(seed2);
  regionGrow->SetReplaceValue(150);

  regionGrow->SetInput(image);

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector2 = ConnectorType::New();
  connector2->SetInput(image2);

  vtkSmartPointer<vtkImageActor> actor2 = vtkSmartPointer<vtkImageActor>::New();
  actor2->SetInput(connector2->GetOutput());

  // Visualize joined image
  ConnectorType::Pointer addConnector = ConnectorType::New();
  addConnector->SetInput(addFilter->GetOutput());

  vtkSmartPointer<vtkImageActor> addActor = vtkSmartPointer<vtkImageActor>::New();
  addActor->SetInput(addConnector->GetOutput());

  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(900, 300);

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.33, 1.0 };
  double centerViewport[4] = { 0.33, 0.0, 0.66, 1.0 };
  double rightViewport[4] = { 0.66, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);
  leftRenderer->SetBackground(.6, .5, .4);

  vtkSmartPointer<vtkRenderer> centerRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(centerRenderer);
  centerRenderer->SetViewport(centerViewport);
  centerRenderer->SetBackground(.4, .5, .6);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(.4, .5, .6);

  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(actor1);
```

(continues on next page)

(continued from previous page)

```
  centerRenderer->AddActor(actor2);
  rightRenderer->AddActor(addActor);

  leftRenderer->ResetCamera();
  centerRenderer->ResetCamera();
  rightRenderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();


  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 100.0);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
```

(continues on next page)

(continued from previous page)

```
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 100.0);
    }
  }
}
```

## Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class RegionGrowImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>

Base class for RegionGrowImageFilter object.

itkRegionGrowImageFilter is the base class for the RegionGrowImageFilter objects. It provides the basic function definitions that are inherent to a RegionGrowImageFilter objects. It is templated over the type of input and output image.

This object defines the interface for those algorithm that perform feature/object segmentation by merging regions (parts of the image) that are similar in nature based on some metric. As a result parts of the image which belong to the same object gets merged and the region grows.

As an example regarding using this class to implementation of advanced region growing algorithm, itkRegionGrowImageFilterKLM class has been derived from this class. The virtual function ApplyRegionGrowImageFilter() provides the interface to the outside world to extend/enhance the scope of the current algorithm or write other region growing algorithms. The function MergeRegions is interface for the operation that merges two regions.

The local variable GridSize is used to define the initial small regions that the image is fragmented (atomic regions) into. For an 12 x 12 input image, GridSize set equal to [3, 3] will result in 16 initial regions. The default values are set equal to 2. The user can sets the number of desired regions via the m_MaxNumRegions parameter and the algorithm tries to perform region merging until there are only m_MaxNumRegions. If m_MaxNumRegions is more than the number of initial blocks, no region merging occurs.

These blocks are important as the labels associated with these blocks keep changing during the region growing process and at the end, while generating the results, each of these atomic blocks are revisited and the blocks with same labels are considered to belong to the same region.

This object supports data handling of multiband images. The object accepts images in vector format, where each pixel is a vector and each element of the vector corresponds to an entry from 1 particular band of a multiband dataset. The input to this object is assumed to be a multiband vector image, and the output is defined by specific algorithm implementation. The second template parameter is used to generate the the output image and can be modified according the algorithm specific output type.

We expect the user to provide the input to the routine in vector format. A single band image is treated as a vector image with a single element for every vector.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Basic Region Growing

Subclassed by itk::KLMRegionGrowImageFilter< TInputImage, TOutputImage >

See itk::RegionGrowImageFilter for additional documentation.

### 3.10.4 LabelVoting

#### Iterative Hole Filling

#### Synopsis

Fill hole or cavity in one itk::Image

#### Results

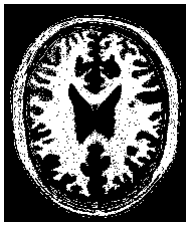


Fig. 377: Input image

Fig. 378: Output image

#### Code

#### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkVotingBinaryIterativeHoleFillingImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 6)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << " <InputFileName> <OutputFileName> <radius> <majority threshold>
↪<number of iterations>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }
```

(continues on next page)

(continued from previous page)

```
  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];

  int          r = std::stoi(argv[3]);
  int          majorityThreshold = std::stoi(argv[4]);
  unsigned int numberOfIterations = std::stoi(argv[5]);

  constexpr unsigned int Dimension = 2;

  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using FilterType = itk::VotingBinaryIterativeHoleFillingImageFilter<ImageType>;
  FilterType::InputSizeType radius;
  radius.Fill(r);

  FilterType::Pointer filter = FilterType::New();
  filter->SetInput(reader->GetOutput());
  filter->SetRadius(radius);
  filter->SetMajorityThreshold(majorityThreshold);
  filter->SetBackgroundValue(itk::NumericTraits<PixelType>::Zero);
  filter->SetForegroundValue(itk::NumericTraits<PixelType>::max());
  filter->SetMaximumNumberOfIterations(numberOfIterations);

  using WriterType = itk::ImageFileWriter<ImageType>;
  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(filter->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

#### Classes demonstrated

template<typename **TImage**>
**class VotingBinaryIterativeHoleFillingImageFilter** : **public** itk::*ImageToImageFilter*<*TImage*, *TImage*>
Fills in holes and cavities by iteratively applying a voting operation.

This filter uses internally the VotingBinaryHoleFillingImageFilter, and runs it iteratively until no pixels are being changed or until it reaches the maximum number of iterations. The purpose of the filter is to fill in holes of medium size (tens of pixels in radius). In principle the number of iterations is related to the size of the holes to be filled in. The larger the holes, the more iteration must be run with this filter in order to fill in the full hole. The size of the neighborhood is also related to the curvature of the hole borders and therefore the hole size. Note that as a collateral effect this filter may also fill in cavities in the external side of structures.

This filter is templated over a single image type because the output image type must be the same as the input image type. This is required in order to make the iterations possible, since the output image of one iteration is taken as the input image for the next iteration.

**See** Image

**See** VotingBinaryImageFilter

**See** VotingBinaryHoleFillingImageFilter

**See** Neighborhood

**See** NeighborhoodOperator

**See** NeighborhoodIterator

See itk::VotingBinaryIterativeHoleFillingImageFilter for additional documentation.

### 3.10.5 LevelSets

#### Segment With Geodesic Active Contour Level Set

#### Synopsis

Segments structures in images based on a user supplied edge potential map.

This example is the same as the one in the ITK software guide. It was re-organized and does not contain any comments about the executed filters. Please refer to the guide for more details.

#### Results

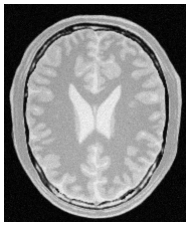


Fig. 379: Input image

Used parameters:

- Left ventricle: 81 114 5.0 1.0 -0.5 3.0 2.0
- Right ventricle: 99 114 5.0 1.0 -0.5 3.0 2.0
- White matter: 56 92 5.0 1.0 -0.3 2.0 10.0
- Gray matter: 40 90 5.0 .5 -0.3 2.0 10.0

### Code

### Python

```
#!/usr/bin/env python

import itk
import argparse

parser = argparse.ArgumentParser(
    description="Segment With Geodesic Active Contour Level Set."
)
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("seed_x", type=int)
parser.add_argument("seed_y", type=int)
parser.add_argument("initial_distance", type=float)
parser.add_argument("sigma", type=float)
parser.add_argument("sigmoid_alpha", type=float)
parser.add_argument("sigmoid_beta", type=float)
parser.add_argument("propagation_scaling", type=float)
parser.add_argument("number_of_iterations", type=int)
args = parser.parse_args()

seedValue = -args.initial_distance

Dimension = 2

InputPixelType = itk.F
OutputPixelType = itk.UC

InputImageType = itk.Image[InputPixelType, Dimension]
OutputImageType = itk.Image[OutputPixelType, Dimension]

ReaderType = itk.ImageFileReader[InputImageType]
WriterType = itk.ImageFileWriter[OutputImageType]

reader = ReaderType.New()
reader.SetFileName(args.input_image)

SmoothingFilterType = itk.CurvatureAnisotropicDiffusionImageFilter[
    InputImageType, InputImageType
]
smoothing = SmoothingFilterType.New()
smoothing.SetTimeStep(0.125)
smoothing.SetNumberOfIterations(5)
smoothing.SetConductanceParameter(9.0)
smoothing.SetInput(reader.GetOutput())
```

(continues on next page)

(continued from previous page)

```
GradientFilterType = itk.GradientMagnitudeRecursiveGaussianImageFilter[
    InputImageType, InputImageType
]
gradientMagnitude = GradientFilterType.New()
gradientMagnitude.SetSigma(args.sigma)
gradientMagnitude.SetInput(smoothing.GetOutput())

SigmoidFilterType = itk.SigmoidImageFilter[InputImageType, InputImageType]
sigmoid = SigmoidFilterType.New()
sigmoid.SetOutputMinimum(0.0)
sigmoid.SetOutputMaximum(1.0)
sigmoid.SetAlpha(args.sigmoid_alpha)
sigmoid.SetBeta(args.sigmoid_beta)
sigmoid.SetInput(gradientMagnitude.GetOutput())

