Non-rigid Image Registration in ITK

Insightful Corporation
University of Pennsylvania
Insight Software Consortium
Special Considerations in Non-rigid Image Registration

- Very high dimensional problem
- More efficient to directly incorporate optimization into implementation
  - Still will be computationally intense
- “Locality” property of similarity measure
  - Sensitive to very localized differences yet exhibits specificity and robustness
General Approaches

- Two approaches
  - Reflect dual formulations
  - Implications for numerical implementation

- Differential methods
  - Gradient based optical flow formulations
  - ITK implements Thirion’s Demons method

- Variational methods
  - Energy minimizing formulations
  - ITK implements an optimization of the objective function
    \[ \text{cost} = \text{deformation} - \text{similarity} \]
Duality of Approaches: Gradient based Optical Flow

- Assume intensity conservation over time

\[ I(x, y, t) = I(x + u\delta t, y + v\delta t, t + \delta t) \]

- Implies the total temporal derivative of the image function should be zero

\[ 0 = \frac{dI}{dt} = \frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} \]

\[ = \vec{I}_{,x} \cdot \vec{v} + \vec{I}_{,t} \]
Duality of Approaches: Gradient based Optical Flow

- Minimize intensity conservation term
  \[ E_D(\vec{v}) = \int \rho(\vec{I}_x \cdot \vec{v} + I_t) dx \]
  \[ \rho(x) = x^2 \quad \text{(Horn and Schunck)} \]

- Regularize problem by penalizing smoothness of the velocity field
  \[ E_S(\vec{v}) = \int \| \vec{v}_x \|^2 dx \]
Duality of Approaches: Classical Elastic Matching

- Find registration transformation that maximizes
  \[ E(\tilde{u}) = \int \text{similarity}(I_t(\bar{x}), I_{t+\delta t}(\bar{x} + \tilde{u})) \]
  \[ -\int \text{smoothness}(\tilde{u}) \]
  \[ \uparrow \]
  \[ -\int \text{deformation}(\tilde{u}) \]

- Deformation = linear elastic internal strain energy
Duality of Approaches: Classical Elastic Matching

- The gradient of the similarity (potential) yields the external load with which one image is deformed to assume the appearance of the second image.
  - Similarity (potential) is implemented with ITK Image Metrics

- Elastostatic configurations, which optimize $E$, represent solutions to the corresponding elastic matching problem.
Duality of Approaches: Classical Elastic Matching

- Point-wise equilibrium equations are derived by applying the principle of balance of linear momentum:

\[ \mu \nabla^2 \vec{u} + (\lambda + \mu) \nabla (\nabla \cdot \vec{u}) + \vec{b} = 0 \]

(Navier’s displacement equations of equilibrium)
Outline

- Overview of differential approach to registration
- Introduction to Thirion’s Demons method
- Discussion of registration parameters
- Code walk-through
- Examples
Differential Approach to Non-rigid Image Registration

- **Benefits**
  - Easy to code
  - Fast implementations

- **Limitations**
  - Difficult to analyze and debug
  - Implementation restricted to uniform grids
Thirion’s Demons Algorithm

\[ \mathbf{v} = \frac{-I_{,t} \mathbf{I}_{,x}}{\left| \mathbf{I}_{,x} \right|^2 + I_{,t}^2} \]

- Induce smoothness of deformation field by periodically convolving with Gaussian filter
- Intensity based so may require histogram matching beforehand
- Implemented in ITK within the Finite Difference Framework
Finite Difference Framework

- Hierarchy of classes for solving Finite Difference problems
  - Subclass for Partial Differential Equations.

- Iterates through the domain making finite difference updates to the solution.
Image Registration with Finite Difference Framework

The update function adds to the current vector field at position vector $x$.

$$V(x) + = U(x)$$

The update vector is in the direction of incrementally improving similarity.
itkDemonsRegistrationFunction implements the traditional Demons update:

// Get fixed image information
fixedValue = (double) m_FixedImage->GetPixel( index );
fixedGradient = m_FixedImageGradientCalculator->EvaluateAtIndex( index );
fixedGradientSquaredMagnitude += vnl_math_sqr( fixedGradient[j] )

// Get moving image related information
mappedPoint[j] = double( index ) * m_FixedImageSpacing + m_FixedImageOrigin;
mappedPoint[j] += it.GetCenterPixel()[j];
movingValue = m_MovingImageInterpolator->Evaluate( mappedPoint );
// Compute update
double speedValue = fixedValue - movingValue;

denominator =
  vnl_math_sqr(speedValue) +
  fixedGradientSquaredMagnitude;

Update = speedValue * fixedGradient / denominator;

FD Registration Variations

- Multi-resolution framework.

