Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under contract DE-AC04-94AL85000.

ParaViewチュートリアル

バージョン3.8

Kenneth Moreland
Sandia National Laboratories
kmorel@sandia.gov

訳 / Japanese translators

大嶋拓也
Takuya OSHIMA
oshima@eng.niigata-u.ac.jp

荻野佳
Kei OGINO
ogino@env.arch.t.u-tokyo.ac.jp

田中秀和
Hidekazu TANAKA
tanaka-h@env.arch.t.u-tokyo.ac.jp

今野雅
Masashi IMANO
imano@arch.t.u-tokyo.ac.jp

目 次

演習の一覧

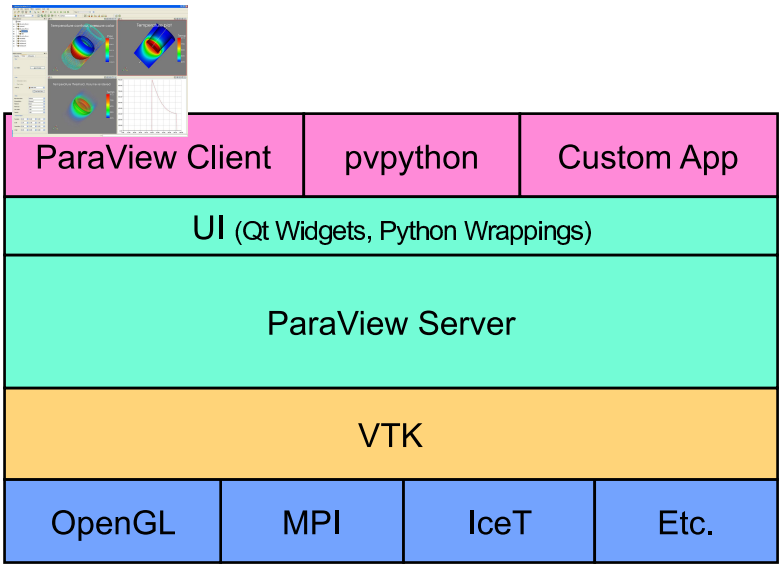
第1章　序

ParaView は、2次元および3次元データセットを可視化するためのオープンソースアプリケーションです。ParaView が扱えるデータセットの大きさは、ParaView が実行されるプラットフォームのアーキテクチャによって大きく変わります。ParaView によってサポートされるプラットフォームは、シングルプロセッサのワークステーションから、多数のプロセッサによって構成される分散メモリスーパーコンピュータやワークステーションクラスタまで、多岐に渡ります。並列コンピュータを利用することで、ParaView は巨大なデータセットを並列に処理し、その後で処理結果を集約することができます。今までのところサンディア国立研究所では、最大で60億セルの構造化メッシュ、2億5千万セルの非構造メッシュ、そして数十億のセルからなる構造 AMR 格子の可視化に ParaView を使用しています。

ParaView の設計においては、以下のような多くの特徴により、他の科学データ可視化ソリューションとの差別化を図っています。

- オープンソース、スケーラブル、かつマルチプラットフォームな可視化アプリケーション。
- 大容量データセットを処理するための分散型計算手法のサポート。
- オープン、柔軟かつ直感的なユーザインターフェイス。
- オープンな規格に基づいた拡張性の高いモジュール化構造。
- 有償の保守およびサポート。

ParaView は世界中の多くの学術、公的、および商業機関によって利用されており、毎月およそ3千件ダウンロードされています。



ParaView としてユーザから見えるアプリケーションの実体は、ParaView 自体の機能を構成する何層ものライブラリ群の上に構築された、小さなクライアントに過ぎません。上の図に示すように、ParaView のほとんどの機能はライブラリとして実装されているため、以下の図に示すように[1]ParaView の GUI をカスタムアプリケーションによって完全に置き換えることが可能です。さらに、ParaView に付属する **pvpython** アプリケーションによって、Python スクリプトを用いて可視化とポストプロセッシングを自動化することが可能です。

ParaView アプリケーションのそれぞれに対し、コードを最大限に共有化するためのユーザインターフェイス部品のライブラリが利用可能です。**ParaView サーバ・ライブラリ**によって、並列かつインタラクティブな可視化を実行するための抽象化レイヤが提供されます。ParaView サーバ・ライブラリによって、クライアントアプリケーションは ParaView が並列に実行されているか否か、またどのように並列実行されているか、といったことに関する問題の多くから解放されます。**Visualization Toolkit** (**VTK**) は、基本的な可視化およびレンダリング・アルゴリズムを提供します。VTK にはレンダリング、並列処理、ファイル入出力、並列レンダリングのような基本的な機能を提供するライブラリも含まれます。このチュートリアルでは、ParaView に付属するクライアントアプリケーションを使用して ParaView の使い方を説明しますが、ParaView 自体は、その高度に部品化された設計によって、大幅な柔軟性およびカスタマイズの可能性を有しています。

1.1 開発および資金提供

ParaView のプロジェクトは、Kitware Inc. とロスアラモス国立研究所の共同努力によって 2000 年に開始されました。初期の資金は、米エネルギー省 ASCI Views 計

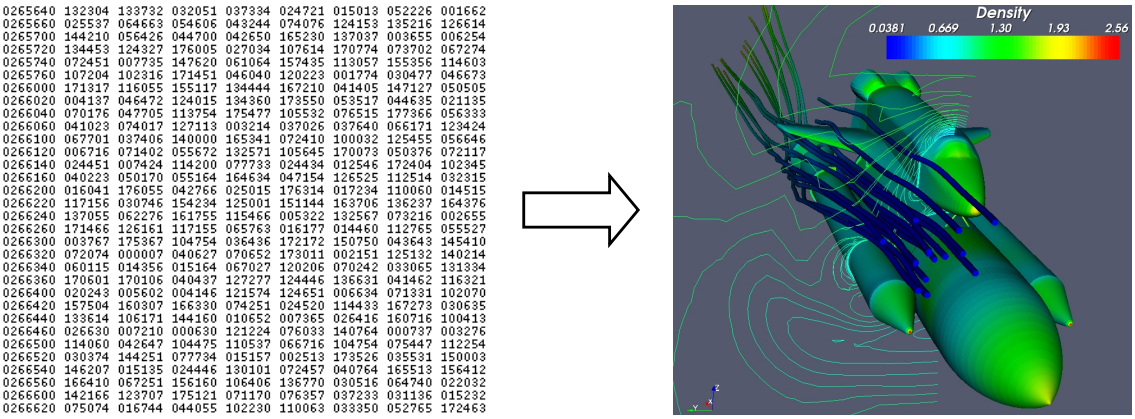
[1]訳注: 実際には図は省略されているようです。

画との 3 年契約によって提供されました。最初の公開リリースである ParaView 0.6 は、2002 年 10 月にアナウンスされました。ParaView の開発は、Kitware Inc. とサンディア国立研究所、ロスアラモス国立研究所、陸軍研究所、およびその他多くの学術・政府機関との共同作業によって継続されました。

2005 年 9 月には、Kitware、サンディア国立研究所、および CSimSoft は ParaView 3.0 の開発を開始しました。この開発においては、ユーザインターフェイスをさらにユーザフレンドリに書き直し、定量的な分析のためのフレームワークを開発することに注力しました。ParaView3.0 は 2007 年 5 月にリリースされました。

ParaView の開発は、今日も続いています。サンディア国立研究所は、ASC 計画を通じ、ParaView の開発への資金提供を続けています。ParaView は、SciDAC Institute for Ultra-Scale Visualization (www.ultravis.org) の主要な開発プラットフォームとして統合されています。米エネルギー省は、ロスアラモス国立研究所、陸軍の中小企業技術革新制度、および ERDC との契約を通じて ParaView に資金提供しています。米国立科学財団もまた、中小企業技術革新制度によって ParaView に資金提供しています。フランス電力公社、Mirarco、石油関連企業など、その他の機関も ParaView への支援を行っています。さらに、ParaView はオープンソースプロジェクトであるため、スイス国立スーパーコンピューティングセンターのような機関も開発成果をコントリビュートしています。

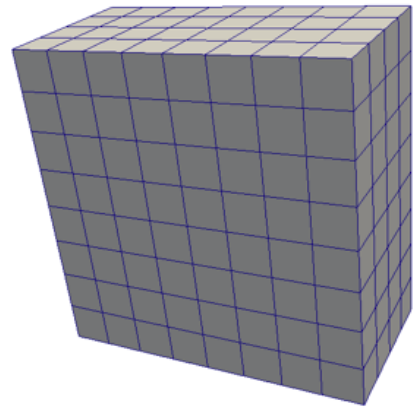
1.2 可視化の基礎



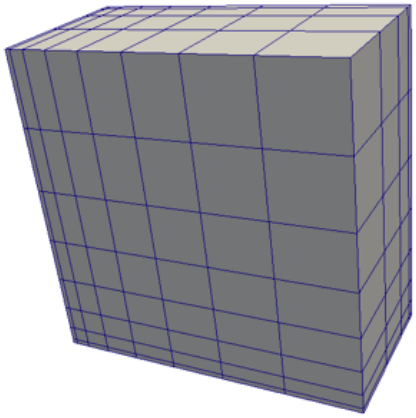
簡単に言うと、可視化の過程とは、生のデータを人間が見て理解することができる形式に変換することです。このことによって、データをより感覚的に理解することができるようになります。科学的な可視化では特に、2 次元あるいは 3 次元空間において明確な表現を有するデータを取扱います。シミュレーションのメッシュやスキャナのデータに由来するデータは、この種の分析に特に向いています。

データの可視化には、大別して 3 つの段階があります。すなわち、読込み、フィルタリング、レンダリングです。まず、データを ParaView に読み込みます。つぎに、データから特徴を生成、抽出、導出するためにデータを処理する**フィルタ**を、必要なだけ適用します。最後に、可視化された画像がデータからレンダリングされます。

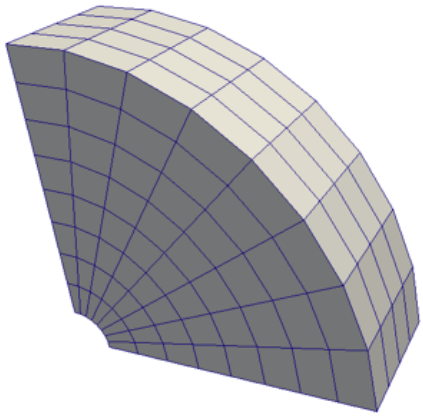
ParaView は基本的に、空間的に表現されるデータを取扱います。したがって、ParaView が使用する基本的な**データ型**はメッシュ (または格子) です。



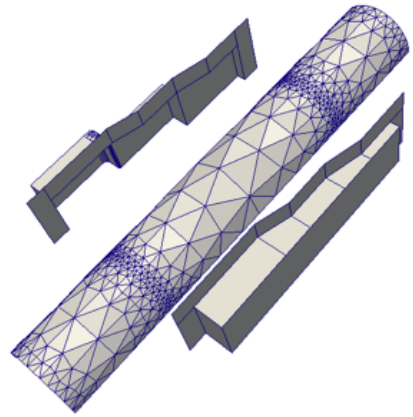
一様直線格子 (画像データ)
等間隔直線格子は 1、2、または 3 次元の配列データです。格子点は互いに直交し、各方向に等間隔に配置されます。



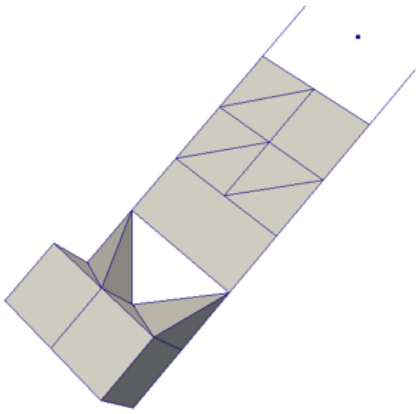
非一様直線格子 (直線格子)
等間隔直線格子と似ていますが、格子点間の距離を各座標軸方向に変化させることができます。



曲線格子 (構造格子)
曲線構造格子は、直線格子と同じトポロジを有します。ただし、曲線格子上の格子点は (セルどうしが重なったり、自分自身と交差しない限り) 任意の座標に置くことができます。曲線格子は直線格子の比較的少ないメモリ使用量、暗黙のトポロジといった特徴を継承しつつ、メッシュの形状に大幅な自由度を与えることができます。



ポリゴン・データ (ポリ・データ)
ポリゴン・データセットは、点、直線、2 次元の任意多角形から構成されます。セル間の結合は任意であり、結合しないこともできます。
レンダリングにおける基本的なプリミティブは、ポリゴン・データによって表されます。あらゆるデータは、(ボリュームレンダリングを除いて) レンダリングの前に必ずポリゴン・データに変換されますが、この変換は ParaView によって自動的に行われます。



非構造格子
非構造格子データセットは、点、直線、2次元の任意多角形、3次元4面体、非線形セルから構成されます。ポリゴン・データと類似していますが、直接レンダリングできないような、3次元の四面体[2]や非線形セルも表現することができます。

これらの基本的なデータ型に加えて、ParaViewは**マルチブロック**データもサポートしています。データセットがグループ化されたときや複数のブロックから成るファイルが読み込まれた時には、必ずマルチブロックのデータセットが生成されます。ParaViewはさらに、**階層化適応メッシュ分割**(**AMR**)、**階層化一様AMR**、**八分木**、**表**、そして**グラフ**データセットを表現するためのデータ型を有します。

1.3 さらなる情報

様々なところから、ParaViewに関するさらなる情報を入手することができます。初心者には、ParaViewアプリケーションの ボタンをクリックすることでアクセス可能な、オンライン・ヘルプがあります。このオンライン・ヘルプに加え、Amy Henderson Squillacoteによって書かれ、Kitwareから入手可能な *The ParaView Guide* は、ParaViewの全てに関するガイドとして役立ちます。

ParaViewのウエブページwww.paraview.orgもまた、ParaViewに関するさらなる情報を入手する絶好のサイトです。このサイトから、メーリングリスト、Wiki、よくある質問集、さらには有償サポートサービスへのリンクを辿ることができます。

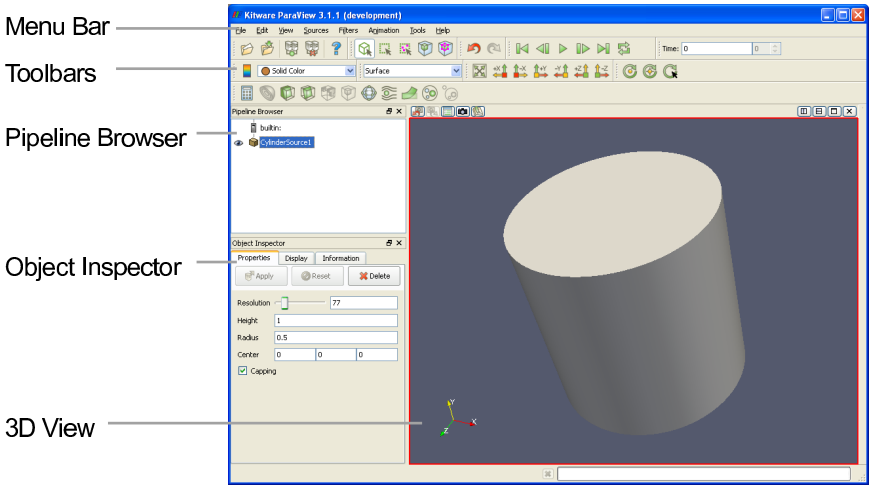
[2]訳注: ピラミッド、プリズム、六面体などもサポートしています。

第2章　基本的な使用法

それでは、ParaView を使ってみましょう。ここから先の内容を進めるには、自身で ParaView をインストールして頂く必要があります。とりわけ、この文書は ParaView のバージョン 3.8 に基づいています。既に ParaView のバージョン 3.8 をお持ちでなければ、www.paraview.orgからダウンロードできます (ダウンロードへのリンクをクリックして下さい)。ParaView は、大抵のアプリケーションと同じようにして起動します。Windows では、スタート・メニューから起動します。Macintosh では、インストールしたアプリケーション・バンドルを開いて下さい。Linux では、コマンド・プロンプトから `paraview` を実行して下さい (パスが設定されている必要があります)。

このチュートリアルの例では、http://www.paraview.org/Wiki/ParaView_Tutorial から入手可能なデータを使用します。このデータを、簡単にアクセスできる適当なディレクトリにインストールして下さい。このチュートリアルによってファイルを読み込むよう指示された時は、このデータをインストールしたディレクトリから読み込んで下さい。

2.1　ユーザ・インターフェイス



ParaView の GUI は、実行されているプラットフォームにおける形式 (ルック・アンド・フィール) に従いますが、基本的には全てのプラットフォームで同じようにふるまいます。ここに示したレイアウトは、ParaView が最初に起動された時のデフォルトのレイアウトです。GUI は以下の要素から構成されます。

メニュー・バー 他のプログラムと同様に、メニューバーから大部分の機能を利用することが可能です。

ツール・バー ツール・バーによって、ParaView の中でも最も一般的に使用される機能を素早く利用することができます。

Pipeline Browser (パイプライン・ブラウザ) ParaView は、パイプラインによってデータの読み込みとフィルタリングを管理します。パイプライン・ブラウザによって、パイプラインの構造を表示し、パイプライン・オブジェクトを選択することができます。ParaView 3 のための再設計によって、パイプライン・ブラウザは、パイプライン構造をインデントして表現する便利なリストとなりました。

Object Inspector (オブジェクト・インスペクタ) オブジェクト・インスペクタでは、現在選択されているパイプライン・オブジェクトの設定を表示および変更することができます。オブジェクト・インスペクタには、3 つのタブがあります。**Properties** (プロパティ) タブには、オブジェクトの状態に関して設定可能なオプションが提示されます。**Display** (ディスプレイ) タブには、ビュー画面でのオブジェクトの表現方法に関するオプションが提示されます。**Information** (情報) タブには、パイプライン・オブジェクトによって生成されたデータに関する基本的統計情報が表示されます。

3D View (3D ビュー) GUI の残りの部分はデータの提示に使用され、ここでデータを表示・操作・分析することができます。この部分は、データの幾何的表現を提示する 3D View で初期化されます。

ここで留意すべきは、GUI のレイアウトは大幅に設定変更が可能であり、そのためウインドウの見た目の変更が容易であることです。ツールバーは各所に移動し、あるいは隠すこともできます。ツールバーの表示を変更するには、View → Toolbars メニューを使用して下さい。パイプライン・ブラウザとオブジェクト・インスペクタは、いずれも**ドック可能な**ウインドウです。すなわち、これらの要素は GUI の中を各所に移動したり、フローティング・ウインドウとして切離したり、完全に隠すことができます。これらの 2 つのウインドウは ParaView の操作に重要なので、これらを隠した後に再度必要となった場合は、View メニューから表示することができます。

2.2 ソース

ParaView にデータを入力するには、2 とおりの方法があります。すなわち、データをファイルから読み込むか、または**ソース**オブジェクトによって生成する方法です。全てのソースは Sources メニューにあります。ソースによって例えば、ビューに

注記を追加することもできますし、ParaView の機能を (実際に動作させながら) 調べたい時にも非常に便利です。

演習 2.1: ソースの作成

簡単な例から始めましょう。Sources メニューから、Cylinder (シリンダ) を選択して下さい。そうすると、パイプライン・ブラウザに Cylinder1 という項目が追加され、選択された状態となります。さらに、オブジェクト・インスペクタには、シリンダ・ソースのプロパティが表示されています。ここではデフォルトの設定で良いので、そのまま Apply ボタン 

をクリックして下さい。

Apply をクリックすると、シリンダ・オブジェクトが右側の 3D ビュー・ウインドウに表示されます。この 3D ビューは、3D ビュー上でマウスをドラッグすることで、視点を操作することができます。マウスのそれぞれのボタン (左、中、右) を押しながらドラッグすることで、回転、平行移動、ズームの操作を行えます。また、シフトや Ctrl キーなどの修飾キーを押しながら、同様の操作を行ってみて下さい。

ParaView によって作成されるシリンダ・オブジェクトは真の円筒ではなく、多角形**小面**による円筒の近似であることにお気づきと思います。シリンダ・ソースのデフォルト設定では、わずか 6 小面からなる非常に粗い近似が生成されます。(実際のところ、このオブジェクトは円筒というより多角柱に近く見えます。) より円筒らしい表現が必要であれば、Resolution 設定の値を増やすことで、円筒に近い形にすることができます。

スライダもしくはテキスト入力フィールドを使用して、分割数を 50、またはそれ以上にして下さい。ここで、Apply ボタン 

が再び緑色 (Mac では青色) になったことに留意して下さい。これは、オブジェクト・インスペクタに対して行った変更が即座には適用されないためです。ハイライト状態のボタンは、パイプライン・オブジェクトのいずれかの設定が、現在表示されているものから変更されていることを示しています。Apply ボタンをクリックするとこれらの変更が適用され、Reset ボタン 

をクリックすると、全ての設定が最後に適用が行われた時の状態に戻ります。ここでは、Apply ボタンをクリックして下さい。真の円筒とほとんど見分けがつかない程、分割数が変わる筈です。 ♦