FastMarchingFilterType = itk.FastMarchingImageFilter[InputImageType, InputImageType]
fastMarching = FastMarchingFilterType.New()

GeoActiveContourFilterType = itk.GeodesicActiveContourLevelSetImageFilter[
    InputImageType, InputImageType, InputPixelType
]
geodesicActiveContour = GeoActiveContourFilterType.New()
geodesicActiveContour.SetPropagationScaling(args.propagation_scaling)
geodesicActiveContour.SetCurvatureScaling(1.0)
geodesicActiveContour.SetAdvectionScaling(1.0)
geodesicActiveContour.SetMaximumRMSError(0.02)
geodesicActiveContour.SetNumberOfIterations(args.number_of_iterations)
geodesicActiveContour.SetInput(fastMarching.GetOutput())
geodesicActiveContour.SetFeatureImage(sigmoid.GetOutput())

ThresholdingFilterType = itk.BinaryThresholdImageFilter[InputImageType,␣
↪OutputImageType]
thresholder = ThresholdingFilterType.New()
thresholder.SetLowerThreshold(-1000.0)
thresholder.SetUpperThreshold(0.0)
thresholder.SetOutsideValue(itk.NumericTraits[OutputPixelType].min())
thresholder.SetInsideValue(itk.NumericTraits[OutputPixelType].max())
thresholder.SetInput(geodesicActiveContour.GetOutput())

seedPosition = itk.Index[Dimension]()
seedPosition[0] = args.seed_x
seedPosition[1] = args.seed_y

node = itk.LevelSetNode[InputPixelType, Dimension]()
node.SetValue(seedValue)
node.SetIndex(seedPosition)

seeds = itk.VectorContainer[itk.UI, itk.LevelSetNode[InputPixelType, Dimension]].New()
seeds.Initialize()
seeds.InsertElement(0, node)

fastMarching.SetTrialPoints(seeds)
fastMarching.SetSpeedConstant(1.0)

CastFilterType = itk.RescaleIntensityImageFilter[InputImageType, OutputImageType]

caster1 = CastFilterType.New()
```

(continues on next page)

(continued from previous page)

```
caster2 = CastFilterType.New()
caster3 = CastFilterType.New()
caster4 = CastFilterType.New()

writer1 = WriterType.New()
writer2 = WriterType.New()
writer3 = WriterType.New()
writer4 = WriterType.New()

caster1.SetInput(smoothing.GetOutput())
writer1.SetInput(caster1.GetOutput())
writer1.SetFileName("GeodesicActiveContourImageFilterOutput1.png")
caster1.SetOutputMinimum(itk.NumericTraits[OutputPixelType].min())
caster1.SetOutputMaximum(itk.NumericTraits[OutputPixelType].max())
writer1.Update()

caster2.SetInput(gradientMagnitude.GetOutput())
writer2.SetInput(caster2.GetOutput())
writer2.SetFileName("GeodesicActiveContourImageFilterOutput2.png")
caster2.SetOutputMinimum(itk.NumericTraits[OutputPixelType].min())
caster2.SetOutputMaximum(itk.NumericTraits[OutputPixelType].max())
writer2.Update()

caster3.SetInput(sigmoid.GetOutput())
writer3.SetInput(caster3.GetOutput())
writer3.SetFileName("GeodesicActiveContourImageFilterOutput3.png")
caster3.SetOutputMinimum(itk.NumericTraits[OutputPixelType].min())
caster3.SetOutputMaximum(itk.NumericTraits[OutputPixelType].max())
writer3.Update()

caster4.SetInput(fastMarching.GetOutput())
writer4.SetInput(caster4.GetOutput())
writer4.SetFileName("GeodesicActiveContourImageFilterOutput4.png")
caster4.SetOutputMinimum(itk.NumericTraits[OutputPixelType].min())
caster4.SetOutputMaximum(itk.NumericTraits[OutputPixelType].max())

fastMarching.SetOutputSize(reader.GetOutput().GetBufferedRegion().GetSize())

writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(thresholder.GetOutput())
writer.Update()

print(
    "Max. no. iterations: " + str(geodesicActiveContour.GetNumberOfIterations()) + "\n
↪"
)
print("Max. RMS error: " + str(geodesicActiveContour.GetMaximumRMSError()) + "\n")
print(
    "No. elpased iterations: "
    + str(geodesicActiveContour.GetElapsedIterations())
    + "\n"
)
print("RMS change: " + str(geodesicActiveContour.GetRMSChange()) + "\n")

writer4.Update()
```

(continues on next page)

(continued from previous page)

```
InternalWriterType = itk.ImageFileWriter[InputImageType]

mapWriter = InternalWriterType.New()
mapWriter.SetInput(fastMarching.GetOutput())
mapWriter.SetFileName("GeodesicActiveContourImageFilterOutput4.mha")
mapWriter.Update()

speedWriter = InternalWriterType.New()
speedWriter.SetInput(sigmoid.GetOutput())
speedWriter.SetFileName("GeodesicActiveContourImageFilterOutput3.mha")
speedWriter.Update()

gradientWriter = InternalWriterType.New()
gradientWriter.SetInput(gradientMagnitude.GetOutput())
gradientWriter.SetFileName("GeodesicActiveContourImageFilterOutput2.mha")
gradientWriter.Update()
```

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkGeodesicActiveContourLevelSetImageFilter.h"
#include "itkCurvatureAnisotropicDiffusionImageFilter.h"
#include "itkGradientMagnitudeRecursiveGaussianImageFilter.h"
#include "itkSigmoidImageFilter.h"
#include "itkFastMarchingImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkBinaryThresholdImageFilter.h"

int
main(int argc, char * argv[])
{
  if (argc != 11)
  {
    std::cerr << "Usage: " << argv[0];
    std::cerr << " <InputFileName>  <OutputFileName>";
    std::cerr << " <seedX> <seedY> <InitialDistance>";
    std::cerr << " <Sigma> <SigmoidAlpha> <SigmoidBeta>";
    std::cerr << " <PropagationScaling> <NumberOfIterations>" << std::endl;
    return EXIT_FAILURE;
  }

  const char * inputFileName = argv[1];
  const char * outputFileName = argv[2];
  const int    seedPosX = std::stoi(argv[3]);
  const int    seedPosY = std::stoi(argv[4]);

  const double initialDistance = std::stod(argv[5]);
  const double sigma = std::stod(argv[6]);
  const double alpha = std::stod(argv[7]);
  const double beta = std::stod(argv[8]);
  const double propagationScaling = std::stod(argv[9]);
  const double numberOfIterations = std::stoi(argv[10]);
  const double seedValue = -initialDistance;
```

(continues on next page)

(continued from previous page)

```
  constexpr unsigned int Dimension = 2;

  using InputPixelType = float;
  using InputImageType = itk::Image<InputPixelType, Dimension>;
  using OutputPixelType = unsigned char;
  using OutputImageType = itk::Image<OutputPixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<InputImageType>;
  using WriterType = itk::ImageFileWriter<OutputImageType>;

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName);

  using SmoothingFilterType = itk::CurvatureAnisotropicDiffusionImageFilter
↪<InputImageType, InputImageType>;
  SmoothingFilterType::Pointer smoothing = SmoothingFilterType::New();
  smoothing->SetTimeStep(0.125);
  smoothing->SetNumberOfIterations(5);
  smoothing->SetConductanceParameter(9.0);
  smoothing->SetInput(reader->GetOutput());

  using GradientFilterType = itk::GradientMagnitudeRecursiveGaussianImageFilter
↪<InputImageType, InputImageType>;
  GradientFilterType::Pointer gradientMagnitude = GradientFilterType::New();
  gradientMagnitude->SetSigma(sigma);
  gradientMagnitude->SetInput(smoothing->GetOutput());

  using SigmoidFilterType = itk::SigmoidImageFilter<InputImageType, InputImageType>;
  SigmoidFilterType::Pointer sigmoid = SigmoidFilterType::New();
  sigmoid->SetOutputMinimum(0.0);
  sigmoid->SetOutputMaximum(1.0);
  sigmoid->SetAlpha(alpha);
  sigmoid->SetBeta(beta);
  sigmoid->SetInput(gradientMagnitude->GetOutput());

  using FastMarchingFilterType = itk::FastMarchingImageFilter<InputImageType,␣
↪InputImageType>;
  FastMarchingFilterType::Pointer fastMarching = FastMarchingFilterType::New();

  using GeodesicActiveContourFilterType = itk::
↪GeodesicActiveContourLevelSetImageFilter<InputImageType, InputImageType>;
  GeodesicActiveContourFilterType::Pointer geodesicActiveContour =␣
↪GeodesicActiveContourFilterType::New();
  geodesicActiveContour->SetPropagationScaling(propagationScaling);
  geodesicActiveContour->SetCurvatureScaling(1.0);
  geodesicActiveContour->SetAdvectionScaling(1.0);
  geodesicActiveContour->SetMaximumRMSError(0.02);
  geodesicActiveContour->SetNumberOfIterations(numberOfIterations);
  geodesicActiveContour->SetInput(fastMarching->GetOutput());
  geodesicActiveContour->SetFeatureImage(sigmoid->GetOutput());

  using ThresholdingFilterType = itk::BinaryThresholdImageFilter<InputImageType,␣
↪OutputImageType>;
  ThresholdingFilterType::Pointer thresholder = ThresholdingFilterType::New();
  thresholder->SetLowerThreshold(-1000.0);
  thresholder->SetUpperThreshold(0.0);
```