- PDEDeformableRegistration smooths the field with a Gaussian filter. May also use other smoothing, e.g. Anisotropic diffusion.

- One may implement an update function with:
  - A different Neighborhood radius, for use with, e.g. Normalized Correlation.
Demons Registration: Code Overview

..\Insight\Examples\Registration\DeformableRegistration2.cxx
**Header Files**

- **Step 1: Include the header files**

```c++
#include "itkDemonsRegistrationFilter.h"
#include "itkHistogramMatchingImageFilter.h"
#include "itkCastImageFilter.h"
#include "itkWarpImageFilter.h"
#include "itkLinearInterpolateImageFunction.h"
```
Step 2: Declare the types of the images

```cpp
const unsigned int Dimension = 2;
typedef unsigned short PixelType;

typedef itk::Image<PixelType, Dimension> FixedImageType;

typedef itk::Image<PixelType, Dimension> MovingImageType;
```
Step 3a: Declare an internal image type and corresponding filter types to cast the images

typedef float InternalPixelType;
typedef itk::Image<InternalPixelType, Dimension> InternalImageType;

typedef itk::CastImageFilter<FixedImageType, InternalImageType> FixedImageCasterType;
typedef itk::CastImageFilter<MovingImageType, InternalImageType> MovingImageCasterType;
Cast Image Types #2

- **Step 3b: Cast the input images to the internal image type**

```cpp
FixedImageCasterType::Pointer fixedImageCaster=FixedImageCasterType::New();
MovingImageCasterType::Pointer movingImageCaster=MovingImageCasterType::New();

fixedImageCaster->SetInput(fixedImageReader->GetOutput());
movingImageCaster->SetInput(movingImageReader->GetOutput());
```
Image Preprocessing #1

- **Step 4a: Declare a histogram-matching filter**

```cpp
typedef itk::HistogramMatchingImageFilter<
    InternalImageType,
    InternalImageType> MatchingFilterType;

MatchingFilterType::Pointer matcher =
    MatchingFilterType::New();
```
**Image Preprocessing #2**

- **Step 4b:** Configure the filter to match the intensity of the moving and fixed images

```cpp
matcher->
    SetInput(movingImageCaster->GetOutput());
matcher->
    SetReferenceImage(fixedImageCaster->GetOutput());
matcher->SetNumberOfHistogramLevels( 1024 );
matcher->SetNumberOfMatchPoints( 7 );
matcher->ThresholdAtMeanIntensityOn();
```
Demons Registration Filter

- **Step 5**: Represent the deformation field as an image whose pixels are floating-point vectors

```cpp
typedef itk::Vector<float, Dimension> VectorPixelType;
typedef itk::Image<VectorPixelType, Dimension> DeformationFieldType;
typedef itk::DemonsRegistrationFilter<
    InternalImageType, InternalImageType,
    DeformationFieldType>
    RegistrationFilterType;
```
Demons Filter Settings

- Step 6: Configure the Demons filter and run the registration

RegistrationFilterType::Pointer filter = RegistrationFilterType::New();

filter->SetFixedImage(fixedImageCaster->GetOutput());
filter->SetMovingImage(matcher->GetOutput());
filter->SetNumberOfIterations(150);
filter->SetStandardDeviations(1.0);
filter->Update();
Applying the Deformation

- Step 7: Configure a warping filter to apply the resulting deformation field to the moving image

```cpp
typedef itk::WarpImageFilter<
    MovingImageType,
    MovingImageType,
    DeformationFieldType> WarperType;

typedef itk::LinearInterpolateImageFunction<
    MovingImageType, double> InterpolatorType;
```
Registration Results

- Step 8: Warp the moving image and output the result

```cpp
WarperType::Pointer warper = WarperType::New();
InterpolatorType::Pointer interpolator = InterpolatorType::New();
FixedImageType::Pointer fixedImage = fixedImageReader->GetOutput();
warper->SetInput(movingImageReader->GetOutput());
warper->SetInterpolator(interpolator);
warper->SetOutputSpacing(fixedImage->GetSpacing());
warper->SetOutputOrigin(fixedImage->GetOrigin());
warper->SetDeformationField(filter->GetOutput());
```
Volumetric analysis of brain volume from MR images important tool for studying diseases.