ここで、ツールバーの取消し 
および再実行 
ボタンを紹介します。データの可視化は多くの場合、試行錯誤の過程であり、一つ前の状態に戻れることが有用な状況があります。実際、データを作成する前の時点にまで取消しで戻って、再実行を行うことができます。

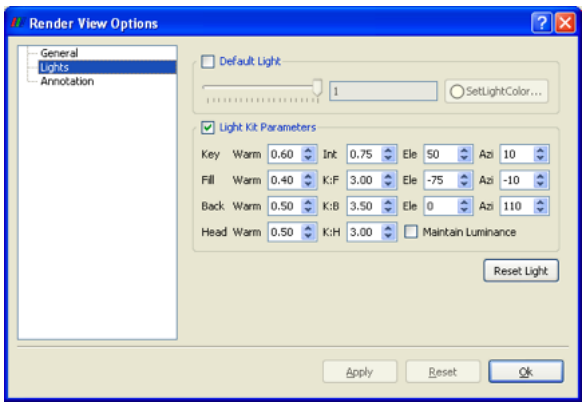
演習 2.2: 取消しおよび再実行

取消し 
ボタンと再実行 
ボタンを試してみて下さい。もしその為の準備が整っ

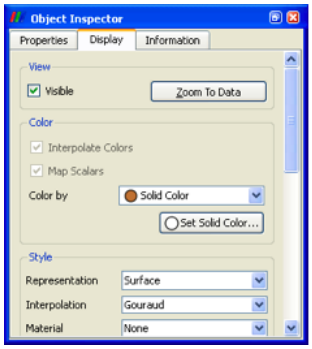
ていないのなら、演習 2.1で行ったように、パイプライン・オブジェクトを作成および変更して下さい。設定の変更が取消されたり、再び適用される様子を見てください。同様に、パイプライン・オブジェクトの全体が消去されたり、再度作成されたりする様子も見て下さい。

さらに、カメラ操作の取消し  、およびカメラ操作の再実行  ボタンがあります。これらによって、今までに試行したカメラのアングルを行ったり来たりすることができ、マウス操作の誤りによって完璧なカメラアングルが崩れることを心配する必要がなくなります。カメラを様々に動かして、これらのボタンでカメラ操作を取消したり、再実行してみて下さい。 ♦

オブジェクトがどのようにレンダリングされるかを選択するための設定も、多数あります。3D ビューの上には  ボタンがあり、レンダリングの設定を変更することができます。これをクリックするとダイアログボックスが開き、背景色、ライティング、注釈などを変更することができます。



レンダリング設定のためのもう一つの操作パネルは、オブジェクト・インスペクタの Display タブです。このタブのレンダリング設定は、選択されたオブジェクトに限って適用されるものです。例えば表示・非表示の切替え、表示色、データセットの表現方法などを設定できます。利便性のため、これらのビュー設定の幾つかとオブジェクトの表示設定は、ParaView の GUI 上の他の場所 (ツールバーなど) にも存在します。



演習 2.3: レンダリング設定の変更

3D ビューの上の  ボタンをクリックして、Render View Options ダイアログ・ボックスを表示させて下さい。設定画面のそれぞれのパネルを試してみて下さい。以下のようにして、ビューの見え方を変更してみて下さい。(Apply もしくは OK をクリックして、変更を適用することを忘れないでください。)

- 背景色を変更して下さい。(デフォルトの色にリセットすることもできます。)
- 方向表示座標軸 (ビューの左下隅の座標軸) の表示の有無を切替えて下さい。
- ビュー内の方向表示座標軸を移動させてみましょう。(ヒント: 座標軸をインタラクティブな状態に設定し、3D ビュー内で座標軸をクリックやドラッグして下さい。)

つぎに、円筒の描画設定を変更してみましょう。(シリンダなどのパイプライン・オブジェクトをまだ作成していなければ、演習 2.1で行ったようにして作成して下さい。) パイプライン・ブラウザ上で円筒が選択されていることを確認して下さい。オブジェクト・インスペクタでは、Display タブを選択して下さい。

- 円筒の色 (Solid Color) を設定して下さい。(注: ツールバーの  ボタンでも、色を設定することができます。)
- 円筒に光沢感を追加して下さい。(ヒント: Specular Intensity を 1.0 にして下さい。)
- 円筒を透明にして下さい。(ヒント: Opacity の値を下げてください。)
- 目盛りのついた 3D 座標軸を表示させ、空間的な位置がわかるようにしてください (Show cube axes)。

シリンダ・ソースについては以上です。オブジェクト・インスペクタの Properties タブを選択し、削除 

をクリックすることで、パイプライン・オブジェクトを削除できます。 ♦

2.3 データの読込み

ParaView の GUI はいくらか練習しましたので、実際のデータを読込んでみましょう。ご想像のとおり、File メニューの最初の項目に Open コマンドがあり、さらにツールバーにもファイルを開くためのボタン  があります。ParaView は多くのファイル形式をサポートし、ファイル形式のリストは新たな形式が追加される度に長くなっています。以下は、現在読込みが可能なファイル形式です。

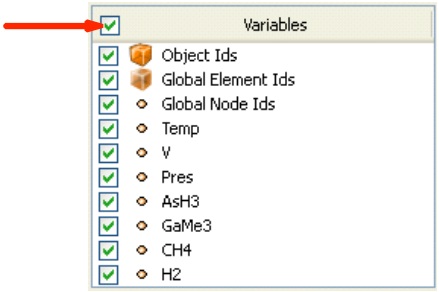
- ParaView Data (.pvd)
- VTK (.vtp, .vtu, .vti, .vts, .vtr)
- VTK Multi Block (.vtm, .vtmb, .vtmg, .vthd, .vthb)
- Partitioned VTK (.pvtu, .pvti, .pvts, .pvtr)
- VTK Legacy (.vtk)
- Exodus
- XDMF (.xmf, .xdmf)
- LS-DYNA
- SpyPlot CTH
- EnSight (.case, .sos)
- netCDF (.ncdf, .nc)
- BYU (.g)
- Protein Data Bank (.pdb)
- XMol Molecule
- PLOT3D
- Digital Elevation Map (.dem)
- VRML (.wrl)
- PLY Polygonal File Format
- Stereo Lithography (.stl)
- Gaussian Cube File (.cube)
- POP Ocean Files
- AVS UCD (.inp)
- Meta Image (.mhd, .mha)
- Facet Polygonal Data
- Phasta Files (.pht)
- SESAME Tables
- MFIX (.RES)
- Fluent Case Files (.cas)
- OpenFOAM Files (.foam)
- Cosmology Files (.cosmo)
- PNG Image Files
- TIFF Image Files
- Raw Image Files
- Comma Separated Values (.csv)

モジュール化された ParaView の設計によって、新たな VTK の読込みオブジェクトの ParaView への統合が容易になっています。従って、新たなファイルフォーマットがサポートされていないか、頻繁にチェックされることをお勧めします。もし、お探しのファイル読込みオブジェクトが ParaView に含まれていないようであれば、ParaView メーリングリスト (paraview@paraview.org) で調べてみて下さい。ParaView からは見えないものの、VTK には含まれており、容易に追加できるファイル読込みオブジェクトも多数あります。さらには VTK フレームワークへの組み込みが可能でありながら、VTK にはまだ組み込まれていない読み込みオブジェクトも多数あります。誰かがあなたの必要な読込みオブジェクトをそのような形で持っ

ており、快く提供してくれるかもしれません。

演習 2.4: ファイルを開く

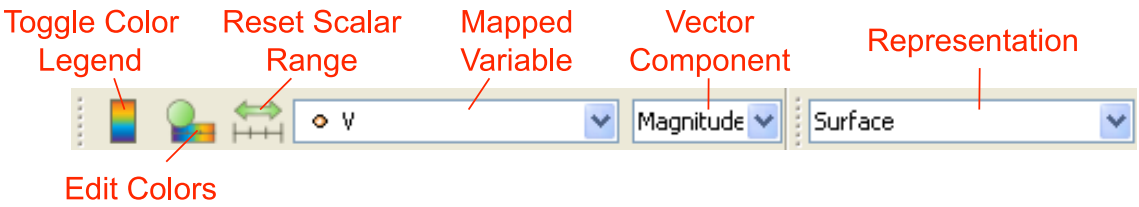
それでは、初めてのファイルを開いてみましょう。Open ツールバー (もしくはメニュー項目) 📂 をクリックし、disk_out_ref.ex2 を開いて下さい。ファイルを開く操作は 2 段階の操作であり、この段階では、まだ読込んだデータは見られないことに注意して下さい。その代わり、オブジェクト・インスペクタに、どのようにデータを読み込むかを指定するための設定が提示されます。



Variables (変数) リストのヘッダ (最上部) にあるチェックボックスをクリックし、全ての変数が読込まれるようにして下さい。これは小さなデータセットなので、メモリに読み込むデータ量が過大になることを心配する必要はありません。全ての変数を選択したら、 

をクリックして全てのデータを読込んで下さい。データが読込まれたら、一端に穴の開いた円筒のような形状が表示されます。このデータは、加熱された回転円盤周囲の気流シミュレーションの結果です。表示されているメッシュは円盤周りの空気で、シミュレーションの領域境界が円筒形になっています。真ん中の中空部分は、もしこのシミュレーションのためにこの部分もメッシングされていれば、加熱された円盤が存在するはずの場所です。 ♦

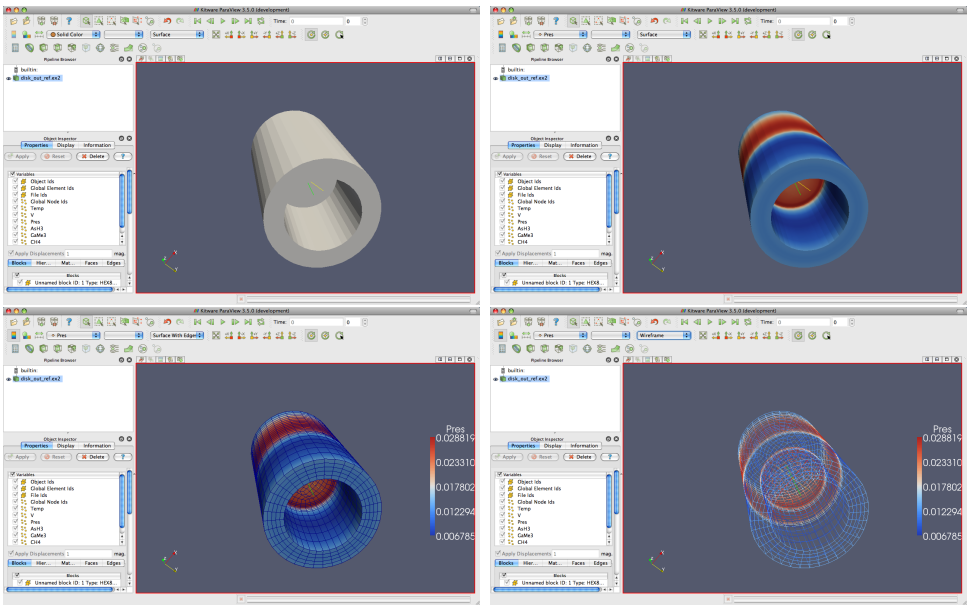
データのフィルタリングに進む前に、データの表現方法の幾つかを簡単に見ておきましょう。データの表現に関する最も一般的な設定は、2 つのツールバーに配置されています。



演習 2.5: 表現方法とフィールド・データによる色付け

データの表現方法を幾つか試してみましょう。パイプライン・ブラウザで、disk_out_ref.ex2 が選択されていることを確認して下さい。(もし、まだデータを読込んでい

なければ、演習 2.4の手順を行って下さい。) 変数選択 (上図の Mapped Variable) を使用して、表面を Pres 変数で色付けしてみて下さい。次に色の凡例表示 (上図の Toggle Color Legend) をオンにして、実際の圧力の値を見てみて下さい。メッシュの構造を見るには、表現方法 (上図の Representation) を Surface With Edges にします。Wireframe 表現にすると、セル構造とメッシュの内部の両方を見ることができます。



♦

2.4 フィルタ

これまでのところ、とりあえずはデータの読込みに成功し、そのデータの情報を幾らか見ることができました。つまり、メッシュの基本的な構造を表示し、メッシュの表面に何らかのデータをマッピングできるようになりました。しかしながら、これから見るように、データの表面をただ眺めるだけでは判らない興味深い特徴が、このデータには多数あります。(スカラーやベクトルのような) 異なった型の多くの変数が、このメッシュには関連づけられています。さらに、メッシュはソリッド (中実) なモデルであることに注意して下さい。興味深い情報の多くは、内部に存在しています。

フィルタを適用することで、データに関してさらに多くのことを発見できます。フィルタとは、データを処理して、データから特徴を生成、抽出、または導出するための機能ユニットです。フィルタは読込みオブジェクト、ソース、あるいは他のフィルタに接続され、それらのデータに対し何らかの形で変更を加えます。これらの互いに接続されたフィルタによって、**可視化パイプライン**が形成されます。ParaView

では、非常に多数のフィルタが利用可能です。以下は対応するフィルタ・ツールバー上のアイコンをクリックすることで利用できる、最も一般的なフィルタです。

 **Calculator (電卓)** 格子点またはセル毎に、ユーザによって定義された数式を評価します。

 **Contour (コンター)** スカラー場がユーザによって指定された値に等しくなるような点、曲線、面を抽出します。この面は**等値面**とも呼ばれます。

 **Clip (クリップ)** 形状を半空間で切断します。その結果、ユーザによって定義された面のいずれか一方の側の形状が削除されます。

 **Slice (スライス)** 形状を面で切断します。その結果はクリップと同様ですが、面上の形状だけが残されます。

 **Threshold (しきい値)** スカラー場の指定された範囲に存在するセルを抽出します。

 **Extract Subset (サブセットの抽出)** 抽出すべきボリュームあるいは間引き率を指定し、格子のサブセットを抽出します。

 **Glyph (グリフ)** **グリフ**、すなわち単純な形状をメッシュの各格子点に配置します。グリフはベクトルによって方向を指定し、ベクトルまたはスカラーによってスケーリングすることができます。

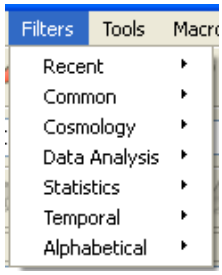
 **Stream Tracer (流線追跡)** ベクトル場にシード点を配置し、(定常の) ベクトル場をそれらのシード点から追跡します。

 **Warp (vector) (ワープ (ベクトル))** メッシュの各格子点を、与えられたベクトル場で変形させます。

 **Group Datasets (データセットのグループ化)** 複数のパイプライン・オブジェクトの出力を、一つのマルチブロック・データセットに統合します。

 **Extract Level (レベルの抽出)** マルチブロック・データセットを構成する要素 (ブロック) を抽出します。

これらの 11 種のフィルタは、ParaView で利用可能なもののごく一部の例です。Filters メニューには、データ処理のためのフィルタがさらに多数存在します。ParaView からは現在のところ 100 以上のフィルタが利用可能であるため、簡単にフィルタを見つけられるよう、Filters メニューは以下のようにサブメニューに整理されています。



Recent (最近使ったフィルタ) 最も最近使われたものが一番上に来るようソートされた、最も最近使われたフィルタのリストです。

Common (一般的) 最も一般的なフィルタです。これはフィルタ・ツールバーにリストされているフィルタと同じで、前述のとおりです。

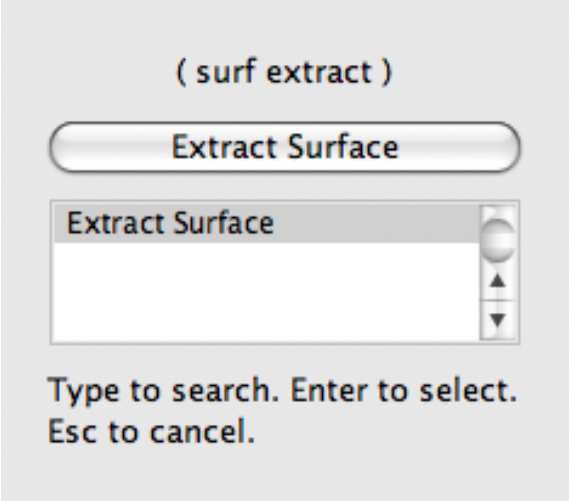
Cosmology (宇宙論) これには、LANL (ロスアラモス国立研究所) で開発された、宇宙論に関する研究のためのフィルタが含まれます。

Data Analysis (データ分析) 定量的な値をデータから取り出すためのフィルタです。これらのフィルタはメッシュ上でデータを計算したり、メッシュから要素を抽出したり、データをプロットするのに使用します。

Statistics (統計) ここには、データの統計値を計算するフィルタが含まれます。それらは、基本的には表形式で表現されます。

Temporal (時間依存型) 時間によって変化するデータを分析、あるいは加工するフィルタです。全てのフィルタは各時刻のスナップショットに対して実行されるため、時間によって変化するデータに対して使用可能です。しかしながらこの分類のフィルタは特に、利用可能な時間範囲を走査し、データが時間とともにどのように変化しているかを調べることができます。

Alphabetical (アルファベット順) 全ての利用可能なフィルタのアルファベット順のリストです。あるフィルタがどこにあるか判らなければ、このリストには確実に存在します。また、このリストにしか存在しないフィルタも多数存在します。



これらのフィルタの一覧、特にアルファベット順の全フィルタのリストはあまりに多く、目的のフィルタを探すのは面倒です。フィルタを素早く選ぶには、**クイック起動**ダイアログを使用して下さい。Windows および Linux では Ctrl と Space キー、Macintosh では Alt と Space キーを同時に押すと、ここで示すような小さなダイアログボックスが現れます。フィルタ名に含まれる単語、またはその一部を入力すると、その入力した語を含むソースとフィルタが一覧表示されます。目的のフィルタまたはソースを選択し、Enter キーを押すと、それがパイプライン・ブラウザに追加されます。

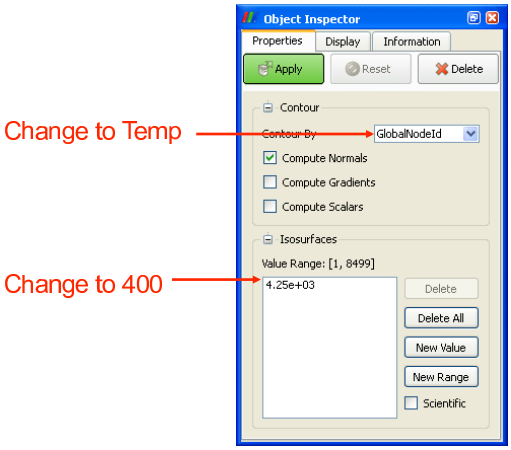
フィルタの一部はメニュー上でグレーになっていて、選択できないことにお気づきかと思います。特定の型のデータにしか適用できないため、常に使用できる訳で

はないフィルタも多数あります。ParaView はそのようなフィルタをメニューおよびツールバーから選択不可にし、それらのフィルタが使用不可であることを示し (また、それを強制的に使用できないようにし) ます。

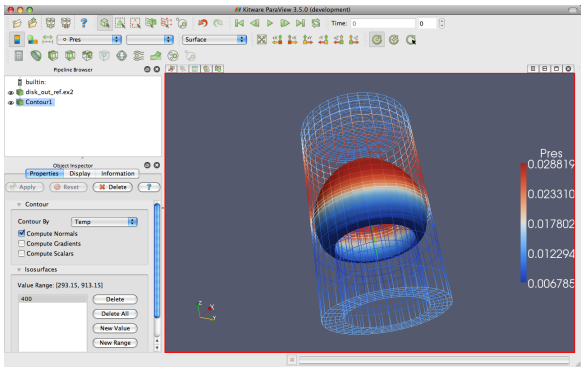
このチュートリアルでは多くのフィルタを使用しますが、それでも全てを試せる訳ではありません。それぞれのフィルタのさらなる情報については、ParaView のオンラインヘルプ、もしくは内蔵のヘルプのフィルタ・メニューの章をご覧下さい。

演習 2.6: フィルタを適用する

それでは最初のフィルタを適用しましょう。disk_out_ref.ex2 がまだ読込まれていなければ、ここで読込んで下さい (演習 2.4)。パイプライン・ブラウザ上で disk_out_ref.ex2 が選択されていることを確認し、フィルタ・ツールバーまたは Filters メニューから、Contour フィルタを選択して下さい。パイプライン・ブラウザの読込みオブジェクトの下に新たな項目が追加され、オブジェクト・インスペクタにフィルタの設定画面が表示されます。ファイルの読込みと同様に、フィルタの適用も 2 段階の操作です。フィルタの作成後、フィルタの適用前に設定を変更することができます (そしてそれはほぼ大抵の場合、必要な操作です)。



コンター・フィルタを使って、温度が 400 K の等値面を作成することにしましょう。まず、Contour By の設定を Temp 変数に変更します。つぎに、等値面の値を 400 に変更します。最後に、Apply をクリックしてください。等値面がボリュームの内部に現れる筈です。もし disk_out_ref.ex2 が演習 2.5のとおりに圧力で色付けされたままであったら、等値面も同様に圧力で色付けされます。



◆

以上の演習では、フィルタを使ってデータを加工し、必要な結果を得る方法を学んで来ました。ほとんどの一般的な処理では、単一のフィルタで必要な情報を得ることができます。しかしながら、フィルタは読込みオブジェクトと同種のオブジェクトです。すなわち、読込みオブジェクトに対して適用するような操作を、フィルタに対しても適用することができます。それはつまり、あるフィルタによって生成されたデータに対して、さらにフィルタを適用することができるということです。これらの一連の互いに接続された読込みオブジェクトとフィルタを、**可視化パイプライン**と呼びます。この可視化パイプラインを形成する機能は、要求に合わせてデータの可視化を行うための強力な仕組みです。

もう少し、フィルタを色々試してみましょう。内部にあるものの表示をしばしば妨げるメッシュ表面のワイヤフレーム表示の代わりに、表面の一部を切り取った状態にしましょう。この操作には 2 つのフィルタが必要です。すなわち、1 つ目のフィルタで表面を抽出し、2 つ目で表面の一部を切り取ります。

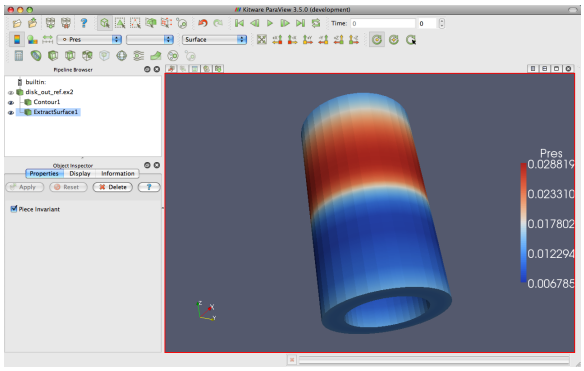
演習 2.7: 可視化パイプラインの作成

この演習の画像および議論のいくつかは、演習 2.6を終えた直後の状態からこの演習を始めることを前提としています。もし ParaView を再起動していたり、その他の理由で異なる状態にある場合は、たんに disk_out_ref.ex2 を読込めば充分です。

それでは表面を抽出するフィルタを追加しましょう。それは以下の手順で行います。

1. パイプライン・ブラウザで disk_out_ref.ex2 を選択します。
2. メニューバーから Filters → Alphabetical → Extract Surface を選択します。 もしくはクイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を呼び出して、extract surface と入力し、そのフィルタを選択して下さい。
3. 