(continues on next page)

(continued from previous page)

```
  thresholder->SetOutsideValue(itk::NumericTraits<OutputPixelType>::min());
  thresholder->SetInsideValue(itk::NumericTraits<OutputPixelType>::max());
  thresholder->SetInput(geodesicActiveContour->GetOutput());

  using NodeContainer = FastMarchingFilterType::NodeContainer;
  using NodeType = FastMarchingFilterType::NodeType;

  InputImageType::IndexType seedPosition;
  seedPosition[0] = seedPosX;
  seedPosition[1] = seedPosY;

  NodeContainer::Pointer seeds = NodeContainer::New();
  NodeType               node;
  node.SetValue(seedValue);
  node.SetIndex(seedPosition);

  seeds->Initialize();
  seeds->InsertElement(0, node);

  fastMarching->SetTrialPoints(seeds);
  fastMarching->SetSpeedConstant(1.0);

  using CastFilterType = itk::RescaleIntensityImageFilter<InputImageType,␣
↪OutputImageType>;

  CastFilterType::Pointer caster1 = CastFilterType::New();
  CastFilterType::Pointer caster2 = CastFilterType::New();
  CastFilterType::Pointer caster3 = CastFilterType::New();
  CastFilterType::Pointer caster4 = CastFilterType::New();

  WriterType::Pointer writer1 = WriterType::New();
  WriterType::Pointer writer2 = WriterType::New();
  WriterType::Pointer writer3 = WriterType::New();
  WriterType::Pointer writer4 = WriterType::New();

  caster1->SetInput(smoothing->GetOutput());
  writer1->SetInput(caster1->GetOutput());
  writer1->SetFileName("GeodesicActiveContourImageFilterOutput1.png");
  caster1->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  caster1->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());
  writer1->Update();

  caster2->SetInput(gradientMagnitude->GetOutput());
  writer2->SetInput(caster2->GetOutput());
  writer2->SetFileName("GeodesicActiveContourImageFilterOutput2.png");
  caster2->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  caster2->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());
  writer2->Update();

  caster3->SetInput(sigmoid->GetOutput());
  writer3->SetInput(caster3->GetOutput());
  writer3->SetFileName("GeodesicActiveContourImageFilterOutput3.png");
  caster3->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  caster3->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());
  writer3->Update();

  caster4->SetInput(fastMarching->GetOutput());
```

(continues on next page)

(continued from previous page)

```
  writer4->SetInput(caster4->GetOutput());
  writer4->SetFileName("GeodesicActiveContourImageFilterOutput4.png");
  caster4->SetOutputMinimum(itk::NumericTraits<OutputPixelType>::min());
  caster4->SetOutputMaximum(itk::NumericTraits<OutputPixelType>::max());

  InputImageType::Pointer inImage = reader->GetOutput();
  fastMarching->SetOutputDirection(inImage->GetDirection());
  fastMarching->SetOutputOrigin(inImage->GetOrigin());
  fastMarching->SetOutputRegion(inImage->GetBufferedRegion());
  fastMarching->SetOutputSpacing(inImage->GetSpacing());

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName(outputFileName);
  writer->SetInput(thresholder->GetOutput());
  try
  {
    writer->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  std::cout << std::endl;
  std::cout << "Max. no. iterations: " << geodesicActiveContour->
↪GetNumberOfIterations() << std::endl;
  std::cout << "Max. RMS error: " << geodesicActiveContour->GetMaximumRMSError() <<␣
↪std::endl;
  std::cout << std::endl;
  std::cout << "No. elpased iterations: " << geodesicActiveContour->
↪GetElapsedIterations() << std::endl;
  std::cout << "RMS change: " << geodesicActiveContour->GetRMSChange() << std::endl;

  try
  {
    writer4->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  using InternalWriterType = itk::ImageFileWriter<InputImageType>;

  InternalWriterType::Pointer mapWriter = InternalWriterType::New();
  mapWriter->SetInput(fastMarching->GetOutput());
  mapWriter->SetFileName("GeodesicActiveContourImageFilterOutput4.mha");
  try
  {
    mapWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
```

(continues on next page)

(continued from previous page)

```
  }

  InternalWriterType::Pointer speedWriter = InternalWriterType::New();
  speedWriter->SetInput(sigmoid->GetOutput());
  speedWriter->SetFileName("GeodesicActiveContourImageFilterOutput3.mha");
  try
  {
    speedWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  InternalWriterType::Pointer gradientWriter = InternalWriterType::New();
  gradientWriter->SetInput(gradientMagnitude->GetOutput());
  gradientWriter->SetFileName("GeodesicActiveContourImageFilterOutput2.mha");
  try
  {
    gradientWriter->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TFeatureImage**, typename **TOutputPixelType** = float>
**class GeodesicActiveContourLevelSetImageFilter** : **public** itk::SegmentationLevelSetImageFilter<*TInputImage*, *T*

Segments structures in images based on a user supplied edge potential map.

$$g(I) = 1/(1 + |(\nabla * G)(I)|)$$

$$g(I) = \exp^{-|(\nabla * G)(I)|}$$

**IMPORTANT** The SegmentationLevelSetImageFilter class and the GeodesicActiveContourLevelSetFunction class contain additional information necessary to gain full understanding of how to use this filter.

**OVERVIEW** This class is a level set method segmentation filter. An initial contour is propagated outwards (or inwards) until it ''sticks" to the shape boundaries. This is done by using a level set speed function based on a user supplied edge potential map.

**INPUTS** This filter requires two inputs. The first input is a initial level set. The initial level set is a real image which contains the initial contour/surface as the zero level set. For example, a signed distance function from the initial contour/surface is typically used. Unlike the simpler ShapeDetectionLevelSetImageFilter the initial contour does not have to lie wholly within the shape to be segmented. The initial contour is allow to overlap the shape boundary. The extra advection term in the update equation behaves like a doublet and attracts the contour to the boundary. This approach for segmentation follows that of Caselles et al (1997).

The second input is the feature image. For this filter, this is the edge potential map. General characteristics of an edge potential map is that it has values close to zero in regions near the edges and values close to one inside the shape itself. Typically, the edge potential map is compute from the image gradient, for example:

where $I$ is image intensity and $(\nabla * G)$ is the derivative of Gaussian operator.

This implementation allows the user to set the weights between the propagation, advection and curvature term using methods SetPropagationScaling(), SetAdvectionScaling(), SetCurvatureScaling(). In general, the larger the CurvatureScaling, the smoother the resulting contour. To follow the implementation in Caselles et al paper, set the PropagationScaling to $c$ (the inflation or ballon force) and AdvectionScaling and CurvatureScaling both to 1.0.

See SegmentationLevelSetImageFilter and SparseFieldLevelSetImageFilter for more information on Inputs.

**PARAMETERS** The PropagationScaling parameter can be used to switch from propagation outwards (POSITIVE scaling parameter) versus propagating inwards (NEGATIVE scaling parameter).

**OUTPUTS** The filter outputs a single, scalar, real-valued image. Negative values in the output image represent the inside of the segmented region and positive values in the image represent the outside of the segmented region. The zero crossings of the image correspond to the position of the propagating front.

See SparseFieldLevelSetImageFilter and SegmentationLevelSetImageFilter for more information.

**REFERENCES**

"Geodesic Active Contours", V. Caselles, R. Kimmel and G. Sapiro. International Journal on Computer Vision, Vol 22, No. 1, pp 61-97, 1997

**See** SegmentationLevelSetImageFilter

**See** GeodesicActiveContourLevelSetFunction

**See** SparseFieldLevelSetImageFilter

See itk::GeodesicActiveContourLevelSetImageFilter for additional documentation.

### 3.10.6 RegionGrowing

#### Connected Components in Image

> **Warning: Fix Problem** Contains problems not fixed from original wiki.

#### Synopsis

Find connected components in an image.

### Results

**Note:** **Help Wanted** Implementation of Results for sphinx examples containing this message. Reconfiguration of CMakeList.txt may be necessary. *Write An Example <https://itk.org/ITKExamples/Documentation/Contribute/WriteANewExample.html>*

### Code

#### C++

```
#include "itkImage.h"
#include "itkImageFileWriter.h"
#include "itkConnectedThresholdImageFilter.h"
#include "itkCastImageFilter.h"