Manual delineation:
- Requires substantial time and effort by trained personnel
- Suffers from large inter-observer variability and poor reproducibility

Automatic method using atlas-based matching
Segmentation via Atlas-based Registration

Atlas Image

Subject Image

Registration

Deformation Field

Segmentation

Atlas Mask

Warp

Automatic Subject Segmentation
IBSR Dataset

- Internet Brain Segmentation Repository
- Manually-guided expert segmentation results
- 20 Normal T1-weighted MR Volumes
  - 1mm x 1mm pixels
  - 3mm slice thickness
- One volume was chosen to be the atlas and the remaining for validation
Experimental Procedure

For each subject image
- Normalized intensity to between 0 and 1
- Histogram match of atlas to subject image
- Register the preprocessed atlas image to the normalized subject image using the demons algorithm
- Warp/deform atlas brain volume mask using output field to produce brain volume mask for subject
- Compare results with manually obtained masks provided by IBSR
Evaluation Metric

- Kappa statistic based similarity index

\[ S = \frac{2|A \cap B|}{(|A| + |B|)} \]

- Numerator = overlap between the two sets
- Denominator = mean volume
- Index takes into account both size and location of the overlap
The algorithm performance in terms of speed and accuracy was found to be comparable to other published results.
Outline

- Overview of variational approach to registration
- Introduction to the ITK finite element method (FEM) library
- Discussion of registration parameters
- Code walk-through
- Examples
- Advanced registration features
Overview of Variational-based Registration and its Finite Element Implementation
Variational Approach to Non-rigid Image Registration

Find registration transformation that maximizes

\[
E(\tilde{u}) = \int \text{similarity}(I_t(\bar{x}), I_{t+\delta t}(\bar{x} + \tilde{u})) \\
- \int \text{smoothness}(\tilde{u}) \\
\updownarrow \\
- \int \text{deformation}(\tilde{u})
\]
Variational Approach to Non-rigid Image Registration

- **Benefits**
  - Intuitive; easier to express constraints
  - Powerful numerical solutions available
  - Optimality of solutions; easier to debug

- **Limitations**
  - Difficult/overhead to implement
Finite Element Implementation

- To solve for the deformation, consider only displacements of the form

\[ u_h(x) = \alpha_i \varphi_i(s) \]

- Substitute \( u_h \) into \( E \), then minimizing with respect to \( \alpha_i \):

\[ \frac{\partial E}{\partial \alpha_i} = 0, \quad i = 1, \ldots, n \]
In FEM, $\varphi_i$ are defined piecewise according to subdivisions of the problem (image) domain (called finite elements), and calculations are made on an element by element basis.

- Elements are connected at discrete nodal points, at which the transformation (displacement) is solved.
- Efficiency gained by elemental computations.
- Domain subdivision (or mesh) can be tailored to the underlying geometry or structure of the image.
Variational-based Image Registration

Start Iteration Loop

Physical Assumptions

New Solution

Image Metric Derivative

Solve

End Iteration Loop

- Begin Loop by making physical assumptions and then taking the derivative of the similarity metric.
- End loop when the solution stabilizes.
FEM Implementation

- The iteration loop solves a linear system at each time step. A typical system may be:

\[ KU = F \]

- The linear system numerically captures the energy formulation associated with the physics:
  - K: positive-definite matrix
  - U: regularized solution vector
  - F: image-based forces
FEM Implementation

Physical Assumptions/Regularization

Start Iteration Loop

Image Metric Derivative

Solve

New Solution

$U_{\text{New}} = U_{\text{Old}} + U$

End Iteration Loop

$K$ $U$ $F$
FEM Numerics

\[ U_{NEW} = U_{OLD} + U \]

Start Iteration Loop

Recall, \[ KU = F \]

If \( (U_{NEW} - U_{OLD}) < \varepsilon \) then Stop
KU = F in Code

 itkFEMRegistrationFilter::IterativeSolve()

 FEMSolver::AssembleK()

 FEMSolver::AssembleF() calls FEMImageMetricLoad::Fe()

 FEMSolver::AddSolution()

 FEMSolver::Solve()
FEM-Based Registration Options

- **Element type**
  - Triangles, quadrilaterals, hexahedra, tetrahedra

- **Continuum / physical model**
  - Linear elasticity, membrane, other specialized