ボタンをクリックしてください。



Extract Surface フィルタを適用すると、再びメッシュの表面が見えるようになります。元のメッシュと同じように見えますが、元のデータが中実なのに対し、このデータは中空である点で異なっています。

コンター・フィルタの結果を表示させていた場合は、コンターが見えなくなりましたが、心配する必要はありません。表面に隠されているだけで、依然としてそこに存在しているからです。もしフィルタの適用後の表示に何も変化が無ければ、1 番目の disk_out_ref.ex2 を選択する操作を忘れて、違うオブジェクトにフィルタを適用してしまった可能性があります。もし ExtractSurface1 オブジェクトが disk_out_ref.ex2 に直接接続されていなければ、それが間違えた操作です。もしそのとおりであれば、フィルタを削除して再度適用を試みてください。

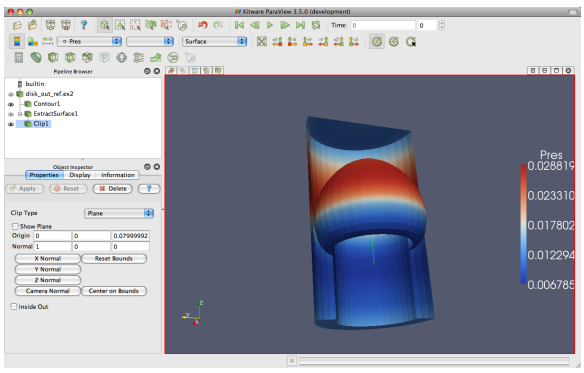
それでは表面を切り取って、(もし存在すれば) 内部の構造と等値面が見えるようにしましょう。

4. パイプライン・ブラウザ上で、ExtractSurface1 が選択されていることを確認します。

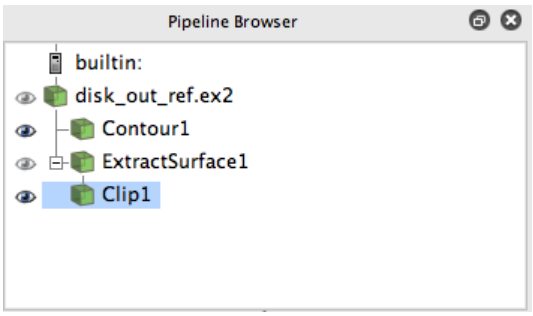
5. ツールバーもしくは Filters メニューから、クリップ・フィルター  を選択してください。

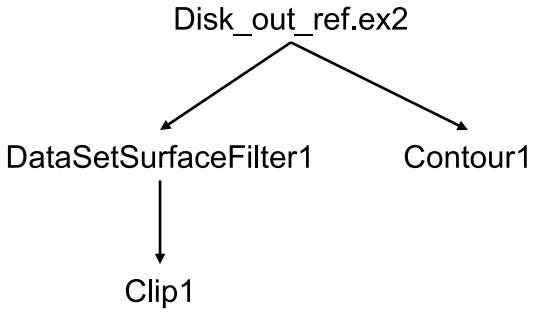
6. オブジェクト・インスペクタの Show Plane チェックボックス  から、チェックを外してください。

7.  ボタンをクリックしてください。



もしコンターを作成していたのであれば、メッシュ表面を切り取った内側から、等値面コンターが見える筈です。コンターがはっきり見えるためには、メッシュを回転させる必要があるかもしれません。 ♦





パイプラインにフィルタが幾つか追加されましたので、パイプライン・ブラウザでのこれらのフィルタの配置を見てみましょう。パイプライン・ブラウザによって、今までに作成されたパイプライン・オブジェクトは判りやすいリストとなっており、また、パイプライン・オブジェクトを選択し、それらのオブジェクトの隣にある目のアイコン をクリックすることで**表示・非表示**を簡単に変更できるようになっています。しかしさらに、このリスト中の項目のインデントと、インデントを辿って右側へ折れ曲がった線にも注目して下さい。これらの機能はパイプラインの**連結状態**を示しています。これは右の図の従来型のグラフと同じ情報を、ずっとコンパクトなスペースで表現しています。パイプライン・オブジェクトの従来型配置の問題点は、多くのスペースが必要であり、さほど大規模でないパイプラインでさえ完全に見渡すには GUI の大部分が必要なことでした。しかし、このパイプライン・ブラウザは、完全でありながらコンパクトです。

2.5 複数ビューの利用

科学上の目的で時折、1 つの変数に絞って検討することがあります。しかし、多くの重要な物理現象は、互いに何らかの影響を与え合う複数の変数で記述されます。それゆえ、1 つの視点で多くの変数を表示するのは大変難しいといえます。ParaView はデータを多様な視点で表示し、それらを相互に関連付ける機能を持っており、そのような機能は複雑な可視化データを考察する上で有用です。

今までのところ、可視化では 2 つの変数に注目しています。すなわち、圧力を色のグラデーションで表し、温度による等値面を抽出して表示しています。一見、2 つの変数をきれいに配置できているようにも見えますが、それらを分かりやすく関連付けることはできていません。複数の視点を使うことで、両者の関係を分かりやすくすることができます。各視点がデータの個別な特徴を表示し、さらにそれらを組み合わせることで、より理解しやすい可視化とすることができます。

各ビューの上部には小さなツールバーがあり、このツールバーの右側にビューを作成したり、削除するためのボタンがあります。全部で 4 つのボタンがあります。

もしくは ボタンを押すことで、既に存在するビューをそれぞれ水平もしくは垂直に分割して新たなビューを作成することができます。 ボタンによってビューは削除され、そのビューによって占められていたスペースは隣接するビューで占められます。 ボタンによって、選択したビューを一時的にビュー画面いっぱいに拡大して表示することができ、 で元の状態に戻ります。

演習 2.8: 複数ビューの利用

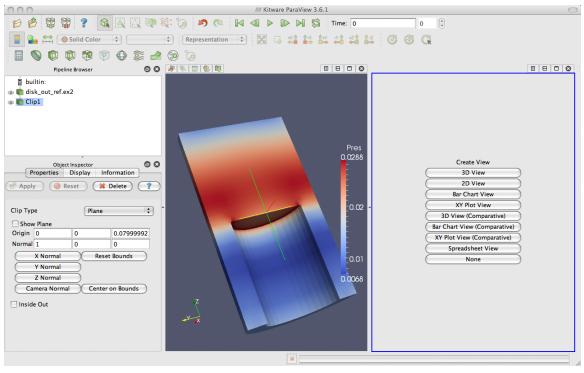
新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、 ボタンを押すのが最も簡単です。このボダンが何を行う物なのかは3章でさらに詳しく解説しますが、今のところは ParaView を再起動するのと概ね同等であると理解しておいて下さい。

まず最初に、1つの変数に着目します。メッシュの中にある変数を見たいので、メッシュを半分にクリップしましょう。

1. disk_out_ref.ex2 ファイルを開き、変数を全て読込んで Apply をクリックしてください (演習 2.4を参照して下さい)。
2. disk_out_ref.ex2 にクリップ・フィルタ を適用してください。
3. オブジェクト・インスペクタの Show Plane のチェックボックス Show Plane からチェックを外して下さい。
4. Apply ボタンをクリックしてください。
5. (ツールバーの) 変数選択を、Solid Color から圧力 Pres に変更し、表面を色付けします。

メッシュ内部の平面の圧力分布が見えるようになりました。ここでさらに、同じ平面の温度分布と比較したいとします。そのためには、新しいビューを作成することで、もうひとつの可視化を行うことができます。

6. ボタンをクリックしてください。

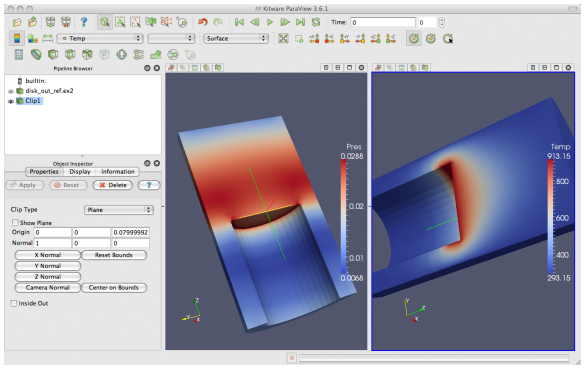


現在の画面は半分に分割され、右側には何も表示されていません。この何も表示されていない部分に、新しく可視化を表示します。右の画面には、周囲に青い境界線が表示されていることに注目してください。これは**アクティブなビュー**であることを示しています。パイプライン・ブラウザやオブジェクト・インスペクタなどの、あるビューの情報を表示したり、各種設定を制御するウイジェットは、アクティブなビューについてそれらを行います。この新しいビューには、メッシュの温度分布を表示させます。

1. 青い境界線が、(右側の) 新しい空白のビューの周囲に表示されている状態にしてください。どのビューでもクリックすることで、アクティブなビューにすることができます。

2. パイプライン・ブラウザの Clip1 の隣にある目のアイコン  をクリックして、クリップしたデータの表示をオンにしてください。

3. パイプライン・ブラウザ上で Clip1 を選択し、(ツールバー内の) 変数選択で変数を Solid Color から Temp に変更し、表面を温度分布で色付けしてください (この時点で、色の凡例の表示をオンにしても良いです)。



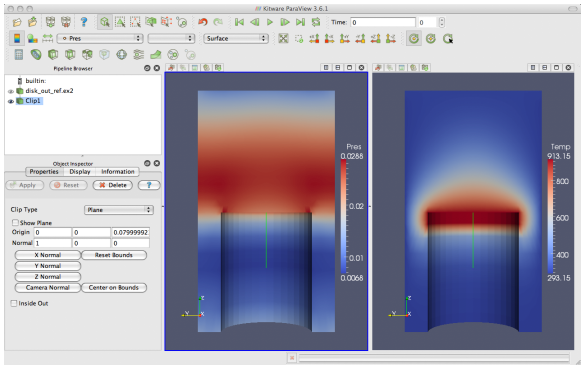
画面に 2 つのビューができました。1 つは圧力の情報を、もう 1 つは温度の情報を表示しています。これらを比較したいところですが、2 つの画面の視点方向が違うために困難です。どのようにして、一方の視点位置をもう一方の始点位置と連動させることができるでしょうか？この問題を解決するには、**カメラのリンク**を使います。**カメラのリンク**を使うことで、2 つの画面は常に同じ視点から描画されます。カメラのリンクは簡単にできます。

4. いずれかのビューの上で右クリックし、ポップアップメニューから Link Camera... を選択してください。(もし Mac を利用していてマウスに右ボタンがない場合、Tools → Add Camera Link... のメニューオプションを選択することで同様の操作が可能です。)

5. もう一方の画面をクリックしてください。

6. それぞれの画面でカメラを動かしてみて下さい。

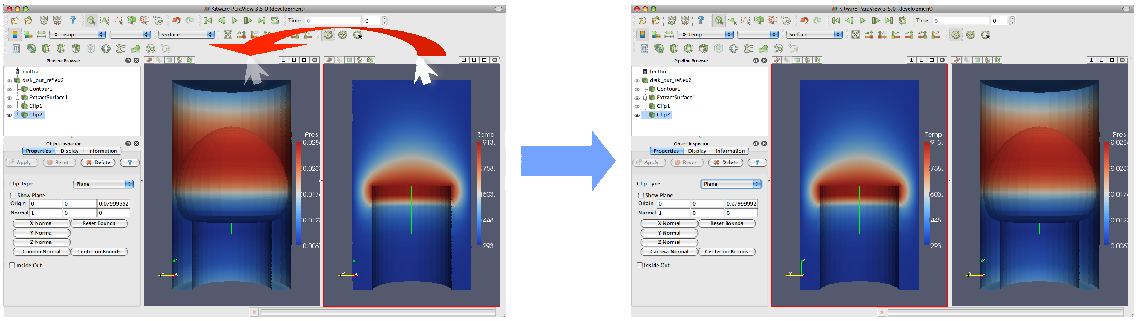
やった！2つのカメラがリンクされました。どちらのカメラも、もう一方を追随して動きます。カメラがリンクされると、2つの画面を比較して見ることができます。 ボタンをクリックして、視点を断面をまっすぐに見る位置にしてください。



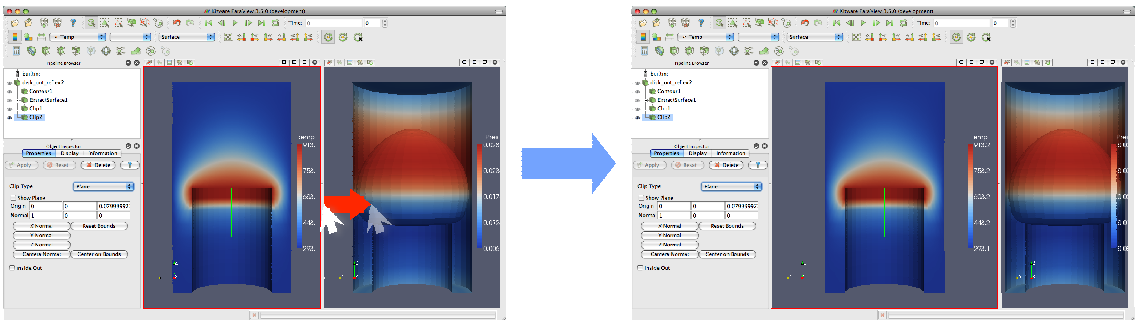
空気温度が、熱せられた円盤との接触面で最大となっていることがわかります。それ自体は、驚くことではありません。空気温度が熱源近くで最大となり、離れるに従って低くなっていくことは予想されます。しかし、同じ地点で圧力が最大ではないことに注目してください。空気の圧力は、円盤の上空で最大値を取っています。この情報を基に、この物理的現象にいくつかの興味深い仮説を立てることができます。空気の圧力分布には、2つの力が関係していることが考えられます。1つ目の力は重力による力で、上方の空気が下方の空気を押さえつける働きをします。2つ目の力は浮力で、熱せられた空気が密度が小さくなることで上方に昇ろうとする力です。空気の圧力が最大値を取る位置から、これらの2つの力が釣り合う点が判ります。このような考察は、温度と圧力の両方を同時に見なければできないでしょう。♦

もちろんParaViewの複数視点の機能は、同時に2つ以上の視点も表示できます。各々のビューには1セットずつ、複数視点の為の分割ボタンがあります。ビュー分割ボタン を使うことでさらにビューを作成し、ワークスペースを分割することができます。そして分割したビューは、 ボタンでいつでも削除することができます。

各ビューの表示位置も固定ではありません。ビューのツールバー上 (ボタンでない部分) をクリックし、マウスのボタンを押したまま別のビューの上にドラッグすることで、2つのビューの表示位置を入れ替えることができます。この操作によって、2つのビューは即座に入れ替わります。



2つのビューの間をクリックし、マウスのボタンを押したままどちらか一方のビューの方向にドラッグすることで、ビューのサイズを変更することもできます。ビューの境界はマウスの動きに従って動き、ビューの大きさもそれに合わせて調整されます。



2.6 ベクトルの表示

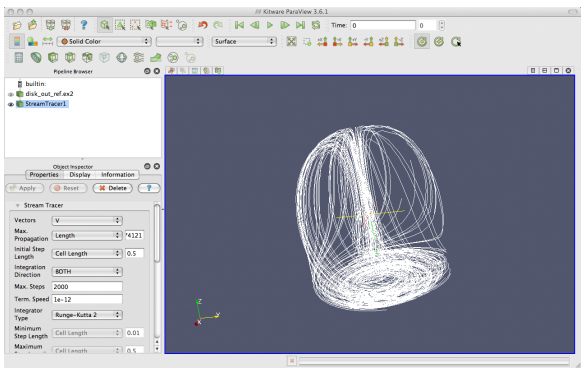
このシミュレーション結果から、他にどのような結論を導けるでしょうか。シミュレーションは、熱せられた回転円盤上の空気の流れを表す速度場も算出しています。ParaView を使って、空気の流れを確認します。

ベクトル場を描画するための一般的で効率的な方法は、**流線**を使用することです。流線とは、あらゆる点でベクトル場に沿っている空間中の曲線です。流線はまた、無重量の粒子がベクトル場で取る経路を表します (流れが定常である場合)。 流線は、**シード点**を与えることで生成されます。

演習 2.9: 流線

新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

1. disk_out_ref.ex2 ファイルを開き、変数を全て読込んで  をクリックしてください (演習 2.4を参照して下さい)。
2. 流線追跡フィルタ  を disk_out_ref.ex2 に適用してください。
3.  ボタンをクリックして、デフォルトの設定を適用します。



メッシュの表面が、渦巻いた線に置き換えられました。これらの線は、ボリュームを通る流れを表しています。円筒の中心軸周りに、渦のような流れができていることに注目してください。中心や端部周辺には、垂直方向の流れもあります。

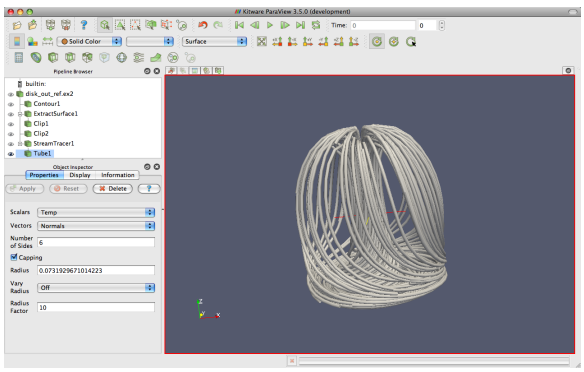
新しい形状は、元々あった形状と中心がずれています。**カメラのリセット**  コマンドを使うことで、新しい形状の視点を即座に中心に移動させることができます。このコマンドは、表示されている形状を現在のビューに収め、中心を移動させると共に、カメラの回転の中心を表示されている形状の中心に一致させます。 ♦

流線の問題の 1 つは、現在表示されている状態がそうであるように、多くの線が互いに接するように表示され、また陰影もないため、線同士の見分けがつかないことです。線は 1 次元で、陰影を表現するには 2 次元の面を必要とするからです。流線のもう 1 つの問題は、流れの方向がわからないことです。

次の演習では、演習 2.9で作成した流線を修正して、これらの問題を解決していきます。チューブ・フィルタを使うことで、流線の周りに 2 次元の面を作成することができます。この面によって、流線に陰影と距離感の手がかりが付加されます。更に流線に矢印を追加することで、流れの方向を示すこともできます。

演習 2.10: 流線を見やすくする

1. クイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を使用して、Tube フィルタを適用します。
2.  ボタンをクリックしてください。