#include <itkImageToVTKImageFilter.h>

#include "vtkImageViewer.h"
#include "vtkRenderWindowInteractor.h"
#include "vtkSmartPointer.h"
#include "vtkImageActor.h"
#include "vtkInteractorStyleImage.h"
#include "vtkRenderer.h"

using PixelType = unsigned char;
constexpr size_t Dimension = 3;
using ImageType = itk::Image<PixelType, Dimension>;

static void
CreateImage(ImageType::Pointer image);

int
main(int argc, char * argv[])
{
  ImageType::Pointer image = ImageType::New();
  CreateImage(image);

  using ConnectedFilterType = itk::ConnectedThresholdImageFilter<ImageType, ImageType>
↪;
  ConnectedFilterType::Pointer connectedThreshold = ConnectedFilterType::New();
  float                        lower = 95.0;
  float                        upper = 105.0;
  connectedThreshold->SetLower(lower);
  connectedThreshold->SetUpper(upper);

  connectedThreshold->SetReplaceValue(255);

  // Seed 1: (25, 35)
  ImageType::IndexType seed1;
  seed1[0] = 25;
  seed1[1] = 35;
  connectedThreshold->SetSeed(seed1);
  connectedThreshold->SetInput(image);
```

(continues on next page)

(continued from previous page)

```
  // Seed 2: (110, 120)
  ImageType::IndexType seed2;
  seed2[0] = 110;
  seed2[1] = 120;
  connectedThreshold->SetSeed(seed2);
  connectedThreshold->SetReplaceValue(150);

  connectedThreshold->SetInput(image);

  // Visualize
  using ConnectorType = itk::ImageToVTKImageFilter<ImageType>;
  ConnectorType::Pointer connector2 = ConnectorType::New();
  connector2->SetInput(image);

  vtkSmartPointer<vtkImageActor> actor2 = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  actor2->SetInput(connector->GetOutput());
#else
  connector2->Update();
  actor2->GetMapper()->SetInputData(connector2->GetOutput());
#endif

  // Visualize joined image
  ConnectorType::Pointer addConnector = ConnectorType::New();
  addConnector->SetInput(connectedThreshold->GetOutput());

  vtkSmartPointer<vtkImageActor> addActor = vtkSmartPointer<vtkImageActor>::New();
#if VTK_MAJOR_VERSION <= 5
  addActor->SetInput(connector->GetOutput());
#else
  addConnector->Update();
  addActor->GetMapper()->SetInputData(addConnector->GetOutput());
#endif

  // There will be one render window
  vtkSmartPointer<vtkRenderWindow> renderWindow = vtkSmartPointer<vtkRenderWindow>::
↪New();
  renderWindow->SetSize(900, 300);

  vtkSmartPointer<vtkRenderWindowInteractor> interactor = vtkSmartPointer
↪<vtkRenderWindowInteractor>::New();
  interactor->SetRenderWindow(renderWindow);

  // Define viewport ranges
  // (xmin, ymin, xmax, ymax)
  double leftViewport[4] = { 0.0, 0.0, 0.33, 1.0 };
  double centerViewport[4] = { 0.33, 0.0, 0.66, 1.0 };
  double rightViewport[4] = { 0.66, 0.0, 1.0, 1.0 };

  // Setup both renderers
  vtkSmartPointer<vtkRenderer> leftRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(leftRenderer);
  leftRenderer->SetViewport(leftViewport);
  leftRenderer->SetBackground(.6, .5, .4);

  vtkSmartPointer<vtkRenderer> centerRenderer = vtkSmartPointer<vtkRenderer>::New();
```

(continues on next page)

(continued from previous page)

```
  renderWindow->AddRenderer(centerRenderer);
  centerRenderer->SetViewport(centerViewport);
  centerRenderer->SetBackground(.4, .5, .6);

  vtkSmartPointer<vtkRenderer> rightRenderer = vtkSmartPointer<vtkRenderer>::New();
  renderWindow->AddRenderer(rightRenderer);
  rightRenderer->SetViewport(rightViewport);
  rightRenderer->SetBackground(.4, .5, .6);

  // Add the sphere to the left and the cube to the right
  leftRenderer->AddActor(actor1);
  centerRenderer->AddActor(actor2);
  rightRenderer->AddActor(addActor);

  leftRenderer->ResetCamera();
  centerRenderer->ResetCamera();
  rightRenderer->ResetCamera();

  renderWindow->Render();

  vtkSmartPointer<vtkInteractorStyleImage> style = vtkSmartPointer
↪<vtkInteractorStyleImage>::New();
  interactor->SetInteractorStyle(style);

  interactor->Start();


  return EXIT_SUCCESS;
}

void
CreateImage(ImageType::Pointer image)
{
  // Create an image with 2 connected components
  ImageType::RegionType region;
  ImageType::IndexType  start;
  start[0] = 0;
  start[1] = 0;

  ImageType::SizeType size;
  size[0] = 200;
  size[1] = 300;

  region.SetSize(size);
  region.SetIndex(start);

  image->SetRegions(region);
  image->Allocate();

  // Make a square
  for (unsigned int r = 20; r < 80; r++)
  {
    for (unsigned int c = 30; c < 100; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;
```

(continues on next page)

(continued from previous page)

```
      image->SetPixel(pixelIndex, 100.0);
    }
  }

  // Make another square
  for (unsigned int r = 100; r < 130; r++)
  {
    for (unsigned int c = 115; c < 160; c++)
    {
      ImageType::IndexType pixelIndex;
      pixelIndex[0] = r;
      pixelIndex[1] = c;

      image->SetPixel(pixelIndex, 100.0);
    }
  }
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ConnectedThresholdImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Label pixels that are connected to a seed and lie within a range of values.

ConnectedThresholdImageFilter labels pixels with ReplaceValue that are connected to an initial Seed AND lie within a Lower and Upper threshold range.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Connected Components In Image

See itk::ConnectedThresholdImageFilter for additional documentation.

### Segment Pixels With Similar Statistics

### Synopsis

Segment pixels with similar statistics using connectivity.

### Results

### Code

### C++

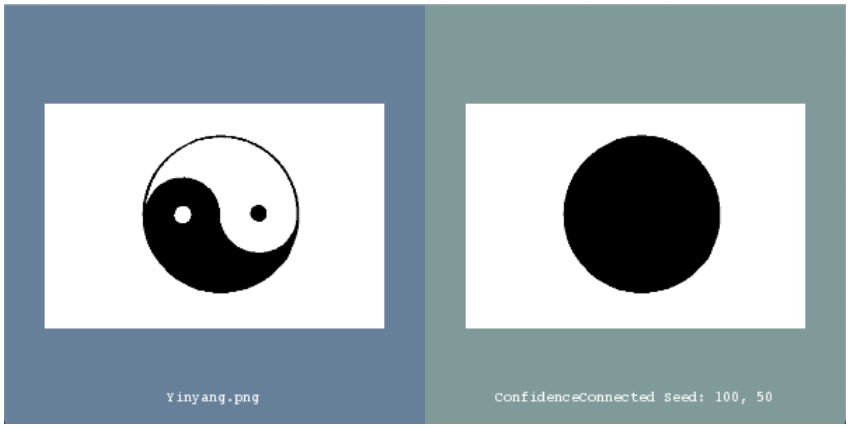



Fig. 380: Output In VTK Window

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkConfidenceConnectedImageFilter.h"

#include "itksys/SystemTools.hxx"
#include <sstream>

#ifdef ENABLE_QUICKVIEW
#  include "QuickView.h"
#endif

using ImageType = itk::Image<unsigned char, 2>;

int
main(int argc, char * argv[])
{
  if (argc < 4)
  {
    std::cerr << "Required: filename.png seedX seedY" << std::endl;

    return EXIT_FAILURE;
  }
  std::string inputFileName = argv[1];

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(inputFileName.c_str());
  reader->Update();

  using ConfidenceConnectedFilterType = itk::ConfidenceConnectedImageFilter<ImageType,
↪ ImageType>;
  ConfidenceConnectedFilterType::Pointer confidenceConnectedFilter =␣
↪ConfidenceConnectedFilterType::New();
  confidenceConnectedFilter->SetInitialNeighborhoodRadius(3);
  confidenceConnectedFilter->SetMultiplier(3);
  confidenceConnectedFilter->SetNumberOfIterations(25);
  confidenceConnectedFilter->SetReplaceValue(255);

  // Set seed
```

(continues on next page)

(continued from previous page)

```
  ImageType::IndexType seed;
  seed[0] = std::stoi(argv[2]);
  seed[1] = std::stoi(argv[3]);
  confidenceConnectedFilter->SetSeed(seed);
  confidenceConnectedFilter->SetInput(reader->GetOutput());

#ifdef ENABLE_QUICKVIEW
  QuickView viewer;
  viewer.AddImage(reader->GetOutput(), true, itksys::SystemTools::
↪GetFilenameName(inputFileName));

  std::stringstream desc;
  desc << "ConfidenceConnected Seed: " << seed[0] << ", " << seed[1];
  viewer.AddImage(confidenceConnectedFilter->GetOutput(), true, desc.str());

  viewer.Visualize();
#endif

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class ConfidenceConnectedImageFilter** : **public** itk::*ImageToImageFilter<TInputImage*, *TOutputImage*>

Segment pixels with similar statistics using connectivity.