- **Mesh geometry**
  - Uniform grid vs. adaptive, anatomy-specific mesh

- **Metric**
  - Mean square, normalized cross-correlation, mutual information, pattern intensity

- **Multi-resolution strategy**
Introduction to the ITK Finite Element Library
Overview

- Library for solving general FEM problems
  - Object oriented
  - C++ classes are used to
    - specify the geometry and behavior of the elements
    - apply external forces and boundary conditions
    - solve problem and post-process the results

Applications
- Mechanical modeling
- Image registration
FEM Basics

- **Mesh**
  - **Nodes**
    - Points in space where solutions are obtained
  - **Elements**
    - e.g., 2-D triangular elements
- **Loads**
  - e.g., gravity (body) load
- **Boundary conditions**
  - e.g., nodes fixed in space
Elements

- Core of the library is the Element class
  - Code is in two functionally independent parts
    - Geometry and Physics
    - Arbitrarily combined to create new elements
- Problem domain is specified by a mesh
Loads

- Classes that apply external forces (loads) to elements
  - Various types
  - Easily extensible
Solvers

- Provide functionality to obtain and process the solution
- Different solution methods $\rightarrow$ different solver classes
  - Static problems
  - Time dependent - dynamic problems
- Use linear system wrappers to link FEM classes to an external numeric library
  - Any numeric library can be used to solve the systems of linear equations in FEM problems
  - VNL and ITPACK currently supported
Setting Up A FEM Problem

- Four-step process
  - Select element classes
  - Discretize problem domain
  - Specify boundary conditions
  - Specify/Apply external loads
- Two options
  - Directly \(\rightarrow\) create proper objects in code
  - Indirectly \(\rightarrow\) read object definitions from a file
Dynamic Elasticity Example #1

- Vibration of bridge under point load
  - Bridge is composed of 1-D Bar elements
  - Left node is fixed in x, y and right only in y
  - Nodal load is applied at the middle point at \( t=0 \)
Dynamic Elasticity Example

#2

- **Elastic square**
  - Composed of 2-D triangular elements
  - Entire left side of the square is fixed
  - Uniform gravity load is applied on all elements at $t=0$
Dynamic Elasticity Example

#3

- Elastic cube
  - Composed of 3-D hexahedral elements
  - Base of the cube is fixed in the xy-plane
  - Uniform gravity load applied in –z direction at t=0
FEM-Based Registration: Parameters
% Parameters for the single- or multi-resolution techniques
% ---------------------------------------------------------
1 % Number of levels in the multi-resolution pyramid (1 = single-res)
1 % Highest level to use in the pyramid
1 1 % Scaling at lowest level for each image dimension
8 % Number of pixels per element
1.e5 % Elasticity (E)
1.e4 % Density (RhoC)
1. % Image energy scaling
4 % NumberOfIntegrationPoints
1 % WidthOfMetricRegion
25 % MaximumIterations

% Parameters for the registration
% --------------------------------
0 1.0 % Similarity metric (0=mean sq, 1=ncc, 2=pattern int, 3=MI)
1.0 % Alpha
1 % DescentDirection
2 % DoLineSearch (0=never, 1=always, 2=if needed)
1.e1 % TimeStep
1.e-15 % Energy Reduction Factor
% Information about the image inputs
% ----------------------------------
2                    % ImageDimension
256                  % Nx (image x dimension)
256                  % Ny (image y dimension)
128                  % Nz (image z dimension - not used if 2D)
brain_slice1.mhd     % ReferenceFileName
brain_slice1warp.mhd  % TargetFileName

% The actions below depend on the values of the flags preceding them.
% For example, to write out the displacement fields, you have to set
% the value of WriteDisplacementField to 1.
% -------------------------------------------------------------------
0                    % UseLandmarks?
-                     % LandmarkFileName
brain_result         % ResultsFileName (prefix only)
1                     % WriteDisplacementField?
brainDisp            % DisplacementsFileName (prefix only)
1                     % ReadMeshFile?
brain_mesh.fem       % MeshFileName