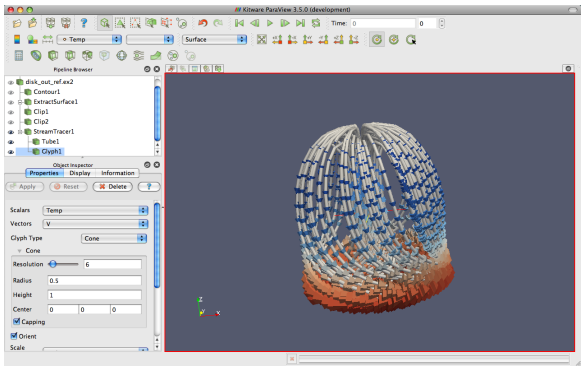


流線が鮮明に見えるようになりました。流線を横方向から眺めてみると、気流の循環を見ることができます。空気が熱せられ上昇し、冷えると下降しています。流線を回転させて Z 軸を見下ろすようにし、熱せられた円盤がある筈の辺りを見ると、回転する円盤との摩擦によって空気が円形に流動しているのが分かります。

もう少し、装飾を施してみましょう。流線に矢印を加えることで、流れの方向と大きさを示すことができます。

3. パイプライン・ブラウザで StreamTracer1 を選択してください。

4. StreamTracer1 にグリフ・フィルタ  を加えてください。
5. オブジェクト・インスペクタで、Vectors の設定 (上から 2 番目の設定) を V に変更してください。
6. オブジェクト・インスペクタで、Glyph Type の設定 (上から 3 番目の設定) を Cone に変更してください。
7.  ボタンをクリックしてください。
8. 矢印を Temp 変数で色付けしてください。



このようにして、流線に小さな矢印が追加されました。矢印の向きは速度の方向を示し、大きさは速度の大きさに比例しています。この新たな情報を利用して、次の問題に答えてみましょう。

- 空気が一番速く流れているのはどこでしょうか？円盤の近く、もしくは離れているでしょうか？円盤の中央付近か、それとも周縁付近でしょうか？
- 円盤の回転の向きはどちらでしょうか？
- 円盤表面では、空気は円盤の中心に向かって流れていますか、それとも周縁部分に向かって流れていますか？

♦

2.7 プロット

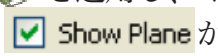
ParaView のプロット機能によって、データを掘り下げ、定量的な解析を行うことが可能になります。プロットは通常、フィルタによって作成され、プロットを作成するためのフィルタは、全て Filters メニューの下の Data Analysis サブメニューにあ

ります。次の演習では、空間内の線分上におけるメッシュ中のフィールドの値をプロットするフィルタを作成してみましょう。

演習 2.11: 空間中の線分上の値をプロットする

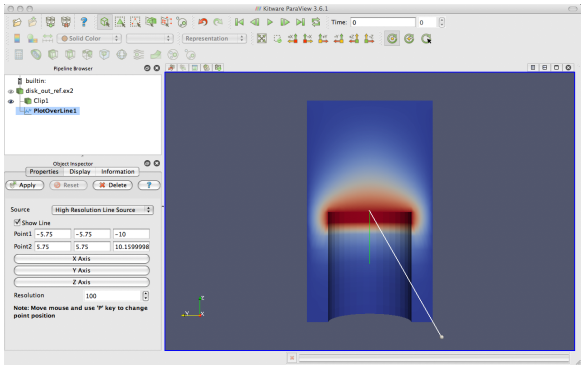
新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

1. disk_out_ref.ex2 ファイルを開き、変数を全て読込んで  をクリックしてください (演習 2.4を参照して下さい)。

2. (演習 2.8のように) disk_out_ref.ex2 にクリップ・フィルタ  を適用し、オブジェクト・インスペクタの Show Plane のチェックボックス  からチェックを外し、  をクリックして下さい。これによって、値をプロットするための線分が見やすく、また操作しやすくなります。

3. パイプライン・ブラウザで disk_out_ref.ex2 をクリックし、アクティブなオブジェクトにして下さい。

4. メニュー・バーから Filters → Data Analysis → Plot Over Line  を選択するか、クイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を使用して、Plot Over Line フィルタを適用します。



アクティブなビューに、両端に球の有る線分が、表示されているデータを貫く形で描画されます。これらの球のどちらかをマウスでドラッグすることで、3D ビュー内を移動させることができます。画面内の球を動かすたびに、オブジェクト・インスペクタの項目の幾つかも変化することにも注目して下さい。目的の場所にマウスポインタを移動させ、p キーを押すことでも球を移動させることができます。この場合、マウスポインタの位置に存在する面に、球を順次移動させることになります。これがクリップ・フィルタを使用した目的で、これによってクリッピング面に端点を置くことができます。この方法で球を移動させることができるのは、ソリッドにレンダリングされた面に対してのみであることに気をつけてください。ボリューム・レンダリング[1]に対しては、この方法は使えません。

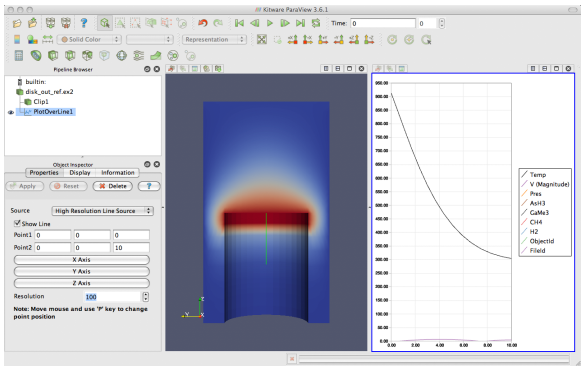
[1]訳注: ボリューム・レンダリングについては、次節で説明します。

この線分と球などの表現は **3D ウィジェット**と呼ばれています。なぜなら、これは 3 次元空間内で操作される GUI コンポーネントだからです。ParaView には、多くの 3D ウィジェットの例があります。特にこのライン・ウイジェットは、空間内で線分を指定するのに使用されます。他にも、点や面を指定するためのウィジェットがあります。

5. 線分が円盤の基部からメッシュ全体の頂部へ垂直に結ばれるように、3D ウイジェットを操作するか、p キーを押すか、もしくはオブジェクト・インスペクタで数値指定をして調整してください。

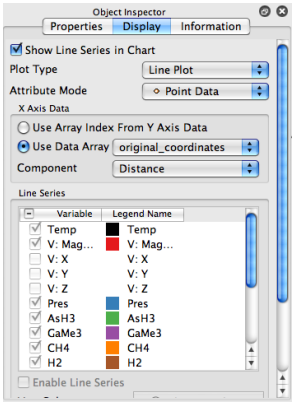
6. 線分を目的の位置に配置できたら、 

ボタンをクリックします。

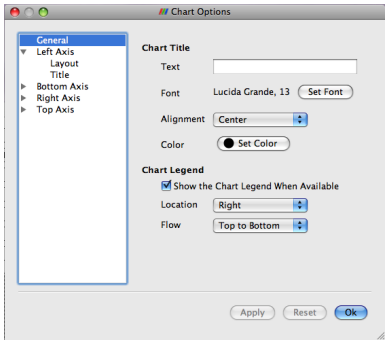


プロットの際に行える操作がいくつかあります。マウスの中央ボタンをクリックした状態で上下にドラッグすると、拡大・縮小を行う事ができます。右ボタンでドラッグすると、ドラッグ範囲へのズームができます。左ボタンでドラッグすると、プロット画面をスクロールすることができます。 
コマンドを使うことで、プロットの範囲が収まるようビューを再設定することもできます。 ♦

3D レンダリングと同様に、プロットはビューであると考えられます。どちらも別々の方法によってではありますが、データの表現方法を提供しています。そのため、3D ビューと同様の操作をプロットに対して行うことができます。オブジェクト・インスペクタの Display タブを見ると、色、線のスタイル、ベクトル成分、凡例など、この表現に関する様々な設定があることがわかります。



プロット・ビュー上端の  ボタンによって、ラベル、凡例、軸の数値範囲など、プロット全体の設定を変更するためのダイアログを呼び出すこともできます。



他のビューと同じように、File →  Save Screenshot を選ぶことで、プロットの画像を保存することができます。他のビューと同様に、プロットを移動させることもできます。

次の演習では、こういった機能を利用して、プロットからより多くの情報を引き出します。例えば、プロットを使って、圧力と温度の値を比較することが出来ます。

演習 2.12: 一連のプロットの表示設定

この演習は演習 2.11の続きです。この演習を始める前に、演習 2.11を完了させておく必要があります。

1. GUI 上でプロットを移動させたい場所を選択し、分割、削除、サイズ変更、位置の変更を使って、プロットをその場所に移動させます。

2. プロットをアクティブな状態にし、Display タブで Temp と Pres 以外の変数を全てオフにします。

Temp と Pres の変数は、異なる単位を持ちます。この 2 つを同じスケールで比べるのは、有用ではありません。2 つのプロット・ビューを使って、同じ線上の 2 つの変数の変化を比較してみましょう。[2]

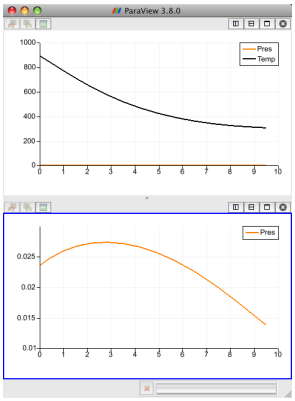
3. プロット・ビューを分割  して、現在存在している Line Chart View に加えて、新たな Line Chart View を作成します。

4. もしまだアクティブになっていなければ、Plot Over Line を Pipeline Browser で選択し、新たなビュー内で可視状態  にすることで、Plot Over Line をアクティブにします。

5. Object Inspector の Display タブで、Pres 変数だけを選択します。

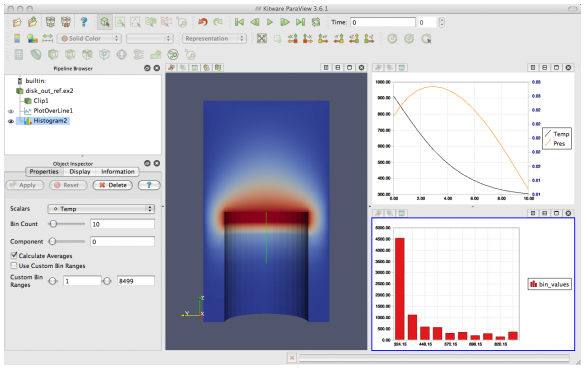
[2]原注: ParaView の 2 次元プロット機能は、バージョン 3.8 で大幅に再構築されました。残念ながら、2 つの変数を同じプロット上に異なったスケールでプロットする機能は失われてしまっています。この機能は、リリース 3.8.1 で復活する予定になっています。

6. Plot Over Line フィルタの可視状態を ON/OFF 操作して、座標軸をリセットします。



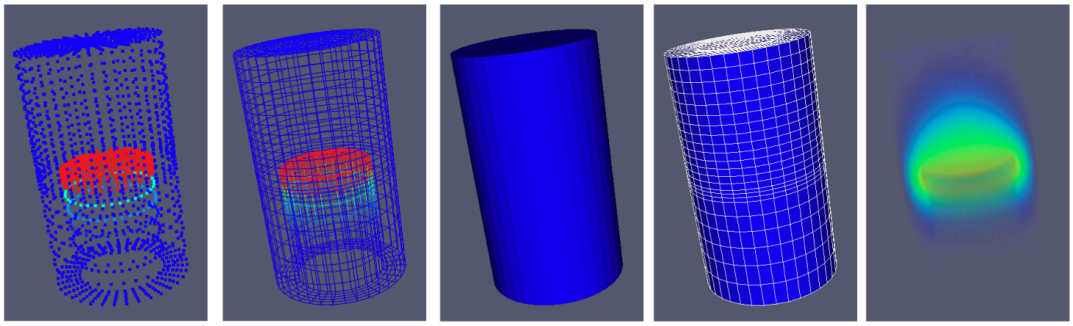
このプロットから、2.5 節で行った考察が正しいことが確認できます。温度がプレートの表面で最大値を取り、そこから離れるに従って下がるのに対し、圧力は少し上昇してから下降し始めます。さらに、圧力が最大値をとるのは (そして、空気中で力が釣り合うのは) 3 単位長さだけ離れた位置であることもわかります。 ♦

ParaView は、任意の数の異なった種類のビューに対応しています。これによって研究者や開発者は、新しい視点でデータを見ることができます。さらにもう一つのビューの例を示すために、パイプライン・ブラウザの disk_out_ref.ex2 を選択し、さらに Filters → Data Analysis → Histogram  を選択してみましょう。温度のヒストグラムを作成し、  ボタンをクリックします。



2.8 ボリューム・レンダリング

ParaView には、データの表現方法がいくつかあります。サーフェス、ワイヤフレーム、およびそれらの組合せといった、いくつかの例を既に見てきました。ParaView はさらに、データセット表面上の点、もしくは単純にバウンディング・ボックスを表示することもできます。



点　ワイヤフレーム　サーフェス　サーフェスとエッジ　ボリューム

それら表現方法の中でも**ボリューム・レンダリング**は、ParaView 上でデータを表現する上で強力なテクニックです。ボリューム・レンダリングでは、中実なメッシュが、メッシュ内の各点における色と濃度を表すスカラー場によって、半透明の雲のような塊としてレンダリングされます。サーフェス・レンダリングとは違い、ボリューム・レンダリングでは、ボリュームを通してデータセット内部の特徴に至るまで把握することができます。

ボリューム・レンダリングは、オブジェクトの表現方法を変更することで簡単に実現できます。温度の表示をボリューム・レンダリング表示に変えてみましょう。

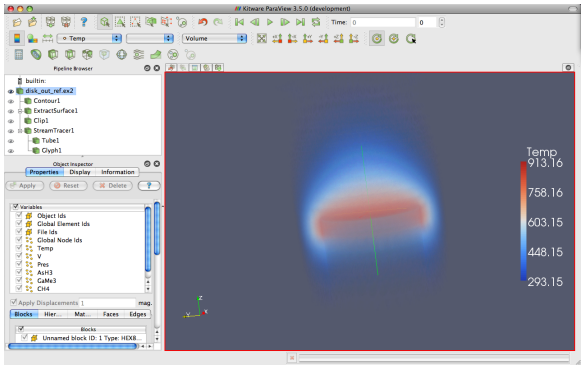
演習 2.13: ボリューム・レンダリングをオンにする

新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

1. disk_out_ref.ex2 ファイルを開き、全ての変数を読み込み、 

をクリックしてください (演習 2.4参照)。

2. パイプライン・ブラウザ上で、disk_out_ref.ex2 が選択されていることを確認してください。表示される変数を Temp に、表現方法を Volume にそれぞれ変更します。



不透明なメッシュの代わりに、半透明なボリュームが表示されました。画像を回転させる際に、パフォーマンスの関係でその画像が一時的に簡単な画像に置き換えられますが、これについての詳細は後ほど、3章に記します。 ♦

ParaView のボリューム・レンダリングの便利な点は、他のオブジェクトのサーフェス・レンダリングと同時に使用できることです。これによって対象のボリューム・レンダリングの周囲の状況を再現したり、また他の表示と組み合わせることで、より多くの情報を伝えることができます。例えば、温度のボリューム・レンダリングを、演習 2.9で行ったような流線ベクトルの可視化と組み合わせることができます。

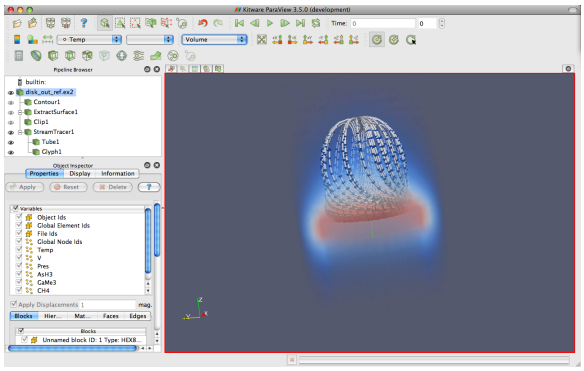
演習 2.14: ボリューム・レンダリングとサーフェス表現の可視化を組合わせる

この演習は演習 2.13の続きです。この演習を始める前に、演習 2.13を完了させる必要があります。

1. 流線追跡フィルタ  を disk_out_ref.ex2 に追加します。
2.  ボタンをクリックして、デフォルトの設定を適用します。

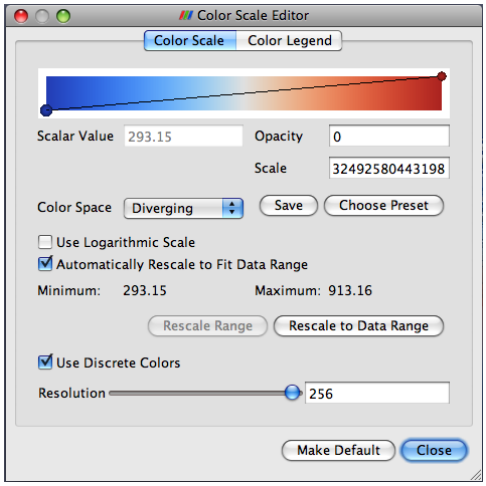
これで、ボリューム・レンダリングの中に、流線が埋め込まれているのが見える筈です。以下の追加の操作によって、演習 2.10でのように、流線に形状を追加して、流線を見やすくすることができます。これらは必須ではありませんので、省略することもできます。

3. クイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を使用して、Tube フィルタを適用し、  をクリックします。
4. 流線が Temp で色付けされていれば、Solid Color に変更します。
5. パイプライン・ブラウザで、StreamTracer1 を選択します。
6. グリフ・フィルタ  を StreamTracer1 に追加します。
7. オブジェクト・インスペクタで、Vectors の設定 (上から 2 つ目の設定) を V に変更します。
8. オブジェクト・インスペクタで、Glyph Type の設定 (上から 3 つ目の設定) を Cone に変更します。
9.  ボタンをクリックします。
10. Temp 変数によってグリフを色付けします。



これで、流線がボリューム全体に渡って温度場の中に表示されました。 ♦

デフォルトでは、ParaView はサーフェス・レンダリングで使われているのと同じ色を、値の範囲の最低値を不透明度 0、最高値を不透明度 1 に設定して描画します。ParaView では**伝達関数** (スカラー変数を表示する際の色や不透明度に関する設定) の変更も簡単に行えます。パイプライン・ブラウザ上でボリューム・レンダリングされたオブジェクトを選択した状態で、カラーマッピングの編集  ボタンをクリックしてください。



それによって現れるダイアログでは、伝達関数に関する設定を編集することができます。ダイアログ上部のグラデーションになったボックスは、色で伝達関数を、黒色の線プロットで不透明度を表しています。伝達関数上の丸い点は、**制御点**を表しています。制御点とは特定のスカラー値に設定された色と不透明度のことであり、制御点間の色と不透明度は補間値が与えられます。バー上の空白部分をクリックすると、新しい制御点が作成されます。既に有る制御点上でクリックすると、その点を選択します。選択された制御点は、ボックスの中をドラッグしてスカラー値と透明度を変更でき、選択された制御点上でもう 1 度クリックすることで新しくダイアログボックスが表示されます。選択された制御点は、Backspace キーもしくは Delete キーを押すと削除されます。

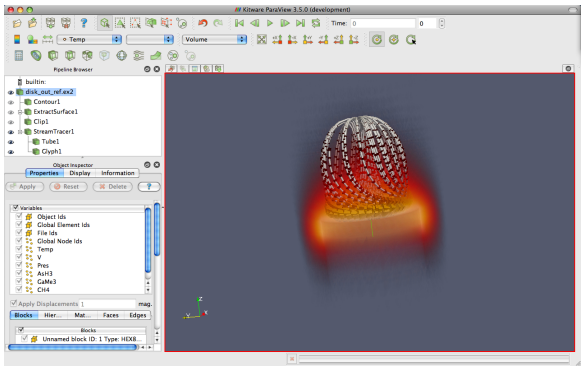
カラーバー直下には、テキスト入力ウィジェットがあり、選択された制御点の Scalar Value (スカラー値) と Opacity (不透明度) の数値を指定できます。Scale の設定は、不透明度の計算における単位長を調節します。数字が大きいほど、ボリュームが透明になります。Color Space の設定では、色の補間方法を変更できます。このパラメータは制御点の色には影響しませんが、制御点間の色には大きく影響します。Use Logarithmic Scale チェックボックスによって、色のスケールを対数尺度に変換することもできます。

伝達関数の設定は面倒になりがちですので、Save ボタンをクリックすることで設定を保存することができます。Choose Preset ボタンによって現れるダイアログでは、作成したカラー・マップや、ParaView に最初から用意されているカラー・マップを、管理および適用することができます。

演習 2.15: ボリューム・レンダリングの伝達関数を変更する

この演習は、演習 2.14の続きです。この演習を始めるには、演習 2.14 (または、少なくとも演習 2.13) を完了させておく必要があります。

1. パイプライン・ブラウザの disk_out_ref.ex2 をクリックし、アクティブにします。
2. カラー・マップの編集ボタン  をクリックします。
3. 制御点を追加・変更し、それらのボリューム・レンダリングへの影響を観察してください。
4. ボリューム・レンダリングを、さらに熱の表現にふさわしくします。Choose Preset をクリックし、ダイアログ・ボックスから Black-Body Radiation を選択し、OK をクリックします。