This filter extracts a connected set of pixels whose pixel intensities are consistent with the pixel statistics of a seed point. The mean and variance across a neighborhood (8-connected, 26-connected, etc.) are calculated for a seed point. Then pixels connected to this seed point whose values are within the confidence interval for the seed point are grouped. The width of the confidence interval is controlled by the "Multiplier" variable (the confidence interval is the mean plus or minus the "Multiplier" times the standard deviation). If the intensity variations across a segment were gaussian, a "Multiplier" setting of 2.5 would define a confidence interval wide enough to capture 99% of samples in the segment.

After this initial segmentation is calculated, the mean and variance are re-calculated. All the pixels in the previous segmentation are used to calculate the mean the standard deviation (as opposed to using the pixels in the neighborhood of the seed point). The segmentation is then recalculated using these refined estimates for the mean and variance of the pixel values. This process is repeated for the specified number of iterations. Setting the "NumberOfIterations" to zero stops the algorithm after the initial segmentation from the seed point.

NOTE: the lower and upper threshold are restricted to lie within the valid numeric limits of the input data pixel type. Also, the limits may be adjusted to contain the seed point's intensity.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- SegmentPixelsWithSimilarStats

See itk::ConfidenceConnectedImageFilter for additional documentation.

### 3.10.7 Voronoi

#### Voronoi Diagram

#### Synopsis

Voronoi diagram.

#### Results

**Warning:** **Fix Errors** Example contains errors needed to be fixed for proper output.

Output:

```
Seed No.0: At (25,25)
Boundary Vertices List (in order):4,8,7,0,1,
Neighbors (Seed No.):2,3,1,

Seed No.1: At (75,25)
Boundary Vertices List (in order):6,11,7,0,2,
Neighbors (Seed No.):2,4,0,

Seed No.2: At (50,50)
Boundary Vertices List (in order):2,0,1,3,
Neighbors (Seed No.):0,1,3,4,

Seed No.3: At (25,75)
Boundary Vertices List (in order):5,9,4,1,3,
Neighbors (Seed No.):2,0,4,

Seed No.4: At (75,75)
Boundary Vertices List (in order):5,10,6,2,3,
Neighbors (Seed No.):2,3,1,

Vertices Informations:
Vertices No.0: At (50,25)
Vertices No.1: At (25,50)
Vertices No.2: At (75,50)
Vertices No.3: At (50,75)
Vertices No.4: At (0,50)
Vertices No.5: At (50,100)
Vertices No.6: At (100,50)
Vertices No.7: At (50,0)
Vertices No.8: At (0,0)
Vertices No.9: At (0,100)
Vertices No.10: At (100,100)
Vertices No.11: At (100,0)
IIBI-SYNAPSE006:bin mseng$ ./VoronoiDiagram
Seed No.0: At (25,25)
Boundary Vertices List (in order):4,8,7,0,1,
Neighbors (Seed No.):2,3,1,

Seed No.1: At (75,25)
```

(continues on next page)

(continued from previous page)

```
Boundary Vertices List (in order):6,11,7,0,2,
Neighbors (Seed No.):2,4,0,

Seed No.2: At (50,50)
Boundary Vertices List (in order):2,0,1,3,
Neighbors (Seed No.):0,1,3,4,

Seed No.3: At (25,75)
Boundary Vertices List (in order):5,9,4,1,3,
Neighbors (Seed No.):2,0,4,

Seed No.4: At (75,75)
Boundary Vertices List (in order):5,10,6,2,3,
Neighbors (Seed No.):2,3,1,

Vertices Informations:
Vertices No.0: At (50,25)
Vertices No.1: At (25,50)
Vertices No.2: At (75,50)
Vertices No.3: At (50,75)
Vertices No.4: At (0,50)
Vertices No.5: At (50,100)
Vertices No.6: At (100,50)
Vertices No.7: At (50,0)
Vertices No.8: At (0,0)
Vertices No.9: At (0,100)
Vertices No.10: At (100,100)
Vertices No.11: At (100,0)
```

## Code

### C++

```
#include "itkVoronoiDiagram2DGenerator.h"
#include "itkImageFileWriter.h"
#include "itkVTKPolyDataWriter.h"

int
main(int, char *[])
{
  constexpr double height = 100;
  constexpr double width = 100;

  using VoronoiDiagramType = itk::VoronoiDiagram2D<double>;
  using VoronoiGeneratorType = itk::VoronoiDiagram2DGenerator<double>;

  using PointType = VoronoiDiagramType::PointType;
  using CellType = VoronoiDiagramType::CellType;
  using CellAutoPointer = VoronoiDiagramType::CellAutoPointer;
  using PointIdIterator = CellType::PointIdIterator;
  using NeighborIdIterator = VoronoiDiagramType::NeighborIdIterator;

  VoronoiDiagramType::Pointer   voronoiDiagram = VoronoiDiagramType::New();
  VoronoiGeneratorType::Pointer voronoiGenerator = VoronoiGeneratorType::New();
```

(continues on next page)

(continued from previous page)

```
  PointType insize;
  insize[0] = width;
  insize[1] = height;
  voronoiGenerator->SetBoundary(insize);

  // Create a list of seeds
  std::vector<PointType> seeds;
  PointType              seed0;
  seed0[0] = 50;
  seed0[1] = 50;
  seeds.push_back(seed0);

  PointType seed1;
  seed1[0] = 25;
  seed1[1] = 25;
  seeds.push_back(seed1);

  PointType seed2;
  seed2[0] = 75;
  seed2[1] = 25;
  seeds.push_back(seed2);

  PointType seed3;
  seed3[0] = 25;
  seed3[1] = 75;
  seeds.push_back(seed3);

  PointType seed4;
  seed4[0] = 75;
  seed4[1] = 75;
  seeds.push_back(seed4);

  for (const auto & seed : seeds)
  {
    voronoiGenerator->AddOneSeed(seed);
  }

  voronoiGenerator->Update();
  voronoiDiagram = voronoiGenerator->GetOutput();

  for (unsigned int i = 0; i < seeds.size(); i++)
  {
    PointType currP = voronoiDiagram->GetSeed(i);
    std::cout << "Seed No." << i << ": At (" << currP[0] << "," << currP[1] << ")" <<␣
↪std::endl;
    std::cout << "  Boundary Vertices List (in order):";
    CellAutoPointer currCell;
    voronoiDiagram->GetCellId(i, currCell);
    PointIdIterator currCellP;
    for (currCellP = currCell->PointIdsBegin(); currCellP != currCell->PointIdsEnd();␣
↪++currCellP)
    {
      std::cout << (*currCellP) << ",";
    }
    std::cout << std::endl;
    std::cout << "  Neighbors (Seed No.):";
```

(continues on next page)

(continued from previous page)

```
    NeighborIdIterator currNeibor;
    for (currNeibor = voronoiDiagram->NeighborIdsBegin(i); currNeibor !=␣
↪voronoiDiagram->NeighborIdsEnd(i);
         ++currNeibor)
    {
      std::cout << (*currNeibor) << ",";
    }
    std::cout << std::endl << std::endl;
  }

  std::cout << "Vertices Informations:" << std::endl;
  VoronoiDiagramType::VertexIterator allVerts;
  int                                j = 0;
  for (allVerts = voronoiDiagram->VertexBegin(); allVerts != voronoiDiagram->
↪VertexEnd(); ++allVerts)
  {
    std::cout << "Vertices No." << j;
    j++;
#if ITK_VERSION_MAJOR < 4
    std::cout << ": At (" << (*allVerts)[0] << "," << (*allVerts)[1] << ")" << std::
↪endl;
#else
    std::cout << ": At (" << (allVerts.Value())[0] << "," << (allVerts.Value())[1] <<
↪")" << std::endl;
#endif
  }

  // Write the resulting mesh
  using WriterType = itk::VTKPolyDataWriter<VoronoiDiagramType::Superclass>;
  WriterType::Pointer vtkPolyDataWriter = WriterType::New();
  vtkPolyDataWriter->SetInput(voronoiDiagram);
  vtkPolyDataWriter->SetFileName("voronoi.vtk");
  vtkPolyDataWriter->Update();

  // Setup an image to visualize the input
  {
    using ImageType = itk::Image<unsigned char, 2>;

    ImageType::IndexType start;
    start.Fill(0);

    ImageType::SizeType size;
    size.Fill(100);

    ImageType::RegionType region(start, size);

    ImageType::Pointer image = ImageType::New();
    image->SetRegions(region);
    image->Allocate();
    image->FillBuffer(0);

    ImageType::IndexType ind;
    ind[0] = 50;
    ind[1] = 50;
    image->SetPixel(ind, 255);

    ind[0] = 25;
```

(continues on next page)