END
this->DoMultiRes(true);

this->m_NumLevels = nlev;
this->m_MaxLevel = mlev;

for (jj=0; jj < ImageDimension; jj++) {
    m_ImageScaling[jj] = dim;
}

for (jj=0; jj < this->m_NumLevels; jj++) {
    this->m_MeshPixelsPerElementAtEachResolution(jj) = p;
    this->SetElasticity(e, jj);
    this->SetRho(p, jj);
    this->SetGamma(g, jj);
    this->SetNumberOfIntegrationPoints(ip, jj);
    this->SetWidthOfMetricRegion(w, jj);
    this->SetMaximumIterations(mit, jj);
}
this->SetDescentDirectionMinimize();

    or

this->SetDescentDirectionMaximize();

this->DoLineSearch(n);  // n = 0, 1, 2

this->SetTimeStep(t);

this->SetEnergyReductionFactor(fbuf);
Configuring Parameters #3

this->m_ImageSize[0] = xdim;
this->m_ImageSize[1] = ydim;
if (dim == 3) this->m_ImageSize[2] = zdim;

this->SetReferenceFile(imgfile1);
this->SetTargetFile(imgfile2);

this->UseLandmarks(true);
this->SetLandmarkFile(lmfile);

this->SetResultsFile(resfile);
this->SetWriteDisplacements(true);
this->SetDisplacementsFile(dispfile);

this->m_ReadMeshFile=true;
this->m_MeshFileName=meshfile;
FEM-Based Registration: Writing the Code

..\Insight\Examples\Registration\DeformableRegistration1.cxx
Header Declarations

#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"

#include "itkFEM.h"
#include "itkFEMRegistrationFilter.h"
typedef itk::Image<unsigned char, 2> fileImageType;
typedef itk::Image<float, 2> ImageType;

typedef
    itk::fem::Element2DC0LinearQuadrilateralMembraneElementType;

typedef
    itk::fem::Element2DC0LinearTriangularMembraneElementType2;
typedef itk::fem::ImageMetricLoad<ImageType, ImageType> ImageLoadType;

template class itk::fem::ImageMetricLoadImplementation<ImageLoadType>;

typedef ElementType::LoadImplementationFunctionPointer LoadImpFP;

typedef ElementType::LoadType ElementLoadType;

typedef itk::fem::VisitorDispatcher<ElementType, ElementLoadType, LoadImpFP> DispatcherType;
typedef itk::fem::FEMRegistrationFilter<
    ImageType, ImageType>
    RegistrationType;
Registering Objects

```cpp
ElementType::LoadImplementationFunctionPointer fp =
&itk::fem::ImageMetricLoadImplementation<ImageLoadType>::ImplementationImageMetricLoad;
DispatcherType::RegisterVisitor((ImageLoadType*)0,fp);
// Software Guide : EndCodeSnippet
```

```cpp
ElementType2::LoadImplementationFunctionPointer fp =
&itk::fem::ImageMetricLoadImplementation<ImageLoadType>::ImplementationImageMetricLoad;
DispatcherType2::RegisterVisitor((ImageLoadType*)0,fp);
```

RegistrationType::Pointer X = RegistrationType::New();

X->SetConfigFileName(paramname);

X->ReadConfigFile();
Material and Element Setup

// Create the material properties
itk::fem::MaterialLinearElasticity::Pointer m;
m = itk::fem::MaterialLinearElasticity::New();
m->GN = 0;
m->E = X->GetElasticity();
m->A = 1.0; // Cross-sectional area
m->h = 1.0; // Thickness
m->I = 1.0; // Moment of inertia
m->nu = 0.; // Poisson's ratio
m->RhoC = 1.0; // Density

// Create the element type
ElementType::Pointer e1=ElementType::New();
e1->m_mat=
    dynamic_cast<itk::fem::MaterialLinearElasticity*>(m);
X->SetElement(e1);
X->SetMaterial(m);
Running the Registration

X->RunRegistration();

X->WriteWarpedImage();

if (X->GetWriteDisplacements()) {
    X->WriteDisplacementField(0);
    X->WriteDisplacementField(1);
}
fe considerations (tessa, w/ aljaz’s help?)