ボリューム・レンダリングのカラー・マッピングのみならず、全てのビューでの Temp のカラー・マッピングが変更されていることに注意してください。これによってビュー同士の一貫性を保証し、同じ変数を違う色や違う値の範囲でマッピングしてしまうことによる混乱を避けることができます。 ♦

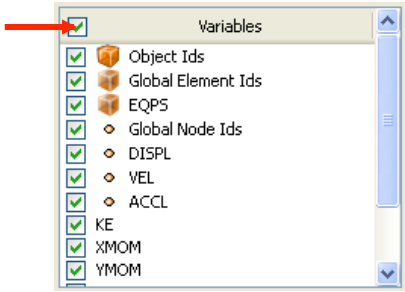
2.9 時間

ここまで disk_out_ref のシミュレーションを余すところなく分析して来ましたので、次は新しいシミュレーションに進んで、ParaView がどのように時間を扱っているのかを見てみましょう。この節では、簡単なシミュレーションによる、時間による変化のある新たなデータセットを使用します。

演習 2.16: 時間情報を持つデータを読み込む

新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

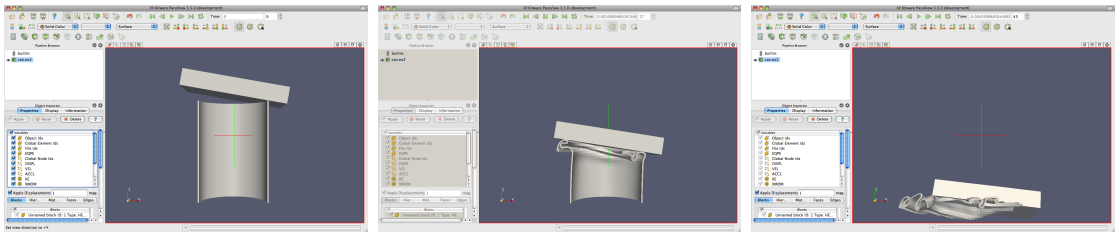
1. Open メニューから、can.ex2 を開きます。



2. 先程と同じように、変数リストの最上部のチェックボックスをクリックして、全ての変数が読み込まれるようにし、  ボタンを押します。

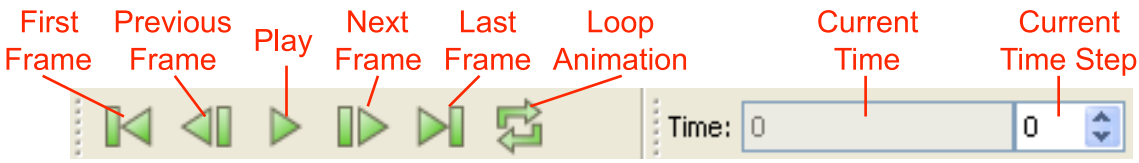
3.  ボタンを押して、カメラをメッシュに向けます。

4. 再生ボタン  を押して、落下するレンガによって缶が潰れる様子のメッシュが、ParaView によってアニメーションされる様子を見てみましょう。



♦

時間変化するデータを扱う際にすることは、これだけです。ParaView には時間の概念が組み込まれていて、データ内の時間と自動的にリンクするようになっています。時間を扱うためのツールバーに慣れてください。



アニメーションの保存も同様に簡単です。メニューから File → Save Animation を選択します。ParaView では、ダイアログでどのようにアニメーションを保存するかの設定ができ、そして自動的に時間を反復してアニメーションを保存します。

演習 2.17: 時間依存のデータの落とし穴

ユーザーが陥る最大の落とし穴は、値の範囲が時間変化する場合の色のマッピングです。例を見るために、以下を行ってください。

1. 演習 2.16から続けているのでなければ、can.ex2 を開き、全ての変数を読み込む設定とし、Apply をクリックして下さい。
2. 最初の時刻ステップ に移動します。
3. EQPS 変数をオンにします。
4. 色の凡例 をオンにします。
5. アニメーションを再生 (または、最後の時刻ステップ へ移動) します。

色付けは、あまり効果を発揮していません。簡単に問題を解決するには、以下を行います。

6. 最後の時刻ステップにいる状態で、データ範囲にスケールを調整する ボタンを押します。
7. もう一度、アニメーションを再生 して下さい。

これで、色付けが効果を発揮するようになりました。 ♦

これは一見、欠陥のように見えますが、そうではありません。これは 2 つの避けがたい要因のためです。1 つ目は、スカラー場の可視化を行うと、スケール範囲はその時刻ステップでのデータの値の範囲に合わせて設定されます。理想を言えば、スケールの範囲は、データの全ての時間を通しての最大値と最小値に設定されるべきです。

しかしながら、それを実現しようとすると ParaView は最初の読み込みで全てのデータをチェックすることになり、大規模データを処理する際にひどく動作が遅くなってしまいます。2 つ目は、時間を追ってアニメーションさせる際には、例えデータの値の範囲が変わってもスケールの範囲は固定である必要があるからです。アニメーションの再生に合わせて表示するスケールの範囲を変更することは、データを

誤って解釈する原因になります。スケールの範囲が暗黙に変動するよりは、当初設定したとおりのカラー・マッピングの範囲から逸脱する方が断然良いでしょう。とはいえ、この問題には以下のような回避方法があります。

- メニューの Edit → Settings (Mac では ParaView → Preferences) から、設定ダイアログ・ボックスを開きます。General タブで、On File Open の設定を Goto last timestep にします。この項目が選択されると、ParaView は時間を有するデータ・セットを読込むと、自動的に最後の時刻ステップに移動します。(can.ex2 のように) 多くのデータでは、フィールド・データの範囲は最初よりも最後のデータの方が代表的な範囲となります。したがって、時刻を変更する前にフィールド・データによって色付けする限り、色の範囲は適切なものとなります。

- もし、何らかの理由でアニメーションの色の範囲が不適切となった場合は、代表的な時刻ステップに移動して  を押します。

- あるいは、カラー・スケールのダイアログを開き  、データの範囲を指定します。これは、“代表的な” 時刻ステップを見つけられない、または判らないか、適切な範囲があらかじめ判っている場合に有効です。

- もし時間のかかる処理でも良ければ、カラー・スケールのダイアログ  の Rescale to Temporal Range ボタンを使うと、ParaView は全ての時間にわたる範囲を算出します。ただし、この設定では、データ・セットを読込むときには常に、全時刻ステップにわたる全てのデータ・セットが読込まれることに注意してください。ParaView が一度にメモリに保持するのは高々1 時刻ステップですが、大規模なデータ・セットにおいては、データをディスクからメモリへと引出すには長時間を要することがあります。

2.10 選択機能

可視化の最終目的はしばしば、大きな情報の実体から重要な細部を見出すことにあります。ParaView における選択の抽象化は、その過程の重要な簡略化です。選択とは、何らかのデータ・セットの一部を同定する行為と言うことができます。この選択を作成するには様々な方法があり、そのほとんどはユーザにとって直感的なものです。また、選択がひとたび作成されれば、その選択されたデータに対して特定の量を表示したり処理する様々な方法があります。

具体的には、1 つのデータ・セットの中の部分領域によって、特定の選択された点、セル、ブロックが同定されます。選択の中にどの要素を含めるかを指定する方法は複数あり、様々な方法によって作成される ID 番号のリスト、空間中の位置、スカラー値、スカラー値の範囲などがあります。

ParaView では、選択はいつでも行うことができ、現在の選択された領域が全てのビューで連動して保持されています。すなわち、いずれかのビューで何かを選択す

れば、その選択範囲は同じオブジェクトを表示する他の全てのビューにも表示されます。

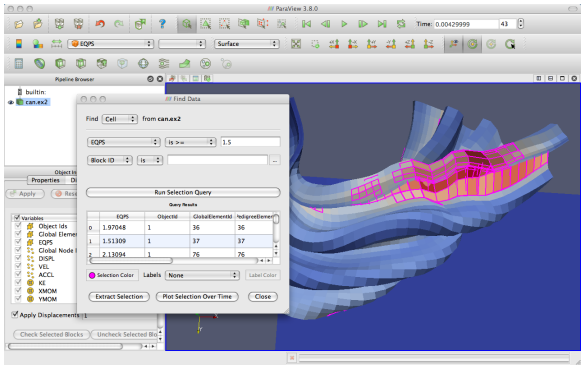
選択を作成する最も直接的な方法は、Find Data ダイアログを用いるものです。Edit メニューから、このダイアログを開きます。このダイアログに、探索したいデータの特徴を入力します。例えば、終端速度に近い速さを有する点を探すことができます。あるいは、材料の破壊限界を上回るひずみを有するセルを探すことができます。以下の演習では、Find Data ダイアログの簡単な使用例を示します。

演習 2.18: 問合せによる選択を行う

この演習では、等価塑性ひずみ (EQPS) の値が大きなセルを探索します。

1. もし、ここまでの演習で読込んでいなければ、can.ex2 ファイルを開き、全ての変数を読込む設定とし、 

をクリックしてください (演習 2.16を参照)。
2. 最後の時刻ステップに移動します  。
3. Edit → Find Data によってデータ探索ダイアログを開きます。
4. 最上部のコンボ・ボックスから、Cell の探索を選択します。
5. 次の行のウイジェットでは、最初のコンボ・ボックスから EQPS を、2 つ目のコンボ・ボックスでは is >= を選択し、最後のテキスト・ボックスに 1.5 を入力します。
6. Run Selection Query ボタンをクリックします。



Run Selection Query ボタンの下のスプレッドシートが、以上の問合せの結果で埋められていくのを確かめてください。それぞれの行はセルを表し、それぞれの列はフィールド値か (ID 番号のような) 属性を表します。

さらに、ParaView のウインドウ中の 3D ビュー中で、幾つかのセルがハイライトされているのにお気付きかもしれません。これらのハイライトは、以上の問合せによって作成された選択を表します。ここで Find Data ダイアログを閉じ、選択がそのまま残されていることを確かめてください。 ♦

選択部分を生成する最も簡単な方法は、要素をまさに 3D ビューの中から選び出すことです。3D ビューにおける全ての選択操作は、**ラバー・バンド**という選択機能によって行います。つまり、3D ビュー内でマウスをクリックしたりドラッグすることで、箱状の範囲内の要素を選択することができます。ラバー・バンドによる選択方法はいくつかの種類があり、選択設定のツールバーにあるアイコンのいずれかを選ぶか、ショートカット・キーのいずれかによってそのうちの一つを選ぶことができます。次の 3D ビューにおける選択方法が可能です。

 **Select Cells On (Surface) (表面上のセルを選択する)** ビュー内で見えるセルを選択します。(ショートカットは s)

 **Selects Points On (Surface) (表面上の点を選択する)** 視点内で見える点を選択します。

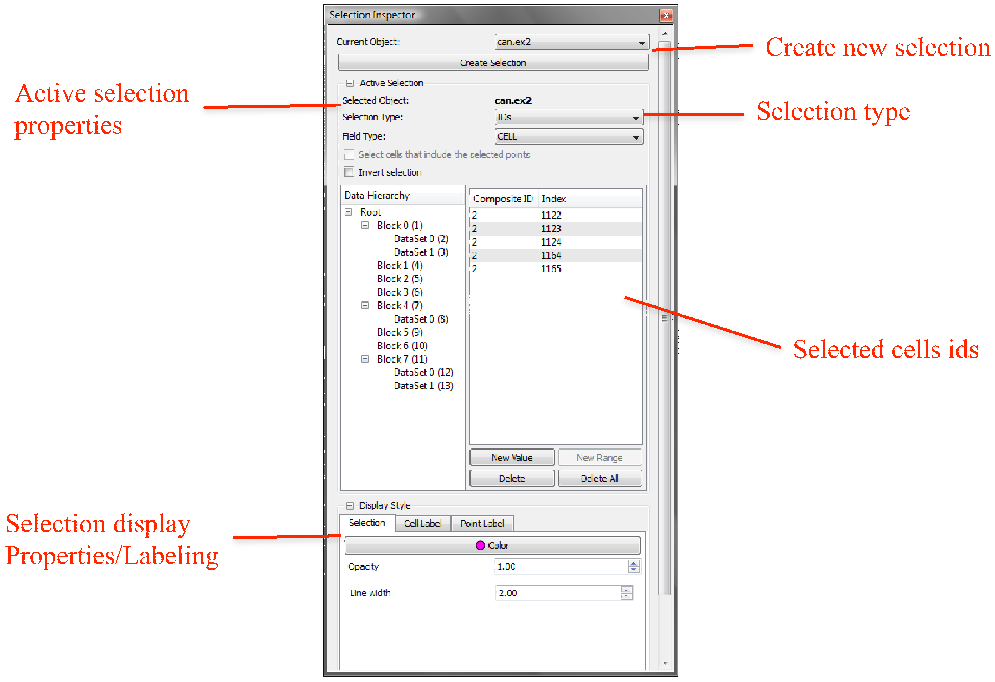
 **Select Cells Through (Frustum) (表面および内部のセルを貫通的に、錐台状に選択する)** ラバー・バンド内に存在する全てのセルを、視点から奥行方向に、対象オブジェクト内部のセルも含め貫通的に、錐台状に選択します。

 **Select Points Through (Frustum) (表面および内部の点を貫通的に、錐台状に選択する)** ラバー・バンド内に存在する全ての点を、視点から奥行方向に、対象オブジェクト内部の点も含め貫通的に、錐台状に選択します。

 **Select Blocks (ブロックを選択する)** マルチブロック・データセットから、ブロックを選択します。(ショートカットは b)

s と b のショートカットによって、それぞれセルやブロックを手早く選択できます。マウスカーソルを選択中の 3D ビュー上のどこかに置いてから、それらのキーを入力しましょう。そして、選択したいセルやブロックをクリックしましょう (あるいは複数の要素上にラバー・バンドをドラッグしましょう)。

ここで選択機能を様々に試してみて下さい。



選択状態は**セレクション・インスペクタ**で管理できます。セレクション・インスペクタは View → Selection Inspector で見ることができます。セレクション・インスペクタでは、選択されている全ての点やセルを見るほかにも、選択を修正することもできます。またセレクション・インスペクタを使って選択要素にラベルを付加し、要素の特定をしやすくすることができます。

セレクション・インスペクタを少し試してみましょう。Selection Inspector を開いてください。そしてラバー・バンドを使って選択範囲を作成し、Selection Inspector で結果を見てみてください。また ID を変更したり、Invert Selection のチェックボックスによって選択範囲を反転して、選択範囲を変更してみてください。

すると / の表面選択ツールは点/セルのリストを表示し、 のブロック選択ツールはブロックのリストを表示しますが、 / の貫通的選択ツールはどちらも表示していないことに気づくでしょう。これは、これらのツールが空間内の領域を選択しているからです。選択領域を見たい場合は、Show Frustum をクリックし、3D ビューを回転させてください。

ここで、一連の点やセルを特定する選択と、空間内の領域を特定する選択には、根本的な違いがある事を記しておくべきでしょう。この違いを理解するには、以下を試してください。

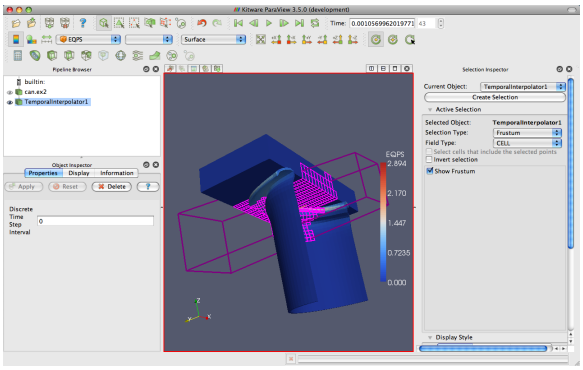
演習 2.19: データ要素を特定する選択と空間的な選択

1. もし、ここまでの演習で読み込んでいなければ、can.ex2 を開き、全ての変数を読み込む設定とし、 Apply をクリックします (演習 2.16を参照)。

2. Select Cells Through  ツールを使って、選択領域を生成してください。

3. もしまだ表示させていなければ、セレクション･インスペクタを View → Selection Inspector によって表示させて下さい。セレクション・インスペクタの画面の内容をよく見えるようにするには、どこか別の場所にドッキング、またはアンドッキングさせる必要があるかもしれません。

4. Selection Inspector の中の Show Frustum チェックボックスをクリックし、3D ビューを回転させます。



5. アニメーションを少しだけ再生 ▶ しましょう。このとき、選択領域は固定されたまま、どのセルが領域内あるいは領域外へ移動するかによって、選択要素が変化していることに注意して下さい。

6. Selection Inspector 内の Selection Type を IDs に変更してください。

7. もう一度再生 ▶ してください。今度は、選択されたセルが位置にかかわらず変化しないことに注意して下さい。

まとめると、(表面および内部のセルまたは点を選択するツールによって作成される) 空間的な選択では、要素が選択領域の内外へ移動するのに対し、それぞれの時刻ステップごとに選択を実行します。同様に、フィールド値の範囲による問合せもまた、データが変化するたびに再度実行されます。それに対し ID による選択では、選択された点やセルは変わらず、アニメーションに伴ってそれら自体の移動するのにつれて動きます。 ♦

スプレッドシート・ビューは選択と組み合わせて使い、分析を定量的に掘り下げるために重要なツールです。スプレッドシート・ビューでスカラー場の実際の数値を読むことができ、また選択の仕組みも活用することで目的の数値を特定しやすくなります。

演習 2.20: スプレッドシート・ビューと選択

1. もし、ここまでの演習で読み込んでいなければ、can.ex2 を開き、全ての変数を読み込む設定とし、 Apply をクリックします (演習 2.16を参照)。

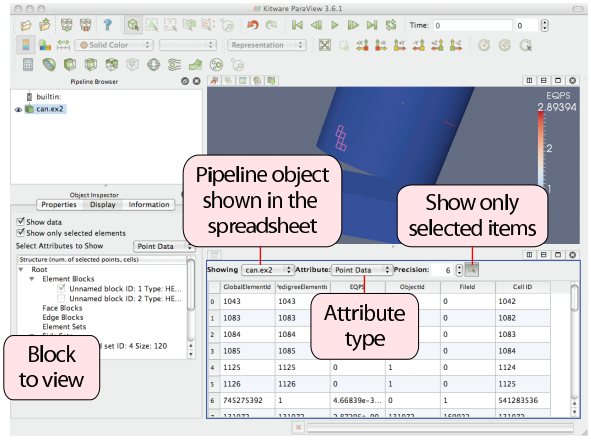
2. 垂直方向にビューを分割 して下さい。

3. 新しいビューにて、Spreadsheet View ボタンをクリックしてください。

4. もし can.ex2 が見えていなかったら、(対応する をクリックして) 見えるようにしてください。

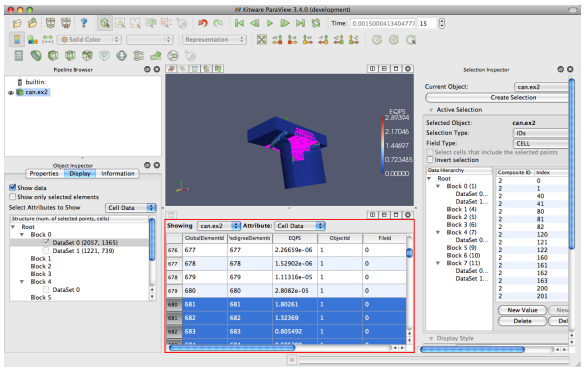
お分かりのとおり、スプレッドシート・ビューはかなり簡潔で、フィールド・データを (例えば、点データまたはセル・データといった) フィールドの型ごとに、表形式で表示します。このフィールド型ごとに表示される制約は、全ての行が確実に同じ列数のデータを含むようにするために設けられています。

スプレッドシート・ビューの上部のウイジェットに注目して下さい。これらによって、パイプライン・オブジェクトの選択、表示するフィールド型の選択、浮動小数点数値の精度の選択、そして選択以外の全てを隠すことが素早く行えるようになっています。どのビューにせよ、スプレッドシート・ビューにはそれ自身の Display パネルがついてきます。



5. Attribute のコンボ・ボックスから Cell Data を選びます。

6. スプレッドシート・ビューをスクロールして、ハイライトされた行を探します。(Display パネルにて、異なるブロックを選択しなければならないかもしれません。)



それらのハイライトされた行は、現在選択中の要素の一部です。このようなビュー間の選択の連携は、複数のビューを関連付ける重要な仕組みです。とはいえ、この例では、スプレッドシート・ビューにおいて選択された要素を識別するのは難しいかもしれません。時には、選択されたデータだけを見たいこともあります。

7. スプレッドシート・ビュー上部の Show only selected elements をクリックしてオンにしてください。

3D ビューにおいて作成された選択に含まれる要素のみが、スプレッドシート・ビューに現れるのが見て取れました。この関連付けは、逆方向にも働きます。すなわち、スプレッドシート・ビューで選択を作成すると、それらは 3D ビューにも表示されます。