(continued from previous page)

```
    ind[1] = 25;
    image->SetPixel(ind, 255);

    ind[0] = 75;
    ind[1] = 25;
    image->SetPixel(ind, 255);

    ind[0] = 25;
    ind[1] = 75;
    image->SetPixel(ind, 255);

    ind[0] = 75;
    ind[1] = 75;
    image->SetPixel(ind, 255);

    using ImageWriterType = itk::ImageFileWriter<ImageType>;
    ImageWriterType::Pointer imageFileWriter = ImageWriterType::New();
    imageFileWriter->SetFileName("image.png");
    imageFileWriter->SetInput(image);
    imageFileWriter->Update();
  }

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

template<typename **TCoordType**>
**class VoronoiDiagram2D** : **public** itk::*Mesh*<*TCoordType*, 2, DefaultDynamicMeshTraits<*TCoordType*, 2, 2, *TCoordType*>>
Implements the 2-Dimensional Voronoi Diagram.

Given a set of seed points, the Voronoi Diagram partitions the plane into regions, each region is a collection of all pixels that is closest to one particular seed point than to other seed points. VoronoiDiagram2D is a mesh structure for storing the Voronoi Diagram, can be Generated by itkVoronoiDiagram2DGenerator.

Template parameters for VoronoiDiagram2D:

TCoordType = The type associated with the coordination of the seeds and the resulting vertices.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Voronoi Diagram

See itk::VoronoiDiagram2D for additional documentation.

template<typename **TCoordType**>
**class VoronoiDiagram2DGenerator** : **public** itk::MeshSource<*VoronoiDiagram2D*<*TCoordType*>>
Implement the Sweep Line Algorithm for the construction of the 2D Voronoi Diagram.

Detailed information on this method can be found in: "A sweepline algorithm for Voronoi diagrams." S. Fortune, Algorithmica 2, 153-174, 1987.

Input parameters are: (1) Size of the region. (2) Seed points coordinates. These coordinates can also be randomly set.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Voronoi Diagram

**Template Parameters**

- `TCoordType`: The type associated with the coordination of the seeds and the resulting vertices.

See itk::VoronoiDiagram2DGenerator for additional documentation.

### 3.10.8 Watersheds

#### Morphological Watershed Segmentation

#### Synopsis

Morphological watershed segmentation.

#### Results

Fig. 381: input.png

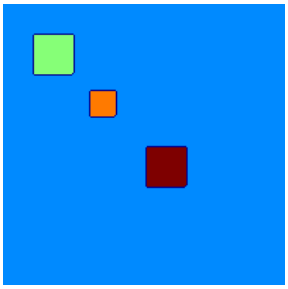


Fig. 382: output_20_3.png

Output:

```
Running with:
Threshold: 20
Level: 3
```

### Code

### C++

```
#include <iostream>

#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkScalarToRGBPixelFunctor.h"
#include "itkUnaryFunctorImageFilter.h"
#include "itkVectorGradientAnisotropicDiffusionImageFilter.h"
#include "itkVectorMagnitudeImageFilter.h"
#include "itkMorphologicalWatershedImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkScalarToRGBColormapImageFilter.h"
#include "itkGradientMagnitudeImageFilter.h"

// Run with:
// ./WatershedImageFilter threshold level
// e.g.
// ./WatershedImageFilter 0.005 .5
// (A rule of thumb is to set the Threshold to be about 1 / 100 of the Level.)

using UnsignedCharImageType = itk::Image<unsigned char, 2>;
using FloatImageType = itk::Image<float, 2>;
using RGBPixelType = itk::RGBPixel<unsigned char>;
using RGBImageType = itk::Image<RGBPixelType, 2>;
using LabeledImageType = itk::Image<unsigned long, 2>;

static void
CreateImage(UnsignedCharImageType::Pointer image);
static void
PerformSegmentation(FloatImageType::Pointer image, const float threshold, const float␣
↪level);

int
main(int argc, char * argv[])
{
  // Verify arguments
  if (argc < 3)
  {
    std::cerr << "Parameters " << std::endl;
    std::cerr << " threshold level" << std::endl;
    return 1;
  }

  // Parse arguments
  std::string       strThreshold = argv[1];
  float             threshold = 0.0;
  std::stringstream ssThreshold;
  ssThreshold << strThreshold;
  ssThreshold >> threshold;

  std::string       strLevel = argv[2];
  float             level = 0.0;
  std::stringstream ssLevel;
  ssLevel << strLevel;
```

(continues on next page)

(continued from previous page)

```
  ssLevel >> level;

  // Output arguments
  std::cout << "Running with:" << std::endl
            << "Threshold: " << threshold << std::endl
            << "Level: " << level << std::endl;

  UnsignedCharImageType::Pointer image = UnsignedCharImageType::New();
  CreateImage(image);

  using GradientMagnitudeImageFilterType = itk::GradientMagnitudeImageFilter
↪<UnsignedCharImageType, FloatImageType>;
  GradientMagnitudeImageFilterType::Pointer gradientMagnitudeImageFilter =␣
↪GradientMagnitudeImageFilterType::New();
  gradientMagnitudeImageFilter->SetInput(image);
  gradientMagnitudeImageFilter->Update();

  // Custom parameters
  PerformSegmentation(gradientMagnitudeImageFilter->GetOutput(), threshold, level);

  return EXIT_SUCCESS;
}


void
CreateImage(UnsignedCharImageType::Pointer image)
{
  // Create a white image with 3 dark regions of different values

  itk::Index<2> start;
  start.Fill(0);

  itk::Size<2> size;
  size.Fill(200);

  itk::ImageRegion<2> region(start, size);
  image->SetRegions(region);
  image->Allocate();
  image->FillBuffer(255);

  itk::ImageRegionIterator<UnsignedCharImageType> imageIterator(image, region);

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 20 && imageIterator.GetIndex()[0] < 50 &&␣
↪imageIterator.GetIndex()[1] > 20 &&
        imageIterator.GetIndex()[1] < 50)
      imageIterator.Set(50);

    ++imageIterator;
  }

  imageIterator.GoToBegin();

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 60 && imageIterator.GetIndex()[0] < 80 &&␣
↪imageIterator.GetIndex()[1] > 60 &&
```

(continues on next page)

(continued from previous page)

```
        imageIterator.GetIndex()[1] < 80)
      imageIterator.Set(100);

    ++imageIterator;
  }

  imageIterator.GoToBegin();

  while (!imageIterator.IsAtEnd())
  {
    if (imageIterator.GetIndex()[0] > 100 && imageIterator.GetIndex()[0] < 130 &&␣
↪imageIterator.GetIndex()[1] > 100 &&
        imageIterator.GetIndex()[1] < 130)
      imageIterator.Set(150);

    ++imageIterator;
  }

  using FileWriterType = itk::ImageFileWriter<UnsignedCharImageType>;
  FileWriterType::Pointer writer = FileWriterType::New();
  writer->SetFileName("input.png");
  writer->SetInput(image);
  writer->Update();
}

void
PerformSegmentation(FloatImageType::Pointer image, const float threshold, const float␣
↪level)
{
  using MorphologicalWatershedFilterType = itk::MorphologicalWatershedImageFilter
↪<FloatImageType, LabeledImageType>;
  MorphologicalWatershedFilterType::Pointer watershedFilter =␣
↪MorphologicalWatershedFilterType::New();
  watershedFilter->SetLevel(level);
  watershedFilter->SetInput(image);
  watershedFilter->Update();

  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<LabeledImageType,␣
↪RGBImageType>;
  RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
  colormapImageFilter->SetInput(watershedFilter->GetOutput());
  colormapImageFilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::
↪RGBColormapFilter::Jet);
  colormapImageFilter->Update();

  std::stringstream ss;
  ss << "output_" << threshold << "_" << level << ".png";

  using FileWriterType = itk::ImageFileWriter<RGBImageType>;
  FileWriterType::Pointer writer = FileWriterType::New();
  writer->SetFileName(ss.str());
  writer->SetInput(colormapImageFilter->GetOutput());
  writer->Update();
}
```

### Classes demonstrated

template<typename **TInputImage**, typename **TOutputImage**>
**class MorphologicalWatershedImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *TOutputImage*>
Watershed segmentation implementation with morphological operators.

Watershed pixel are labeled 0. TOutputImage should be an integer type. Labels of output image are in no particular order. You can reorder the labels such that object labels are consecutive and sorted based on object size by passing the output of this filter to a RelabelComponentImageFilter.

The morphological watershed transform algorithm is described in Chapter 9.2 of Pierre Soille's book "Morphological Image Analysis:

Principles and Applications", Second Edition, Springer, 2003.

This code was contributed in the Insight Journal paper: "The watershed transform in ITK - discussion and new developments" by Beare R., Lehmann G. https://www.insight-journal.org/browse/publication/92

**Author** Gaetan Lehmann. Biologie du Developpement et de la Reproduction, INRA de Jouy-en-Josas, France.

**See** WatershedImageFilter, MorphologicalWatershedFromMarkersImageFilter

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Morphological Watershed Segmentation

See itk::MorphologicalWatershedImageFilter for additional documentation.

### Watershed Image Filter

### Synopsis

This example illustrates how to segment an image using the watershed method.

### Results

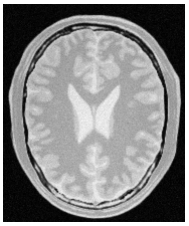


Fig. 383: Input image

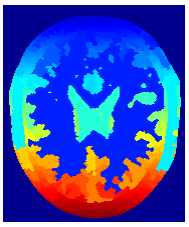


Fig. 384: Segmented image

### Code

### C++

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkScalarToRGBPixelFunctor.h"
#include "itkUnaryFunctorImageFilter.h"
#include "itkVectorGradientAnisotropicDiffusionImageFilter.h"
#include "itkWatershedImageFilter.h"
#include "itkRescaleIntensityImageFilter.h"
#include "itkScalarToRGBColormapImageFilter.h"
#include "itkGradientMagnitudeImageFilter.h"

// Run with:
// ./SegmentWithWatershedImageFilter inputImageFile outputImageFile threshold level
// e.g.
// ./SegmentWithWatershedImageFilter BrainProtonDensitySlice.png OutBrainWatershed.
↪png 0.005 .5
// (A rule of thumb is to set the Threshold to be about 1 / 100 of the Level.)

int
main(int argc, char * argv[])
{
  if (argc < 5)
  {
    std::cerr << "Missing parameters." << std::endl;
    std::cerr << "Usage: " << argv[0] << " inputImageFile outputImageFile threshold␣
↪level" << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;

  using InputImageType = itk::Image<unsigned char, Dimension>;
  using FloatImageType = itk::Image<float, Dimension>;
  using RGBPixelType = itk::RGBPixel<unsigned char>;
  using RGBImageType = itk::Image<RGBPixelType, Dimension>;
  using LabeledImageType = itk::Image<itk::IdentifierType, Dimension>;

  using FileReaderType = itk::ImageFileReader<InputImageType>;
  FileReaderType::Pointer reader = FileReaderType::New();
  reader->SetFileName(argv[1]);

  using GradientMagnitudeImageFilterType = itk::GradientMagnitudeImageFilter
↪<InputImageType, FloatImageType>;
```