- grid/mesh (animated examples)
- integration points
- inhomogeneous ‘material’ properties (previous FE animation examples?)
  - code snippet
- boundary conditions (previous FE animation examples)
- Run examples and show meshes used
- Boston examples for BCs
- 2-material lung meshes + definition of multiple materials
- No example of integration pts needed, just mention
FEM-Based 2D Non-Rigid Registration Examples

- Nonlinear problem
- Similarity – `itk::fem::ImageMetricLoad` with squared intensity difference metric
- Deformation – membrane element physical model
- Boundary conditions – one corner is fixed
- Triangular or quadrilateral elements
- Regular or irregular meshes
- Single- or multi-resolution strategies
- Iterative solver for dynamic problems
Regular Quadrilateral Mesh

- Undeformed
- Deformed
- Result

- 256 quadrilateral elements in a regular (uniform grid) mesh
- Run time: 10 seconds, PIII 750 MHz w/ 256 MB RAM
Adaptive Meshes

- Original segmented image
- We mesh the **fixed** image and compute the inverse warp
Sparse Triangular Mesh

- 586 elements
- Run time: 16 seconds
Dense Triangular Mesh

- 7640 elements
- Run time: 10 minutes
Adaptive Quadrilateral Mesh

- 1616 elements
- Run time: 2 minutes
Single-Material Model

- 2835 elements
Multi-Material Model

- 1202 elements
- Background is modeled as a very compliant material compared to the lung and body
- Loads over the background are intentionally nulled
advanced (tessa, w/ aljaz and brian’s help)

- element types?
  - show how penalty term is implemented?
- load types ↔ metrics
- landmarks (element or load type?)
- Probably not so much about implementation as what else you can do – changing metrics, using landmarks, etc. Maybe just a little bit about creating new elements (combine geometry and behavior, etc.)
FEM-Based Registration: Advanced Features
Creating Adaptive Meshes

- Uniform grids are automatically generated by the registration application.
- Anatomy-specific meshes can be created using public domain mesh generators.
- Must be converted to FEM input file format for use with ITK FEM library.
<Node>
  5       % Global object number
  2 33 4  % Node coordinates
</Node>

<Material Linear Elasticity>
  0       % Global object number
  E : 100  % Young modulus
  A : 1    % Beam cross section area
  I : 1    % Moment of inertia
  nu : 0.2 % Poisson's ratio
  h : 1    % Plate thickness
  RhoC : 1 % Density times capacity
END:      % End of material
ITK FEM Input File #2

<Element2DC0LinearTriangularMembrane>
  40      % Global object number
  375     % Node #1 ID
  376     % Node #2 ID
  180     % Node #3 ID
  0       % MaterialLinearElasticity ID

<LoadBC>
  3       % Global load number
  0       % GN of element
  2       % DOF# in element
  1 0     % rhs of MFC
Landmarks

- Constrain pointwise correspondence at specific locations in the image
- Requires additional parameter file

```
<LoadLandmark>
  -1 % Global load #
  -1 % GN of element
  2 75.5833 56.5833 % Undeformed landmark
  2 76.5000 56.1667 % Deformed landmark
  1.e-2 % Weight
```
Custom Element Types

- Design element with deformation penalty specific to your problem
- Combine existing geometry with new physical behavior specification
- Requires additional coding
Enjoy ITK!
TO DO

- Figure out the best slide order ✓
- Run the lung examples w/ zero backgrounds ✓
- Flip the brain images to face the right way up
- Screen captures of meshes that were used ✓
- Fill in all the blank slides ✓
- Think of what to say at each slide…
- Make sure the movies work ✓
follow from low-dimensional registration (luis)

differentiate above from non-rigid ⇔ high-dimensional registration (use animation example)
  
  more efficient to directly implement optimizer as part of problem (contrast low and hi dimen regr. frameworks)

  still will be computationally intense

  suitability of metric for non-rigid application

  “locality”, support region (see peter’s paper)
two approaches (reflect dual formulations, implications for numerical implementation)
  - differential -> optical flow
    - itk: demons
  - variational -> snakes, energy minimizing
    - itk: cost function = deformation - similarity
  - duality
    - optical flow ⇔ horn & schunk regularization is variational
    - snakes ⇔ differential implementation
    - original elastic matching ⇔ navier’s displacement equations of motion
differential approach \( (\text{me}) \)

- **Benefits**
  - easy to implement
  - fast algorithms

- **Limitations**
  - unintuitive
  - implementation restricted in practice to uniform grids
  - easy to adapt and thus to ‘abuse’
    - difficult to debug (no optimality property typically, except ones derived from variational formulations)
demons *(brian, sean?)*

- overview (use lydia’s notes?)
- finite difference implementation????
- code walk through (use lydia’s notes)
- evaluation studies?
fe-based registration (me)

 itk name?