1. Show only selected elements のチェックを外します。
2. スプレッドシート・ビューで 2〜3 行を選択します。
3. 3D ビューで選択された結果を探します。

♦

スプレッドシートによって、フィールド・データを最も読みやすい形で表示することができます。しかしながら、3D ビューに直接フィールド・データを表示するのが便利であることもあります。次の演習では、そのための方法を解説します。

演習 2.21: 選択にラベルを付ける

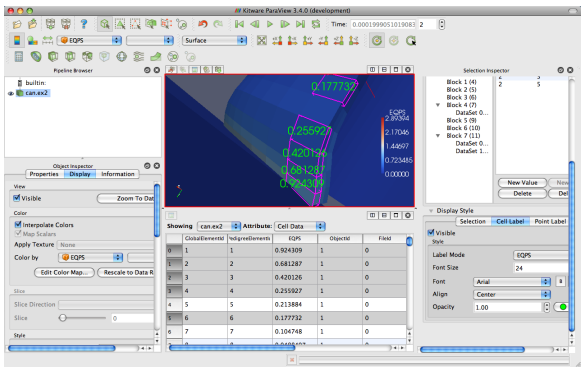
1. もし、ここまでの演習で読み込んでいなければ、can.ex2 を開き、全ての変数を読み込む設定とし、Apply をクリックします (演習 2.16を参照)。
2. もし、ここまでの演習で幾つかのセルを選択していなければ、ここで幾つかを選択して下さい。(この演習では、大量のセルを選択する必要はありません。)

3. まだ表示させていなければ、View → Selection Inspector によってセレクション・インスペクタを表示させて下さい。

4. Selection Inspector 内の (下部にある) Cell Label タブをクリックします。

5. Visible のチェックを入れます。

6. Label Mode を EQPS に変更します。



この操作によって、EQPS フィールドの値が、その値を含んでいる選択されたセルの近くに表示されます。セレクション・インスペクタによって、フォントの種類、大きさ、色を変更することも可能です。 ♦

ParaView では、フィールド・データを時間に沿ってプロットできます。全部のデータを全ての時刻ステップでプロットしたいことは滅多にないでしょうから、これらのプロットは選択部分に対して行われるようになっています。

演習 2.22: 時間に沿ってプロットする

1. もし、ここまでの演習で読み込んでいなければ、can.ex2 を開き、全ての変数を読み込む設定とし、 

をクリックします (演習 2.16を参照)。

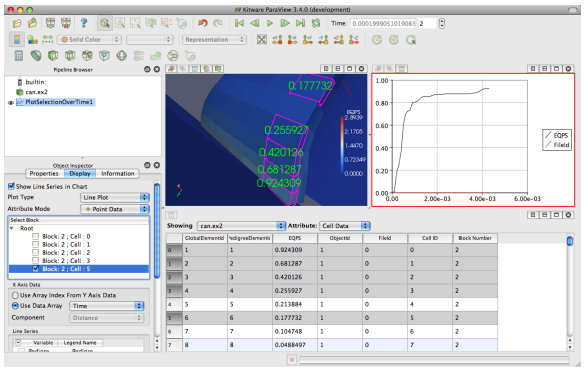
2. もし、ここまでの演習で幾つかのセルを選択していなければ、ここで幾つかを選択して下さい。(この演習では、大量のセルを選択する必要はありません。)

3. 選択が有効な状態で、Plot Selection Over Time フィルタを追加してください (Filters → Data Analysis → Plot Selection Over Time  か、Windows/Linux では Ctrl+Space、Mac では Alt+Space のクイック起動を使用して下さい)。

4. 

をクリックします。

5. Display パネルで、プロットする別のブロック (選択した要素のそれぞれと一致します) を選択してください。

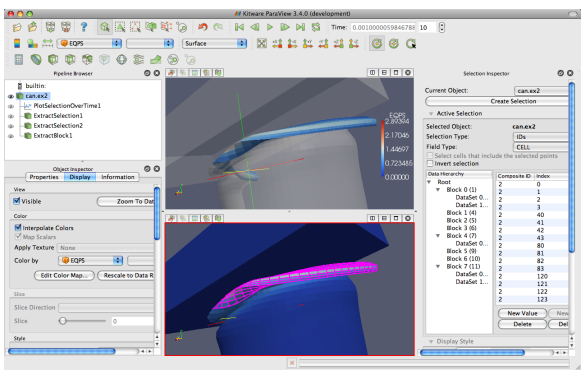


Object Inspector にて、選択していた部分がプロットすべき選択として自動的に追加されていることに注意して下さい。選択を変更するには、新たな選択を行って Object Inspector の Copy Active Selection をクリックします。 ♦

また、選択された点やセルを個別に見たり、それらについてのみ何らかの処理を行うために、選択部分を抽出することもできます。これは Extract Selection フィルタによって行われます。

演習 2.23: 選択の抽出

1. もし、ここまでの演習で読み込んでいなければ、can.ex2 を開き、全ての変数を読み込む設定とし、 Apply をクリックします (演習 2.16を参照)。
2. もし、まだ表示されていれば、セルのラベル表示をオフにします (Selection Inspector を確認して下さい)。
3. 内部も含めたセルの選択 (Select Cells Through) で、多めのセルを選択します。
4. Extract Selection フィルタを適用します (Filters → Data Analysis → Extract Selection か、Windows/Linux では Ctrl+Space、Mac では Alt+Space のクイック起動を使用して下さい)。
5. Apply をクリックします。

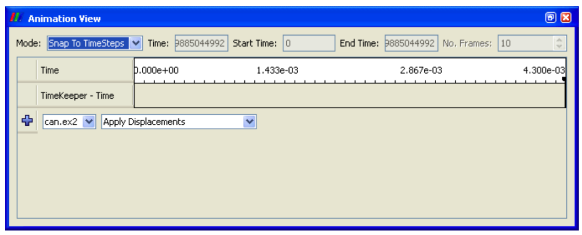


ビュー内のオブジェクトは、選択したセルに置き換えられます。(この画像では、抽出したセルを全体のデータと関連させて見せるために、半透明な表面と、もとの選択部分を示すもう一つのビューを加えています。) フィルタを選択部分を抽出したパイプライン・オブジェクトに単に付加するだけで、抽出されたセルに対しての演算を行うことができます。 ♦

これで選択の演習を完了しましたので、Selection Inspector を今後使うことはありません。必要なら、閉じても構いません。

2.11 時間の制御

ParaView には、時間とアニメーションを制御する強力な設定が多数あります。これらの大多数は、**アニメーション・ビュー**を通じて利用します。メニューから、View → Animation View をクリックします。



まず、アニメーション・ビューの上部にあるウイジェットを試してみましょう。アニメーションの**モード**設定は、ParaView がどのように再生中に時間を進めるかを定義します。3 つのモードを利用できます。

Sequence (シーケンス) 開始、終了時刻を与えれば、アニメーションを決められたフレーム数に等間隔に分割します。

Real Time (リアル・タイム) ParaView は決められた秒数の再生時間となるよう、アニメーションを再生します。実際のフレーム数は、フレーム間の更新時間に依存します。

Snap To TimeSteps (時刻ステップにスナップする) ParaView はデータで定義された時刻ステップどおりに再生します。

時間を含んだファイルを読み込むときは常に、ParaView は自動的にアニメーション・モードを Snap To TimeSteps に変更します。そのためデフォルトでは、データを読込んで、再生 ▶ を押せば、データで定義されたとおりの各時刻ステップを見ることができます。これが最も一般的な使い方です。

それ以外の使い方は、シミュレーションの書出したデータの時刻ステップ間隔が、可変であるときに発生します。このような場合にはおそらく、時間のインデックスよりも、シミュレーション上の時間に従ってアニメーションを再生したいでしょう。

問題ありません。残り 2 つのアニメーション・モードのうちの 1 つに切り替えられます。別の使い方は、再生速度を変えたい場合です。アニメーションの速度を速くしたり、遅くしたいことがあるでしょう。残り 2 つのアニメーション・モードでは、それができます。

演習 2.24: アニメーション・モードによって、アニメーションの再生速度を下げる

新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

1. can.ex2 を開き、全ての変数を読み込む設定とし、 

をクリックします (演習 2.16を参照)。
2.  ボタンをクリックして、カメラをメッシュに向けて下さい。
3. ツールバーの再生ボタン  をクリックして下さい。

アニメーションの間、ParaView は元のデータに定義されたそれぞれの時刻ステップを、ただ一度だけ読み込んで表示します。アニメーションが再生される速度に注意して下さい。

4. もし、まだ表示させていなければ、View → Animation View でアニメーション・ビューを表示させて下さい。
5. アニメーション・モードを Real Time に変更して下さい。デフォルトでは、アニメーションはデータで指定されたとおりの時間の範囲となり、再生時間は 10 秒間になっています。
6. もう一度、アニメーションを再生  します。

結果は前の Snap To TimeSteps によるアニメーションに似ていますが、今回はシミュレーション上の時間に比例してアニメーションが進行し、10 秒間で完了します。

7. Duration を 60 秒に変更して下さい。
8. もう一度、アニメーションを再生 します。

アニメーションは明らかに、ゆっくりと再生されます。使用しているコンピュータでの画面更新が遅くなければ、アニメーションが前に比べて断続的になっていることにも気づくでしょう。これはデータセットの時間解像度を上回ったからです。♦

元のデータに起因する、断続的なアニメーションの表示は多くの場合、望ましい挙動と言えます。データの中でまさにどれが表示されているかが分かるからです。

しかし、プレゼンテーション用にアニメーションを作りたいのなら、より滑らかなアニメーションが求められるでしょう。

ParaView には、このために作られたフィルタが存在します。それは**テンポラル・インターポレータ**と呼ばれています。このフィルタは、メッシュ座標等の位置的データおよびフィールド・データに対して、元のデータセットで定義された時刻ステップの間を補間します。この機能は、最近の ParaView と VTK パイプライン構造の進歩によって可能になりました。

演習 2.25: テンポラル・インターポレータ

この演習は、演習 2.13の続きです。この演習を始める前に、演習 2.13を完了しておく必要があります。

1. パイプライン・ブラウザの can.ex2 がハイライト表示されていることを確認して下さい。
2. Filters → Temporal → Temporal Interpolator を選択するか、クイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を使用して Temporal Interpolator フィルタを適用します。
3.  をクリックして下さい。
4. 必要なら、Animation Inspector で Real Time モードに戻って下さい。
5. ビューを分割  し、一方に TemporalInterpolator1 を、他方に can.ex2 を表示し、カメラをリンクして下さい。
6. アニメーションを再生  して下さい。

テンポラル・インターポレータの出力の方が、元のデータよりも大幅にスムーズに再生されることがお判り頂けるでしょう。 ♦

テンポラル・インターポレータの使用によって、不自然な結果を招きうる (そして、それは実際、しばしば起きる) ことは指摘しておく必要があるでしょう。ParaView がこの種の補間を決して自動的に行わないのはそれが理由であり、Temporal Interpolator は明示的に適用する必要があります。また、一般的には、メッシュの変形は多くの場合上手く補間されますが、一定のメッシュの下で移動する場は上手く補間されません。さらに、Temporal Interpolator は、トポロジが変化しない場合にのみ動作することにも注意して下さい。もし、メッシュが時刻ステップによって変化する適応メッシュの場合は、Temporal Interpolator はエラーとなります。

2.12 テキストによる注釈

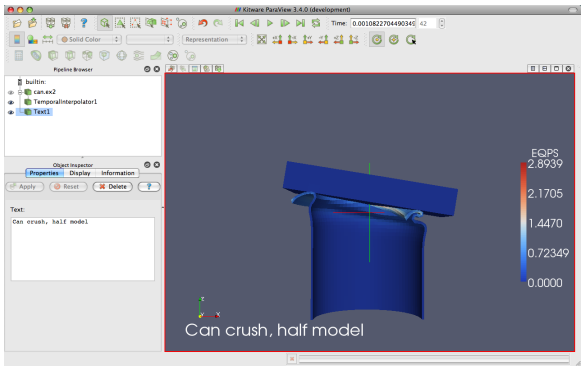
ParaView をコミュニケーションのための道具として用いるとき、作成した画像にテキストの注釈をつけることは時に役に立ちます。ParaView では、3D ビューにいつでも好きな時に注釈をつけることが、とても簡単にできます。ビュー内に単にテキストを配置する特別な**テキスト・ソース**があります。

演習 2.26: テキストによる注釈の追加

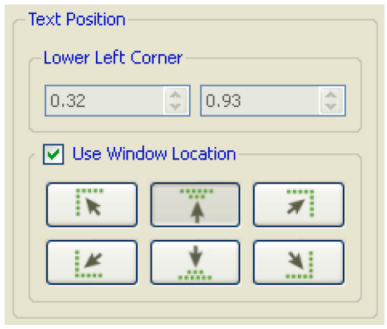
この演習を演習 2.25の続きとして行っているのであれば、そのまま続けて頂いて構いません。もし、ParaView を再起動したり、ParaView の状態が演習を続けるに相応しくなければ、新たな状態から始めて頂いても構いません。

1. メニューバーから、Sources → Text を選択します。
2. オブジェクト・インスペクタのテキスト入力欄に、適当な文を入力します。
3. 

ボタンをクリックします。



入力したテキストは、3D ビュー内に現れます。マウスでドラッグするだけで、好きなところにこのテキストを配置できます。オブジェクト・インスペクタの Display タブでは、テキストのサイズ、フォント、色の追加設定ができます。また、テキストを最もよく使う位置に配置するためのボタンもあります。

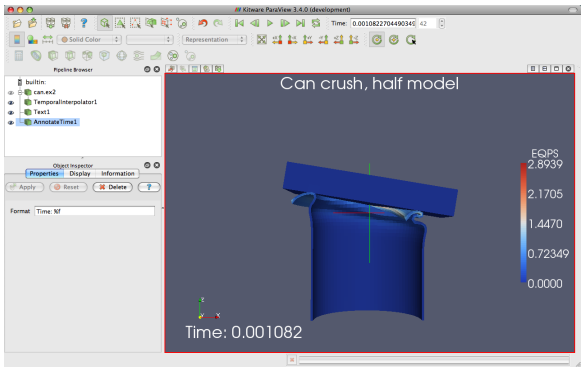


♦

しばしばアニメーション上の現在の時刻の値を、テキスト注釈に入れる必要があるでしょう。標準のテキスト・ソースで現在の時間の値を入力するのは退屈で、間違いを起こしやすく、さらにアニメーションを作る時は、これが不可能です。そこで、現在のアニメーションの時刻を文字列に挿入する特別な**アノーテート・タイム・**ソースがあります。

演習 2.27: アノーテート・タイムを追加する

1. 前の演習から読み込んでいなければ、can.ex2 ファイルを開き、Apply をクリックします。
2. Animate Time ソースを追加します (Sources → Annotate Time か、Windows/Linux では Ctrl+Space、Mac では Alt+Space のクイック起動を使用します)。
3. 必要ならば、注釈の位置を適宜移動します。
4. 再生 ▶ し、時刻の注釈がどう変化するか観察します。



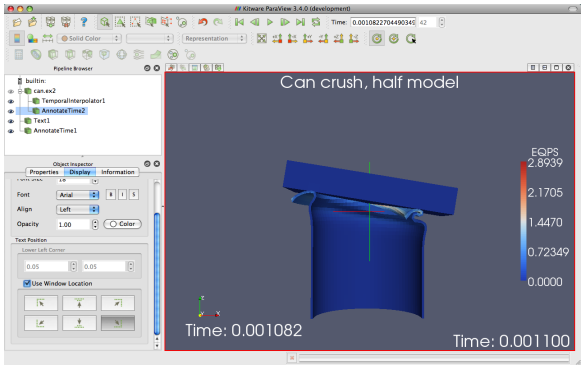
現在のアニメーションの時刻が、データ・ファイルから読み込まれた時刻ステップと同じでない場合があります。時には、データ・ファイルに保存されている時刻がいつであるかを知ることが重要なことがありますが、そのような場合には、フィルタとして機能するアノーテート・タイムの特別版を使用することができます。

5. can.ex2 を選択します。
6. クイック起動 (Windows/Linux では Ctrl+Space、Mac では Alt+Space) を使用し、Annotate Time フィルタを適用します。
7. Apply をクリックします。

8. 必要ならば、注釈の位置を適宜移動します。

9. Animation View で、アニメーション・モードを確認して下さい。もし Snap to TimeSteps になっていれば、Real Time に変更して下さい。

10. 再生 ▶ をクリックして、時刻の注釈がどう変化するか観察して下さい。



♦

2.13 アニメーション

私たちは既に、時間を含むデータ・セットをアニメーションさせる方法を見てきました (▶ を押します)。しかし、ParaView のアニメーション能力はそれ以上のものです。ParaView では、全てのパイプライン・オブジェクトのほぼ全てのプロパティをアニメーションさせることができます。

演習 2.28: プロパティをアニメーションさせる

新たな可視化を始めますので、ここまでの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

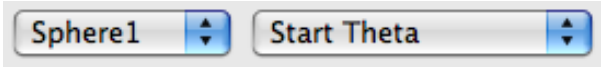
1. スフィア・ソースを作成 (Sources → Sphere) し、 

をクリックします。

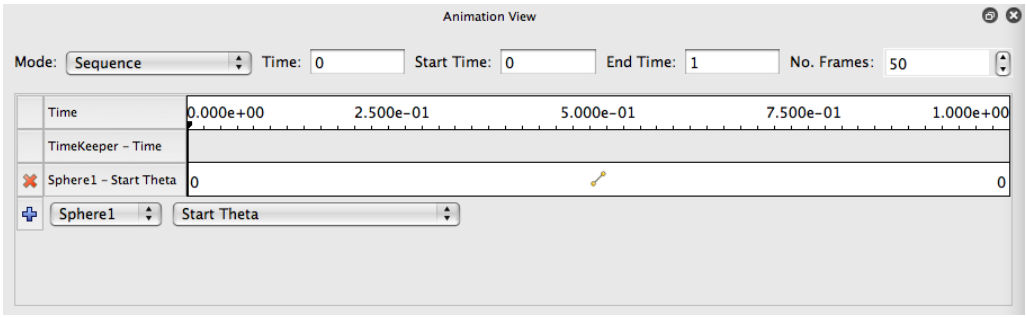
2. アニメーション・ビュー・パネルが表示されていることを確認して下さい (表示されていなければ、View → Animation View を選択して下さい)。

3. No. Frames の設定を 50 に変更します (10 では速すぎるでしょう)。

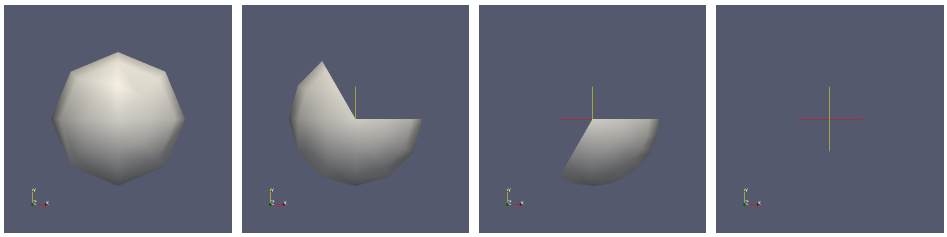
4. アニメーション・ビュー下部のプロパティ選択ウィジェットで、1 つ目のボックスで Sphere1 を、2 つ目のボックスで Start Theta を選択します。




ボタンをクリックします。



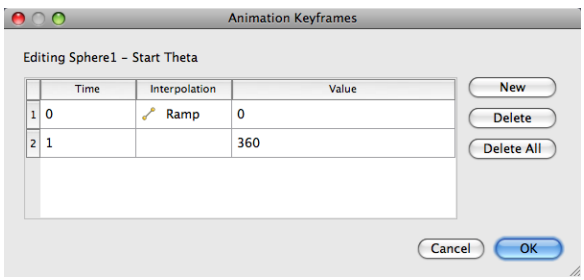
このアニメーションを再生  させると、球が展開して、ついには開ききって消滅するのが見られるでしょう。



♦

ここで行ったことは、Sphere1 オブジェクトの Start Theta プロパティのための**トラック**を作成することです。トラックは、アニメーション・ビュー内の水平のバーで表されます。そのバーには、ある特定の時刻における、そのプロパティの値を指定するための**キー・フレーム**が保持されています。そのプロパティの値は、キー・フレームの間で補間されます。トラックを作成すると、開始時刻における最小値と終了時刻に置ける最大値を保持するそれぞれのキー・フレームが自動的に作成されます。ここで設定したプロパティによって、球の開始角度の範囲が決まります。

トラックはその上でダブルクリックすることで、修正できます。ダブルクリックするとダイアログ・ボックスが表示され、キー・フレームを追加、削除および修正することができます。