(continues on next page)

(continued from previous page)

```
  GradientMagnitudeImageFilterType::Pointer gradientMagnitudeImageFilter =␣
↪GradientMagnitudeImageFilterType::New();

  gradientMagnitudeImageFilter->SetInput(reader->GetOutput());
  gradientMagnitudeImageFilter->Update();

  using WatershedFilterType = itk::WatershedImageFilter<FloatImageType>;
  WatershedFilterType::Pointer watershed = WatershedFilterType::New();

  float threshold = std::stod(argv[3]);
  float level = std::stod(argv[4]);

  watershed->SetThreshold(threshold);
  watershed->SetLevel(level);

  watershed->SetInput(gradientMagnitudeImageFilter->GetOutput());
  watershed->Update();

  using RGBFilterType = itk::ScalarToRGBColormapImageFilter<LabeledImageType,␣
↪RGBImageType>;
  RGBFilterType::Pointer colormapImageFilter = RGBFilterType::New();
  colormapImageFilter->SetColormap(itk::ScalarToRGBColormapImageFilterEnums::
↪RGBColormapFilter::Jet);
  colormapImageFilter->SetInput(watershed->GetOutput());
  colormapImageFilter->Update();

  using FileWriterType = itk::ImageFileWriter<RGBImageType>;
  FileWriterType::Pointer writer = FileWriterType::New();
  writer->SetFileName(argv[2]);
  writer->SetInput(colormapImageFilter->GetOutput());
  writer->Update();

  return EXIT_SUCCESS;
}
```

### Python

```
#!/usr/bin/env python

# Run with:
# ./WatershedImageFilter.py <InputFileName> <OutputFileName> <Threshold> <Level>
# e.g.
# ./WatershedImageFilter.py BrainProtonDensitySlice.png OutBrainWatershed.png 0.005 .5
# (A rule of thumb is to set the Threshold to be about 1 / 100 of the Level.)
#
#  threshold: absolute minimum height value used during processing.
#        Raising this threshold percentage effectively decreases the number of local␣
↪minima in the input,
#        resulting in an initial segmentation with fewer regions.
#        The assumption is that the shallow regions that thresholding removes are of␣
↪of less interest.
#  level: parameter controls the depth of metaphorical flooding of the image.
#        That is, it sets the maximum saliency value of interest in the result.
#        Raising and lowering the Level influences the number of segments
```

(continues on next page)

(continued from previous page)

```
#         in the basic segmentation that are merged to produce the final output.
#         A level of 1.0 is analogous to flooding the image up to a
#         depth that is 100 percent of the maximum value in the image.
#         A level of 0.0 produces the basic segmentation, which will typically be very␣
↪oversegmented.
#         Level values of interest are typically low (i.e. less than about 0.40 or 40
↪%),
#         since higher values quickly start to undersegment the image.

import itk
import argparse

parser = argparse.ArgumentParser(description="Segment With Watershed Image Filter.")
parser.add_argument("input_image")
parser.add_argument("output_image")
parser.add_argument("threshold", type=float)
parser.add_argument("level", type=float)
args = parser.parse_args()


Dimension = 2

FloatPixelType = itk.ctype("float")
FloatImageType = itk.Image[FloatPixelType, Dimension]

reader = itk.ImageFileReader[FloatImageType].New()
reader.SetFileName(args.input_image)

gradientMagnitude = itk.GradientMagnitudeImageFilter.New(Input=reader.GetOutput())

watershed = itk.WatershedImageFilter.New(Input=gradientMagnitude.GetOutput())

threshold = args.threshold
level = args.level
watershed.SetThreshold(threshold)
watershed.SetLevel(level)

LabeledImageType = type(watershed.GetOutput())

PixelType = itk.ctype("unsigned char")
RGBPixelType = itk.RGBPixel[PixelType]
RGBImageType = itk.Image[RGBPixelType, Dimension]

ScalarToRGBColormapFilterType = itk.ScalarToRGBColormapImageFilter[
    LabeledImageType, RGBImageType
]
colormapImageFilter = ScalarToRGBColormapFilterType.New()
colormapImageFilter.SetColormap(
    itk.ScalarToRGBColormapImageFilterEnums.RGBColormapFilter_Jet
)
colormapImageFilter.SetInput(watershed.GetOutput())
colormapImageFilter.Update()

WriterType = itk.ImageFileWriter[RGBImageType]
writer = WriterType.New()
writer.SetFileName(args.output_image)
writer.SetInput(colormapImageFilter.GetOutput())
```

(continues on next page)

(continued from previous page)

```
writer.Update()
```

## Classes demonstrated

template<typename **TInputImage**>
**class WatershedImageFilter** : **public** itk::*ImageToImageFilter*<*TInputImage*, *Image*<IdentifierType, *TInputImage*::ImageD

A low-level image analysis algorithm that automatically produces a hierarchy of segmented, labeled images from a scalar-valued image input.

**Overview and terminology**

This filter implements a non-streaming version of an image segmentation algorithm commonly known as "watershed segmentation". Watershed segmentation gets its name from the manner in which the algorithm segments regions into catchment basins. If a function $f$ is a continuous height function defined over an image domain, then a catchment basin is defined as the set of points whose paths of steepest descent terminate at the same local minimum of $f$.

The choice of height function (input) depends on the application, and the basic watershed algorithm operates independently of that choice. For intensity-based image data, you might typically use some sort of gradient magnitude calculation as input. (see itk::GradientMagnitudeImageFilter)

The watershed algorithm proceeds in several steps. First, an initial classification of all points into catchment basin regions is done by tracing each point down its path of steepest descent to a local minima. Next, neighboring regions and the boundaries between them are analyzed according to some saliency measure (such as minimum boundary height) to produce a tree of merges among adjacent regions. These merges occur at different maximum saliency values. The collective set of all possible merges up to a specified saliency "flood level" is referred to in this documentation as a "merge tree". Metaphorically, the flood level is a value that reflects the amount of precipitation that is rained into the catchment basins. As the flood level rises, boundaries between adjacent segments erode and those segments merge. The minimum value of the flood level is zero and the maximum value is the difference between the highest and lowest values in the input image.

Note that once the initial analysis and segmentation is done to produce the merge tree, it is trivial to produce a hierarchy of labeled images in constant time. The complexity of the algorithm is in the computation of the merge tree. Once that tree has been created, the initial segmented image can be relabeled to reflect any maximum saliency value found in the tree by simply identifying a subset of segment merges from the tree.

**Implementational details** This filter is a wrapper for several lower level process objects (watershed algorithm components in the namespace "watershed"). For a more complete picture of the implementation, refer to the documentation of those components. The component classes were designed to operate in either a data-streaming or a non-data-streaming mode. The pipeline constructed in this class' GenerateData() method does not support streaming, but is the common use case for the components.

**Description of the input to this filter** The input to this filter is a scalar itk::Image of any dimensionality. This input image is assumed to represent some sort of height function or edge map based on the original image that you want to segment (such as would be produced by itk::GradientMagnitudeImageFilter). This filter does not do any pre-processing on its input other than a thresholding step. The algorithm does not explicitly require that the input be of any particular data type, but floating point or double precision data is recommended.

The recommended pre-processing for scalar image input to this algorithm is to use one of the itk::AnisotropicDiffusionImageFilter subclasses to smooth the original image and then perform some sort of edge calculation based on gradients or curvature.

**Description of the output of this filter** This filter will produce an itk::Image of IdentifierType integer type and of the same dimensionality as the input image. The IdentifierType output image is referred to as the "labeled image" in this documentation. Each pixel in the image is assigned an IdentifierType integer label that groups it within a connected region.