- general problem statement
  - cost = deformation/penalty – similarity
    - note transformations are constrained by deformation/penalty to regularize the optimization problem, whereas in low-dim problem we explicitly restrict the solution space to a particular family of low-dimen transformations (few unknowns, many data samples)

- benefits
  - intuitive, easier to express constraints
  - powerful numerical solutions available
  - optimality of solutions, easier to debug

- limitations
  - difficult/overhead to implement

- overview modified registration framework??
elastic matching?? (me)

- deformation/penalty = linear elastic strain energy
  - hence ‘elastic’ matching
Variational-based Registration

- Registration is formulated as a variational problem in which a transformation is sought that maximizes the similarity between the pair of images subject to constraints on the transformation:

\[
\pi(\tilde{u}) = \int_{\text{source}} \text{similarity}(I_{\text{source}}(\tilde{x}), I_{\text{target}}(\tilde{x} + \tilde{u})) - \int_{\text{source}} \text{smoothness}(\tilde{u}) - \int_{\text{source}} \text{deformation}(\tilde{u})
\]
Variational-based Registration
Typical Formulation

- Typical constraint on transformation is that it models the deformative behavior of a continuum, such as an elastic body.
  - Deformation \( \equiv \) Internal strain energy

- The gradient of the similarity (potential) yields the external load with which one image is deformed to assume the appearance of the second image.
  - Similarity (potential) is implemented with ITK Image Metrics, e.g., mean square.
FEM-based Variational Image Registration

- User chooses
  - Mesh geometry
  - Continuum/Physics model
  - Metric
  - Multi-resolution strategy
Components for FEM-based Variational Image Registration

- **Finite Element Physical Model**
  - **Requests Force given Vector Field**
  - **Transform**
  - **Fixed Image**
  - **Interpolator**
  - **Moving Image**
  - **Metric Derivative**

FEM Image Metric Load
FEM-based 3D Registration Example

Fixed Image          Deforming Image
FEM Numerical Connection

\( K \) Positive definite matrix

\( U \) Regularized Solution Vector

\( F \) Image-related Force
FE-Based Registration in ITK

- Overview of deformation/regularization penalties implemented in ITK
  - Linear elasticity
  - Elastic membrane
  - Thin plate spline (possible but not impl.)
  - Membrane only with 2D and 3D elems for now
overview deformation/regularization penalties implemented in itk (list these)
Linear elasticity, membrane, others in 1D elements?
Restrictions on use with certain elements?
general fem introduction
  - need to cover this in order to make sense of code walk through
  - cover aspects that correspond to ‘design’ choices $\Leftrightarrow$ parameters for the particular method
  - highlight benefits
    - domain-specific grids
    - better numerical properties

Discuss to reflect any necessary parameter changes

Expand the old content…
itk fem library \textit{(tessa)}

- ‘complete’ general FEM library in ITK
  - not discussed but can use for mechanical modeling, etc
- overview (use previous slides?)
Linear System Wrappers

- Link FEM classes to an external numeric library
- Any numeric library can be used to solve the systems of linear equations in FEM problems
  - VNL and ITPACK currently supported
itk (brian, w/ tessa’s help?)
fe-based registration

- code walk through
- Include all headers/typedefs a la SoftwareGuide
- Step by step parameter definitions (ReadConfigFile)
- Parameter file at the end
Reverse Sparse Triangular Mesh

- 540 elements
- 16 iterations, 9 seconds
- RMS 19.8 (originally 41.1)
Reverse Dense Triangular Mesh

- 6825 elements
- 16 iterations, 585 seconds
- RMS 18.1 (originally 41.1)
Variational-Based Registration

Registration is formulated as a variational problem in which a transformation is sought that maximizes the similarity between the pair of images subject to constraints on the transformation:

\[
\pi(\tilde{u}) = \int_{\text{source}} \text{similarity}(I_{\text{source}}(\tilde{x}), I_{\text{target}}(\tilde{x} + \tilde{u})) \\
-\int_{\text{source}} \text{smoothness}(\tilde{u}) \\
\updownarrow \\
-\int_{\text{source}} \text{deformation}(\tilde{u})
\]
Typical Formulation

- Typical constraint on formulation is that it models the deformative behavior of a continuum, such as an elastic body.
  - Deformation $\equiv$ internal strain energy

- The gradient of the similarity (potential) yields the external load with which one image is deformed to assume the appearance of the second image.
  - Similarity (potential) is implemented with ITK Image Metrics