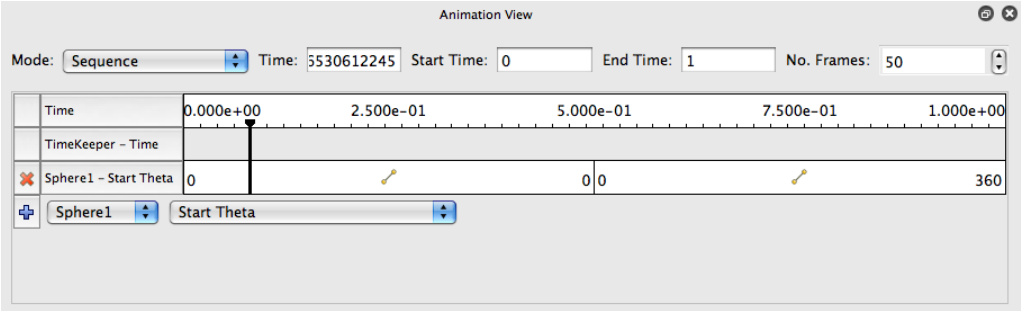


この機能を、アニメーションに新たなキー・フレームを追加するのに使います。

演習 2.29: アニメーション・トラックのキー・フレームを変更する

この演習は、演習 2.28の続きです。この演習を始める前に、演習 2.28を完了させておく必要があります。

1. Sphere1 – Start Theta トラックをダブルクリックします。
2. Animation Keyframes ダイアログ内の、New ボタンをクリックします。これで新たなキー・フレームが作成されます。
3. 1 番目のキー・フレームの値を 360 に変更します。
4. 2 番目のキー・フレームの時刻を 0.5 に変更し、値を 0 に変更します。
5. OK をクリックします。



アニメーションを再生すると、球ははじめ大きくなり、その後また小さくなります。 ♦

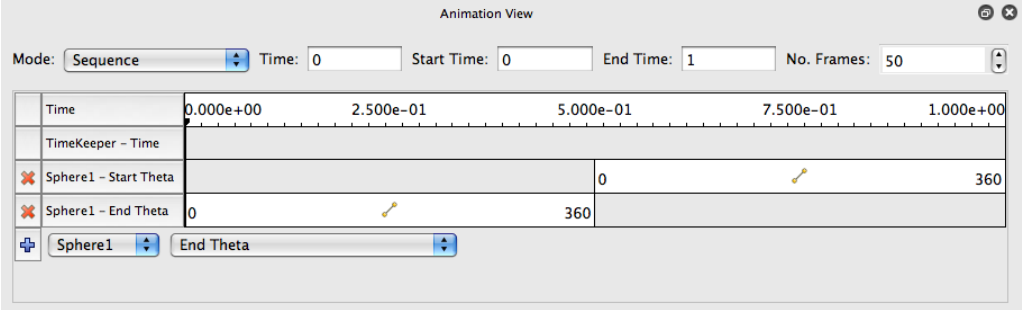
アニメーションさせられるのは 1 つのプロパティだけではありません。必要なだけの数のプロパティを、アニメーションさせることができます。ここでは、2 つのプロパティを変更するようなアニメーションを作成してみましょう。

演習 2.30: 複数のアニメーション・トラック

この演習は演習 2.28と演習 2.29の続きです。この演習を始める前に、これらの演習を完了させておく必要があります。

1. Sphere1 – Start Theta トラックをダブルクリックします。
2. Animation Keyframes ダイアログで、(時刻ステップ 0 の) 最初のキー・フレームを、Delete ボタンをクリックして削除します。
3. OK をクリックします。
4. アニメーション・ビューで、Sphere1 オブジェクトの End Theta プロパティのためのトラックを作成します。
5. Sphere1 – End Theta トラックをダブルクリックします。

6. 2 番目のキー・フレームの時刻を 0.5 に変更します。



このアニメーションでは、球が生成・消滅するのが見られますが、今回は球形状の描画範囲の変化部分が同じ方向に回転しています。それによって、このアニメーションをループ させたときに、とても満足できるアニメーションとなっています。 ♦

パイプライン・オブジェクトのプロパティをアニメーションさせるのに加えて、カメラをアニメーションさせることもできます。ParaView には、指定された曲線に沿ってカメラを動かす機能があります。最も一般的なアニメーションは、カメラをオブジェクトの周りで、常にオブジェクトの方向を向けながら回転させるものでしょう。ParaView には、そのようなアニメーションを自動的に生成する方法があります。

演習 2.31: カメラを周回させるアニメーション

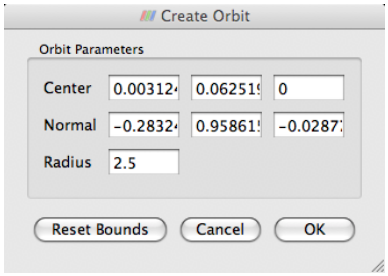
この演習では、何らかの読み込まれたデータに対してカメラを周回させます。もし以前の演習からの続きでこの演習を行うのであれば、そのまま続けることができます。そうでなければ、何らかのデータを読み込むか作成して下さい。効果を確認するには、読み込むデータが非対称な形状であることが最適です。can.ex2 は、この演習のために読み込むデータセットとして良いデータでしょう。

1. カメラを周回を始めたい点に設定します。なお、このあとカメラは、視点の右側に移動して行きます。

2. アニメーション・ビュー・パネルが表示されていることを確認して下さい (表示されていなければ、View → Animation View を選択して下さい)。

3. プロパティ選択のウイジェットで、最初のコンボ・ボックスでは Camera を、2 番目のコンボ・ボックスでは Orbit を選択します。

ボタンをクリックします。



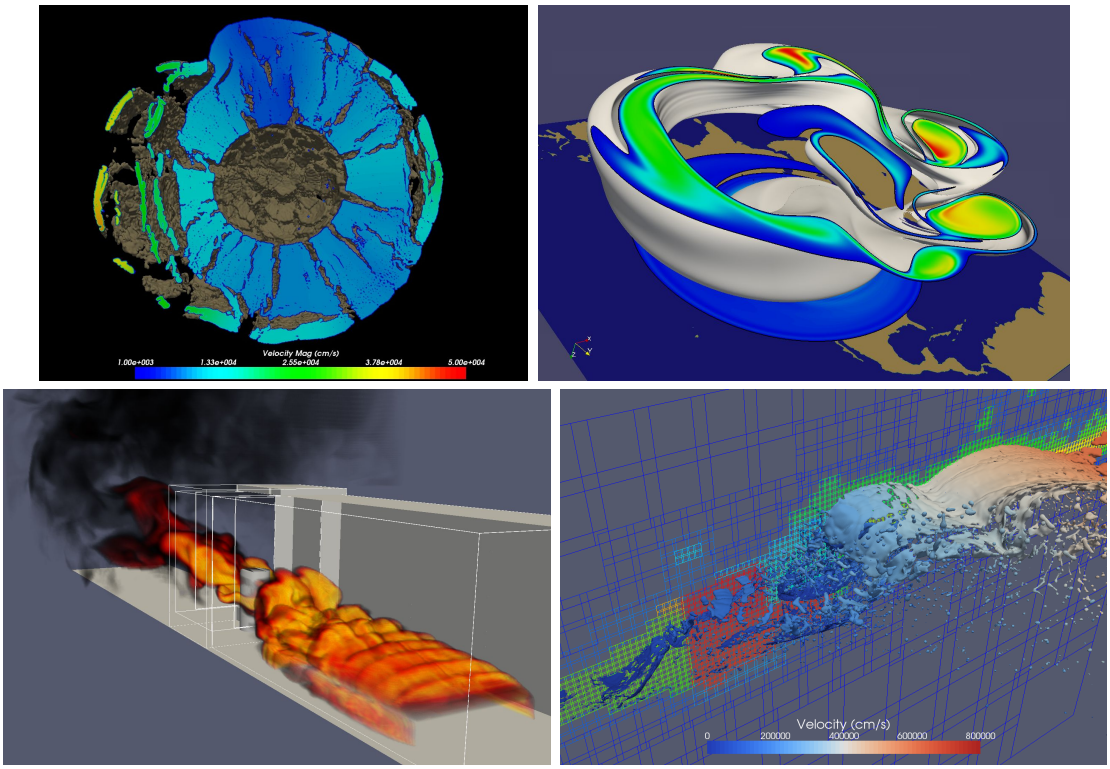
新たなトラックが作成される前に、周回軌道の設定ダイアログ・ボックスが表示されます。デフォルトの値は現在のカメラ位置から決定され、通常これらの値のままで構いません。

4. OK をクリックして下さい。

5. 再生 ▶ をクリックして下さい。

これで、カメラがオブジェクトの周りを周回します。 ♦

第3章　大規模モデルの可視化



サンディア国立研究所では、ここに示した例のような Red Storm スーパーコンピュータで実行された大規模シミュレーションのデータを可視化するために、ParaView を頻繁に使用しています。左上の画像は、ゴレブカ小惑星の中心で起爆した 10 メガトン爆発の、10 億セル以上の規模での CTH による衝撃波シミュレーションです。右上の画像は、高緯度で極大気を捕捉する極周囲のジェット気流である極渦の崩壊の、10 億セル以上の規模での SEAM 気候モデルシミュレーションです。左下は、横風火災における物体の、1000 万の非構造 6 面体格子による SIERRA/Fuego/Syrinx/Calore を用いた弱連成シミュレーションです。右下は、AMR データを生成する CTH によるシミュレーションです。ParaView は、数十億セル、数十万ブロック、そして 11 レベルの階層構造からなる CTH のシミュレーションによる AMR データの可視化に使用されてきました (ここでは示しませんが)。

本節では、ParaView の並列可視化機能を用いて、このような大規模メッシュを可視化する方法を解説します。本節は前節ほど実践的ではありません。その代わり、大規模な並列可視化を行うための概念的な知識を学んで頂くことになります。ここで

は基本的な ParaView の構造および並列アルゴリズムを示し、これらの知識をどのように利用するかを実演します。

3.1 ParaView の構造

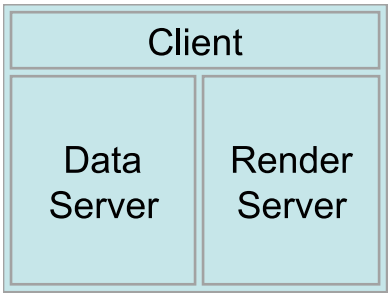
ParaView は 3 層のクライアント・サーバ構造になっています。ParaView におけるそれら 3 つの論理的なユニットは、以下のとおりです。

データ・サーバ データの読込み、フィルタリング、書出しを担当するユニットです。パイプライン・ブラウザに表示される全てのパイプライン・オブジェクトは、データ・サーバが保持しています。データ・サーバは並列実行が可能です。

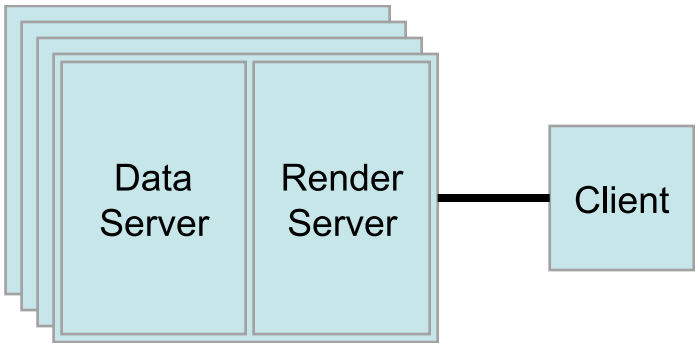
レンダー・サーバ レンダリングを担当するユニットです。レンダー・サーバも並列実行が可能で、その場合は内蔵の並列レンダリング機能が有効化されます。

クライアント 可視化の作成を担当するユニットです。クライアントはサーバにおけるオブジェクトの作成、実行、削除を制御しますが、実際のデータは全く保持しません (そのため、クライアントがボトルネックとなることが無く、サーバが大規模データへとスケーリングすることができます)。GUI もまた、クライアントに含まれます。クライアントは常にシリアルに実行される、非並列化アプリケーションです。

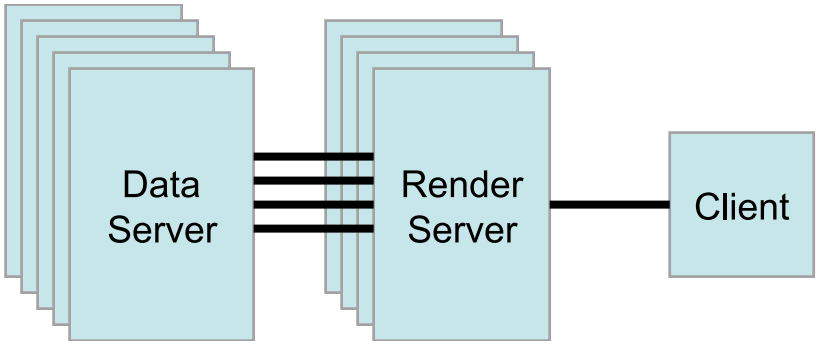
これらの論理ユニットは、物理的に分割されている必要はありません。論理ユニットは多くの場合、同一のアプリケーションに含まれ、それらのユニット間での通信を不要としています。ParaView は、以下の 3 つのモードで実行されます。



最初のモードは**スタンドアローン**・モードで、既に使用してきたとおりのモードです。スタンドアローン・モードでは、クライアント、データ・サーバ、レンダー・サーバは全て一つのシリアル実行アプリケーションに統合されています。`paraview` アプリケーションを実行すると、ParaView の全ての機能をすぐに使えるよう、**ビルトイン**・サーバに接続されています。



2 つ目のモードは**クライアント－サーバ**・モードです。クライアント－サーバ・モードでは、pvserver プログラムを並列マシン上で実行し、それに paraview クライアント・アプリケーションを接続します。pvserver プログラムはデータ・サーバとレンダー・サーバの両方を含んでおり、したがってデータ処理とレンダリングの両方が pvserver で実行されます。クライアントとサーバはソケットを通じて接続します。ソケット通信は比較的低速な通信手法であるため、このソケットを通じたデータ転送量は最小に抑えられています。



3 つ目のモードは、**クライアント－レンダー・サーバ－データ・サーバ**・モードです。このモードでは、全ての 3 つの論理ユニットが個別のプログラムとして実行されます。クライアント－サーバ・モードと同様、クライアントはレンダー・サーバに 1 つのソケットによって接続されます。レンダー・サーバとデータ・サーバはレンダー・サーバのプロセスそれぞれにつき 1 つづつ、多数のソケットによって接続されます。ソケットを通じたデータ転送量は最小に抑えられています。

クライアント－レンダー・サーバ－データ・サーバ・モードはサポートされてはいますが、このモードはまず推奨されません。このモードの元々の意図は、大規模でパワフルな計算プラットフォームと、小規模なグラフィックス・ハードウエアが内蔵された並列マシンからなる異種混在環境を活用することでした。しかしながら実際には、データ・サーバからレンダー・サーバへの転送所要時間によって、あらゆる利点が相殺されてしまうことが判りました。もし計算プラットフォームのほうがグラフィックス・クラスタより非常に大きい場合は、ソフトウエア・レンダリングを使用して下さい。もし両プラットフォームが概ね同じ規模である場合は、グラフィックス・クラスタで全ての処理を実行して下さい。

3.2 ParaView サーバのセットアップ

スタンドアローンの ParaView をセットアップするのは通常、難しくありません。コンパイル済みのバイナリをダウンロードし、コンピュータにインストールすれば、すぐに実行可能です。しかしながら、ParaView サーバのセットアップはどうしても、それよりは難しくなります。まず、サーバを自前でコンパイルしなければなりません。並列プログラミングのためのライブラリである MPI には様々な種類のものがあり、さらに個々の MPI は並列コンピュータの通信ハードウエアに合わせて変更が可能であるため、全ての考えうる組合せについて信頼できるバイナリファイルを提供するのは不可能です。

ParaView を並列マシンでコンパイルするには、以下が必要です。

- CMake クロスプラットフォーム・ビルド・セットアップ・ツール (www.cmake.org)
- MPI
- OpenGL (または、他に利用可能なものが無ければ Mesa 3D www.mesa3d.org)
- Qt 4.6 (オプション)
- Python (オプション)

オプションのライブラリ無しでコンパイルした場合は、それに対応する機能を利用できなくなります。Qt 無しでコンパイルした場合は、GUI アプリケーションを利用できません。Python 無しでコンパイルした場合は、スクリプト機能を利用できません。

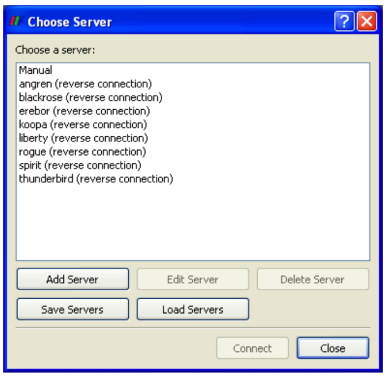
ParaView をコンパイルするには、まず CMake を実行し、コンパイルのための設定とシステム上のライブラリの指定を行います。これによって Makefile が生成され、その Makefile を使用して ParaView をビルドします。ParaView サーバのビルドに関するさらなる詳細については、ParaView Wiki をご覧下さい。

http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server#Compiling

ParaView を並列に実行するのもまた、必然的にスタンドアローンのクライアントを実行するより難しくなります。リモート・コンピュータにログインし、並列ノードを確保し、並列プログラムを起動し、接続を確立し、ファイアウオールのトンネリングを行うといった、実行環境に依存する幾つもの段階を経る必要があります。

クライアントーサーバ間接続は、`paraview` クライアント・アプリケーションによって確立されます。サーバに接続し、またサーバへの接続を解除するには、  および  ボタンを使用します。ParaView の起動時には、ビルトイン (内蔵)・サーバというサーバに自動的に接続されます。また、サーバへの接続が解除された時にも、常にビルトイン・サーバに接続されます。それら両者の例は、既に見たとおりです。

 ボタンをクリックすると、接続可能な既知のサーバのリストを含んだダイアログが現れます。このサーバのリストはサイト、またはユーザごとに設定できます。

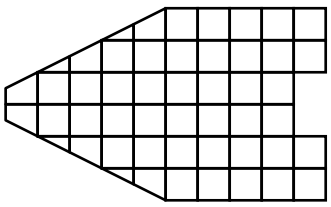


どのようにしてサーバに接続するかは、Add Server ボタンをクリックして GUI で設定するか、または XML の設定ファイルによって設定することができます。サーバ接続を指定するための方法は幾つかありますが、最終的にはサーバを起動するためのコマンドと、サーバの起動後に接続するためのホスト名を設定することになります。サーバ接続の確立に関するさらなる詳細については、ParaView Wiki をご覧下さい。

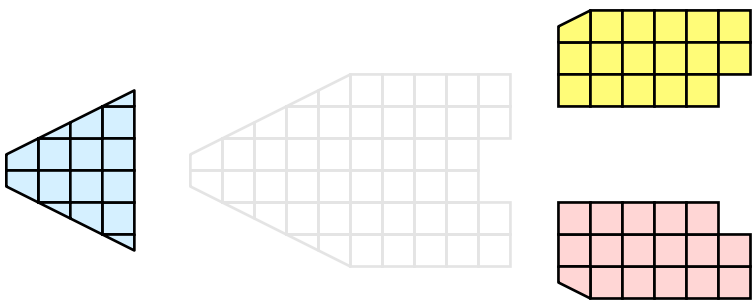
http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server#Running_the_Server

3.3 並列可視化アルゴリズム

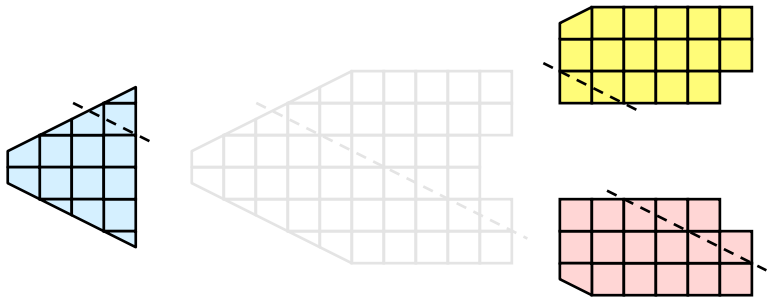
ひとたび並列化のフレームワークさえ作成すれば、並列可視化を行うのは単純である点で、私たちは幸運といえます。データはメッシュに含まれていますから、それはすなわち、既にデータがセルによって細かな断片に細分化されていることになります。それらのセルをプロセスごとに分割することで、分散並列マシン上で可視化を行うことができます。具体的に示すため、以下の非常に簡略化されたメッシュを考えます。



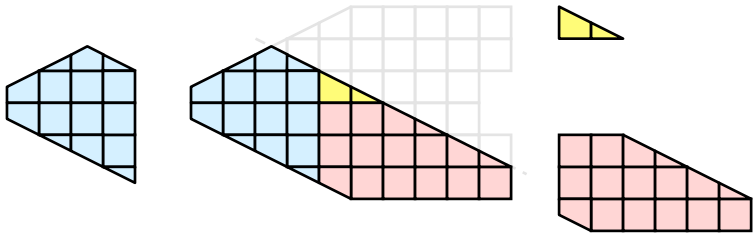
このメッシュについて、3 つのプロセスを使用して可視化を行いたいとします。以下に青、黄、ピンク色の領域で示すように、このメッシュのセルを分割します。