**Some notes on filter parameters** Two parameters control the output of this filter, Threshold and Level. The units of both parameters are percentage points of the maximum height value in the input.

Threshold is used to set the absolute minimum height value used during processing. Raising this threshold percentage effectively decreases the number of local minima in the input, resulting in an initial segmentation with fewer regions. The assumption is that the shallow regions that thresholding removes are of of less interest.

The Level parameter controls the depth of metaphorical flooding of the image. That is, it sets the maximum saliency value of interest in the result. Raising and lowering the Level influences the number of segments in the basic segmentation that are merged to produce the final output. A level of 1.0 is analogous to flooding the image up to a depth that is 100 percent of the maximum value in the image. A level of 0.0 produces the basic segmentation, which will typically be very oversegmented. Level values of interest are typically low (i.e. less than about 0.40 or 40% ), since higher values quickly start to undersegment the image.

The Level parameter can be used to create a hierarchy of output images in constant time once an initial segmentation is done. A typical scenario might go like this: For the initial execution of the filter, set the Level to the maximum saliency value that you anticipate might be of interest. Once the initial Update() of this process object has finished, the Level can be manipulated anywhere below the initial setting without triggering a full update of the segmentation mini-pipeline. All that is now be required to produce the new output is a simple relabeling of the output image.

Threshold and Level parameters are controlled through the class' Get/SetThreshold() and Get/SetLevel() methods.

**ITK Sphinx Examples:**

- All ITK Sphinx Examples
- Watershed Image Filter

See itk::WatershedImageFilter for additional documentation.

## 3.11 Video

### 3.11.1 BridgeOpenCV

#### Convert an ITK Gray Scale Image to CV Mat

#### Synopsis

Convert an itk::Image (gray scale) to OpenCV cv::Mat.

### Results

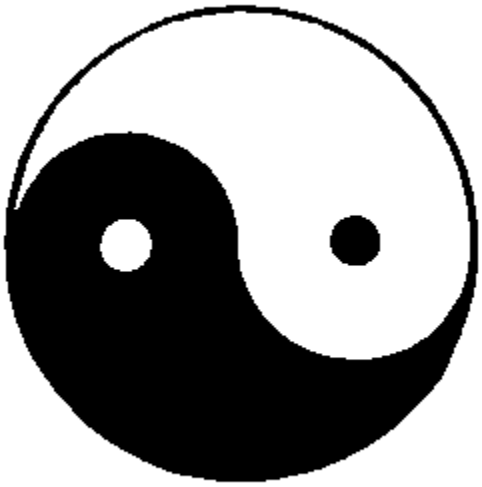


Fig. 385: Input image

### Code

### C++

```
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkOpenCVImageBridge.h"

// includes from OpenCV
#include "opencv2/opencv.hpp" // cv::imwrite


int
main(int argc, char * argv[])
{
  if (argc != 3)
  {
    std::cerr << "Usage: " << std::endl;
    std::cerr << argv[0];
    std::cerr << "<InputFileName> <OutputFileName>";
    std::cerr << std::endl;
    return EXIT_FAILURE;
  }

  constexpr unsigned int Dimension = 2;
```

(continues on next page)

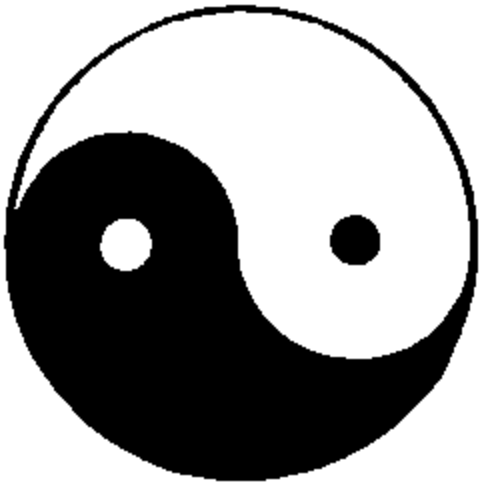


Fig. 386: Output image

(continued from previous page)

```
  using PixelType = unsigned char;
  using ImageType = itk::Image<PixelType, Dimension>;

  using ReaderType = itk::ImageFileReader<ImageType>;
  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName(argv[1]);

  try
  {
    reader->Update();
  }
  catch (itk::ExceptionObject & error)
  {
    std::cerr << "Error: " << error << std::endl;
    return EXIT_FAILURE;
  }

  cv::Mat img = itk::OpenCVImageBridge::ITKImageToCVMat<ImageType>(reader->
↪GetOutput());

  cv::imwrite(argv[2], img);

  return EXIT_SUCCESS;
}
```

### Classes demonstrated

**class OpenCVImageBridge**

This class provides static methods to convert between OpenCV images and itk::Image.

This class provides methods for the following conversions: IplImage -> itk::Image cv::Mat -> itk::Image itk::Image -> IplImage itk::Image -> cv::Mat

Each method is templated over the type of itk::Image used. The conversions copy the data and convert between types if necessary.

See itk::OpenCVImageBridge for additional documentation.

CHAPTER

FOUR

# DOWNLOAD

## 4.1 Archives in various formats

The complete set of examples can be downloaded in various formats:

- HTML (Packed as .tar.gz)
- HTML (Packed as .zip)
- PDF
- EPUB

## 4.2 Individual examples

To download the code, data, and documentation for an individual example, click the **Download** button in the navbar of an example's webpage.

Once an example tarball has been downloaded and unpacked, follow the additional documentation on *how to build and run an example*.

## 4.3 Source repository

The entire source tree can be downloaded with Git:

```
git clone --recursive https://github.com/InsightSoftwareConsortium/ITKSphinxExamples.
↪git
```

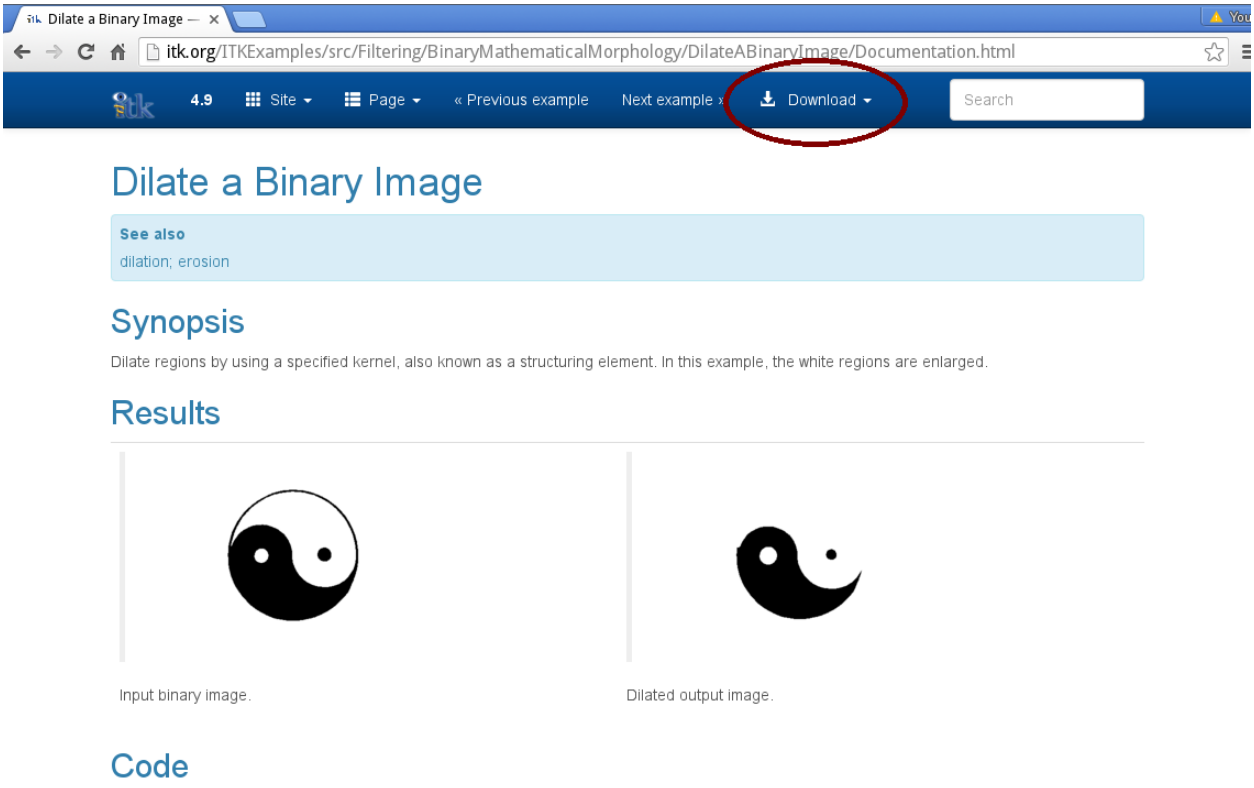



Fig. 1: Links to download individual examples.

CHAPTER

FIVE

# CREDITS

CHAPTER

SIX

# INDEX

- genindex

# INDEX