ひとたび分割されれば、幾つかの可視化アルゴリズムは、それぞれのプロセスにおいてローカルに保持されるセルに対してアルゴリズムを独立に実行することで、処理することができます。クリッピングを例にしましょう。クリッピングを行う切断面を定義し、この切断面をそれぞれのプロセスに与えます。

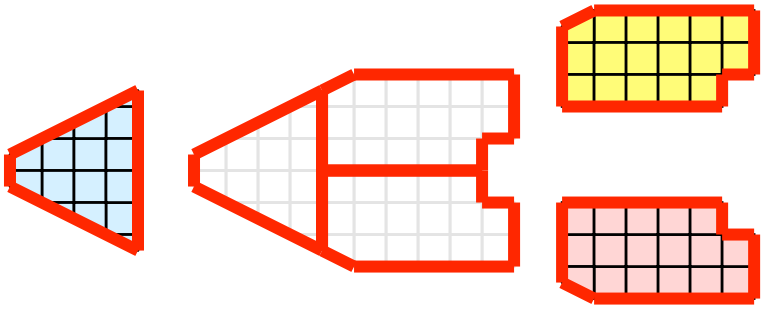


それぞれのプロセスは、この切断面によって独立にクリッピングを行うことができます。最終的に得られる結果は、このクリッピングをシリアルに実行した場合と同じです。(大規模データでは明らかに、絶対に行うべきではありませんが) これらのセルをもし、再び結合したとすれば、クリッピング演算が正しく行われたことがわかります。



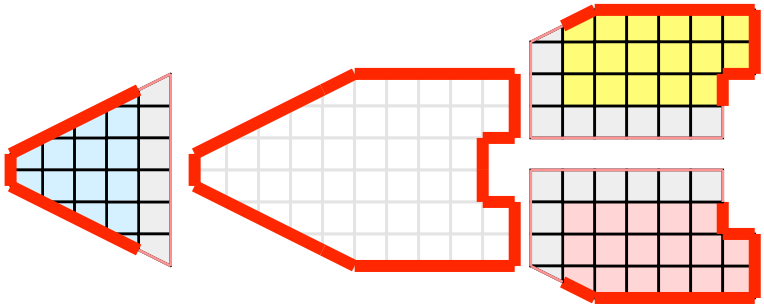
3.4 ゴースト・レベル

しかしながら残念なことに、可視化アルゴリズムを分割されたセルに対して闇雲に実行しても、必ず答えが正しいとはとは限りません。簡単な例として、**外部界面**アルゴリズムを考えて下さい。外部界面アルゴリズムとは、1 つのセルにのみ属するセル界面のすべてを探索する、すなわちメッシュの境界面を特定するアルゴリズムです。



おっと。全てのプロセスが外部界面アルゴリズムを個別に実行すると、多くの内部界面が外部界面として誤って特定されるのが判ります。これは、あるパーティションのセルが別のパーティションのセルに隣接するところで起こります。プロセスが別のパーティションのセルにアクセスする方法がありませんので、これらの隣接セルが存在することを知る方法はありません。

ParaView や、その他の並列可視化システムで採用されている解決法は、**ゴースト・セル**の利用です。ゴースト・セルとは、あるプロセスによって保持されていますが、実際には別のプロセスに属するセルのことです。ゴースト・セルを使用するには、それぞれのパーティションにおける全ての隣接セルを特定しなければなりません。そして、それらの隣接セルをそのパーティションにコピーし、コピーされたセルをゴースト・セルとして、以下の例では灰色のセルで示されるようにマークします。

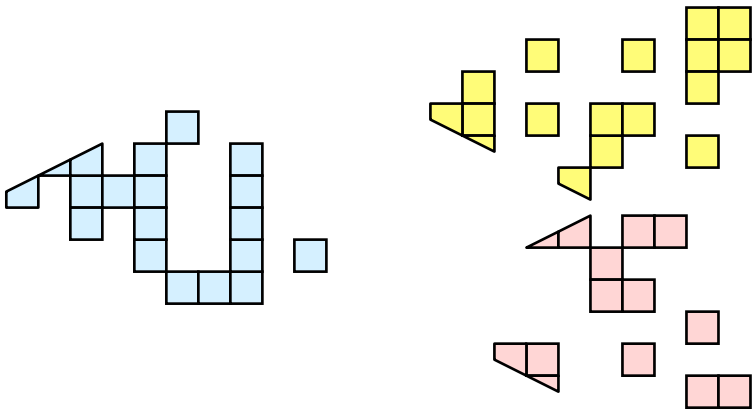


外部界面アルゴリズムをゴースト・セルに対して実行すると、それでもなお幾つかの内部界面を外部として誤って特定しているのが判ります。しかしながら、これらの全ての誤って特定された界面はゴースト・セルに属しており、したがってそれらの界面は属するセルの「ゴーストである」 との状態を継承しています。それらの「ゴーストな」界面は ParaView によって取り除かれ、正しい解に辿り着きます。

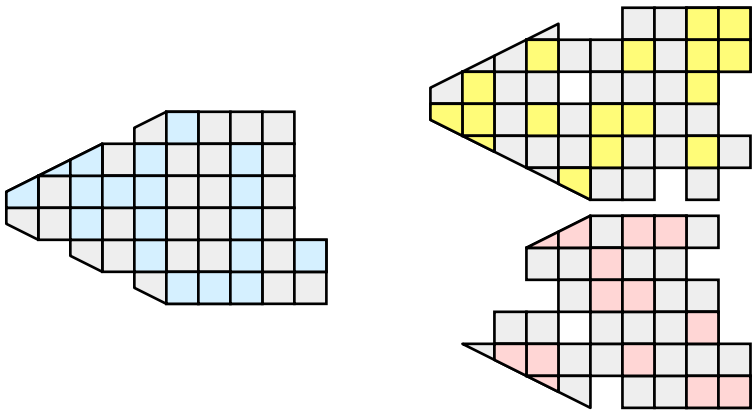
この例では 1 層のゴースト・セル、すなわちパーティション内のセルに直接隣接するセルのみを示しました。ParaView では、複数の層からなるゴースト・セルを取出すことが可能です。すなわち、それぞれの層が、下のゴースト層あるいは元のデータそのものに含まれていない一つ下の層の隣接セルを含んでいます。これは、それぞれがゴースト・セルの層を必要とするようなフィルタを直列に使用する際に便利です。それらのフィルタは、それぞれ上流側に対して追加のゴースト・セルの層を要求し、下流側へデータを送る際に 1 層だけ取り除きます。

3.5 データの分割

ここではデータを分割して分散させるため、データの分割法の影響に関して議論しておくこととします。前述の例で示されたデータは**空間的にコヒーレントな**分割法と言えます。すなわち、各パーティションのセルは全て空間中のコンパクトな領域に存在しています。その他にもデータを分割する方法は存在します。例えば、ランダムに分割する方法も有り得ます。



ランダム分割には、良い面があります。というのも、パーティションの作成および負荷分散が容易であるからです。しかしながら、ゴースト・セルに関しては、深刻な問題が存在します。



この例では、1 層のゴースト・セルによって、全プロセスにおいてデータセットのほぼ全体が複製されてしまっていることが判ります。したがって、並列処理の利点がほぼ失われてしまっています。ParaView においてゴースト・セルは頻繁に使用されるため、ParaView ではランダム分割は使用されていません。

3.6 D3 フィルタ

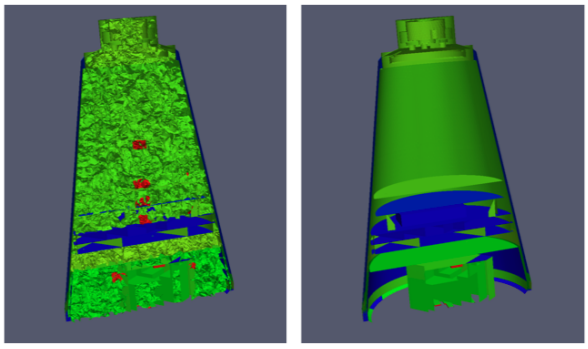
前節では、並列可視化のための負荷分散およびゴースト・セルの重要性を述べました。本節では、負荷分散のための方法を述べます。

負荷分散およびゴースト・セルは、構造データ (画像データ、直線格子、および構造格子) を読み込んだ際には、ParaView によって自動的に取扱われます。構造データが暗黙に有するトポロジのために、データを空間的にコヒーレントなパーティションに分割し、隣接セルの位置を特定するのが容易であるためです。

ところが、非構造格子 (ポリ・データおよび非構造格子) を読み込む際には、全く異なった状況になります。非構造格子には、暗黙のトポロジや隣接セルの情報があ

りません。このときの ParaView の動作は、どのようにデータがディスクに書かれたかに依存します。したがって、非構造格子が読み込まれる際には、そのデータがどの程度良好に負荷分散されるかについては保証されません。さらに、そのようなデータがゴースト・セルの情報を含んでいる可能性は低いため、幾つかのフィルタの出力は正しくない可能性があります。

幸い ParaView には、非構造格子の負荷分散を行って、ゴースト・セルを作成するためのフィルタがあります。このフィルタは、distributed data decomposition (分散データ分割) の頭文字をとって D3 と呼ばれます。D3 フィルタの使用法は簡単で、(Filters → Alphabetical → D3 に存在する) このフィルタを、分割し直したいデータに適用するだけです。



D3 の最も一般的な使い方は、非構造格子データ読み込みモジュールに対して直接適用する方法です。読み込まれたデータが如何にうまく負荷分散されていても、以降のフィルタが正しいデータを生成するよう、ゴースト・セルを抽出することは重要です。上記の例は、ある非構造格子へ表面抽出フィルタを適用した結果の断面です。左図ではゴースト・セルが無いために、多くの界面が誤って抽出されていることが判ります。一方で右図では、D3 フィルタをまず最初に適用することで、その不具合が修正されています。

3.7 ジョブ・サイズにデータ・サイズを合わせる

ParaView **サーバとして、幾つのプロセスを使用すべきでしょうか。** これは多くの重要な影響を含む、普遍的な質問と言えます。そしてまた、非常に難しい質問でもあります。それぞれのプロセスにどのようなハードウエアが対応付けられているか、どの程度の大きさのデータを処理しようとしているか、どのような型のデータを処理しようとしているか、どのような種類の可視化操作を行おうとしているか、そしてユーザ自身の我慢強さなどの要因によって、その回答は大きく左右されます。

したがって、確固たる回答はありません。しかしながら、幾つかの経験則はあります。

構造データ (画像データ、直線格子、構造格子) に対しては、少なくとも 2,000 万セルにつき 1 つのプロセッサが与えられるようにしてください。もしさらにプロセッ

サを割り当てられるなら、500 から 1,000 万セルにつき 1 プロセッサが与えられれば、通常は充分です。

非構造データ (ポリ・データ、非構造格子) に対しては、少なくとも 100 万セルに対し 1 プロセッサが与えられるようにしてください。もしさらにプロセッサを割り当てられるなら、25 万から 50 万セルにつき 1 プロセッサが与えられれば、通常は充分です。

前述のように、これらは経験則に過ぎず、絶対的法則ではありません。常にデータ量に対してどの程度のプロセッサが適切かを評価し、実験するように努めるべきです。そしてもちろん、読み込みたいデータの規模が、ユーザが利用可能な計算機資源の限界を拡大する時が来る可能性は、常にあります。このような状況になれば、データ量の爆発的増大を確実に回避し、データを確実に間引きたくなることでしょう。

3.8 データ量の爆発的増大の回避

ParaView の有するパイプライン・モデルは、試行錯誤による可視化にはとても便利です。コンポーネント間の連携が緩やかであるため、型どおりでない可視化の構築のためのとても柔軟なフレームワークとなっています。また、パイプライン構造のため、設定を素早く容易に追い込むことができるようになっています。

この連携方法の欠点は、メモリ使用量が比較的多くなることです。パイプラインの各段階でそれぞれがデータを重複して保持するためです。ParaView は可能な時は常に、パイプラインの各段階がメモリ内の同一の領域のデータを参照するよう、データの**浅いコピー**を行います。しかしながら、新たなデータを作成したり、データの値やトポロジを変更するようなフィルタは全て、それらの結果のための新たなメモリを確保せざるを得ません。ParaView が非常に大きなメッシュに対してフィルタを適用する場合には、フィルタの使用法が適切でなければ即座に利用可能なメモリを全て使い尽くしてしまいます。したがって、大規模なデータセットを可視化する際には、フィルタのメモリ所要量を理解しておくことが重要です。

ただし、以下の助言は**非常に大規模なデータを取扱っていながら、使用可能なメモリが少なくなっている時のためのみ**であることに注意してください。メモリが尽きる心配がなければ、以下の助言は全て無視してください。

構造データを扱っているときは、どのフィルタがデータを非構造データに変更するかを知っておくことは絶対的に重要です。なぜなら、非構造データは、トポロジを明示的に記述する必要があるため、構造データよりもセルあたりのメモリ使用量が大幅に増大するからです。ParaView には、トポロジを何らかの方法で変更するフィルタが数多く存在し、それらのフィルタは生成されるあらゆる種類のトポロジを扱えるデータセットが非構造データのみであるため、データを非構造格子として書き出します。以下にリストされたフィルタは、入力とおおむね等しい新たな非構造格子トポロジを出力に書き出します。これらのフィルタは、**絶対に**構造データに対しては使用するべきではなく、非構造データに対してのみ注意しながら使用するべき

です。

- Append Datasets
- Append Geometry
- Clean
- Clean to Grid
- Connectivity
- D3
- Delaunay 2D/3D
- Extract Edges
- Linear Extrusion
- Loop Subdivision
- Reflect
- Rotational Extrusion
- Shrink
- Smooth
- Subdivide
- Tessellate
- Tetrahedralize
- Triangle Strips
- Triangulate

技術的には、Ribbon および Tube フィルタもこのリストに含まれます。しかしながら、それらはポリ・データ中の 1 次元のセルにのみ作用するため、入力データは通常小規模で、ほとんど影響はありません。

次の同様な一連のフィルタもまた、非構造格子を出力しますが、一般的にはそのデータ量 (格子数) をいくらか削減します。ただし、このデータ量削減は多くの場合、非構造格子への変換によるオーバーヘッドを補う程ではないことに注意して下さい。また多くの場合、このデータ量の削減は (負荷分散の点では) あまり上手くバランスしないことに注意して下さい。したがって、これらのフィルタは非構造データに対しては注意深く、また構造データに対しては非常に注意深く使用する必要があります。

- Clip 
- Decimate
- Extract Cells by Region
- Extract Selection 
- Quadric Clustering
- Threshold 

上のリストの項目と同様に、Extract Subset  は構造データセットに対してデータ量削減を行いますが、構造データを出力します。したがって、新たなデータが作成されるという注意点は同様ですが、非構造データに変換されるとの心配は不要です。

以下の一連のフィルタもまた非構造データを出力しますが、(3 次元から 2 次元へのような) データの次元の削減を行いますので、出力はずっと小さくなります。したがって、これらのフィルタは通常、非構造データに対しても使用することができ、構造データに対しても多少の注意を払えば充分です。

- Cell Centers
- Contour 
- Extract CTH Fragments
- Extract CTH Parts
- Extract Surface
- Feature Edges
- Mask Points
- Outline (curvilinear)
- Slice 
- Stream Tracer 

これらのフィルタはデータのコネクティビティを全く変更しません。そのかわり、データにフィールド配列のみを付加します。既に存在するデータに対しては浅いコピーが行われます。これらのフィルタは通常、どのようなデータに対しても使用することができます。

- Block Scalars
- Calculator 
- Cell Data to Point Data
- Curvature
- Elevation
- Generate Surface Normals
- Gradient
- Level Scalars
- Median
- Mesh Quality
- Octree Depth Limit
- Octree Depth Scalars
- Point Data to Cell Data
- Process Id Scalars
- Random Vectors
- Resample with dataset
- Surface Flow
- Surface Vectors
- Texture Map to...
- Transform
- Warp (scalar)
- Warp (vector) 

以下の最後の一連のフィルタは、データを出力に全く追加しない (全ての出力データは浅いコピーによって得られる) か、付加されるデータは入力データの量に依存しないフィルタです。これらはどんな状況でも、まず問題なく使用することができます (ただし、処理時間は要するかもしれません)。

- Annotate Time
- Append Attributes
- Extract Block
- Extract Datasets
- Extract Level 
- Glyph 

- Group Datasets 
- Histogram 
- Integrate Variables
- Normal Glyphs
- Outline
- Outline Corners
- Plot Global Variables Over Time
- Plot Over Line 
- Plot Selection Over Time 
- Probe Location 
- Temporal Shift Scale
- Temporal Snap-to-Time-Steps
- Temporal Statistics

上記の分類には上手くあてはまらない、特殊なフィルタも存在します。それらのフィルタの幾つか (今のところ Temporal Interpolator と Particle Tracer) は、データが時間によってどのように変化するかに基づいて処理を行います。したがって、これらのフィルタは 2 ステップ、またはそれ以上の時刻ステップを読み込む必要があり、メモリ上で必要なデータの量が倍、またはそれ以上となる可能性があります。また、時間軸方向の処理を行うフィルタの幾つか、例えば Temporal Statistics や、時間軸に沿ってデータをプロットするようなフィルタは、全てのデータをディスクから反復的に読み込む必要があります。したがって、たとえ余分なメモリを使用しなくても、非実用的なほど長い処理時間を要する可能性があります。

Programmable Filter  もまた、分類が不可能な特殊なケースです。このフィルタはプログラミングされたとおりの処理を行いますので、これらの分類のいずれにもあてはまる可能性があります。

3.9 データを間引く

大規模なデータを扱う際には、可能な限りデータを間引くことが明らかに最良の策であり、それも早い段階で行うほど、良いといえます。ほとんどの大規模データは 3 次元の形状として読込まれますが、所要の形状はそのデータの (表面などの) 面であることがしばしばあります。面データの所要メモリは通常、その元となる立体データよりも大幅に小さいため、早い段階で面データに変換するのが最良といえます。ひとたびそのようにすれば、他のフィルタを比較的安全に適用することができます。

非常に一般的な可視化の操作としては、Contour  フィルタを使用して立体データから等値面を抽出することが挙げられます。Contour フィルタは通常、入力よりずっと小さなデータ量の形状を出力します。したがって、いかなる状況であれ、もし Contour フィルタを使用するのであれば、早い段階で適用するべきです。とはいえ、Contour フィルタは大量のデータを生成する可能性もありますので、設定には注意して下さい。大量の等値面を生成する値を指定すれば、当然ながらそのような

状況は起こり得ます。データ中の等値面を生成する値の上下に、ノイズのような高周波の変動が存在すれば、それも大量の不整形な面を生成する原因となります。

立体データの内部を見るもう一つの方法は、Slice フィルタを適用することです。Slice フィルタは、立体データを面によって薄切りにして、立体内の面が切断する部分のデータを見えるようにします。もし大規模データの中で興味深い特徴を有する位置が既に判っていれば、薄切りにするのがそのデータを見る良い方法です。

もしもデータについて**あらかじめ与えられた**情報がほとんどなく、しかも全データセットに対するメモリ量や処理時間を必要とすることなくデータを調べたいのであれば、Extract Subset フィルタを使用してデータの一部の領域を抽出、および間引くことができます。間引かれたデータの量を元のデータよりも大幅に小さくすることは可能でありながら、間引かれたデータは良好に負荷分散されているはずです。もちろん、間引くことによって細かなデータの変化は失われる可能性があり、必要なデータの特徴を発見したら完全なデータセットに戻って可視化を行うべきであることは注意して下さい。

立体データの一部を取り出すことのできるフィルタは幾つかあります。Clip 、Threshold 、Extract Selection、そして Extract Subset はいずれも、何らかの基準によってセルを抽出することができます。しかしながら、抽出されたセルが上手く負荷分散されている可能性はほとんど無く、全くセルが取り除かれないプロセスも存在する可能性に注意して下さい。また、Extract Subset 以外のこれらのフィルタは全て、構造データ型を非構造格子に変換します。したがってこれらは、抽出されたセルが元のデータより少なくとも一桁以上少なくなるのでない限り、使用するべきではありません。

可能であれば、3 次元データを抽出するフィルタを 2 次元の面を抽出するフィルタで置き換えてみて下さい。例えば、データ中のある平面を調べたいのであれば、Clip フィルタよりも Slice フィルタを使用して下さい。もしある範囲内の値を含むセルの領域の位置を調べるのであれば、Threshold フィルタを使って全てのセルを抽出するよりも、Contour フィルタを使ってその範囲の両端となる等値面を生成するようにしてみてください。ただし、このように代わりのフィルタを使用すると、下流側のフィルタに影響する可能性があることに注意して下さい。例えば、Threshold フィルタの後に Histogram フィルタを実行するのは、ほぼ同等の Contour フィルタの後に実行するのとは全く異なった結果となります。

3.10 レンダリング

レンダリングとは、データから実際に目にする画像を生成する処理のことです。データと効率的にインタラクションする能力は、レンダリングのスピードに大きく依存します。コンピュータ・ゲーム市場に牽引された 3 次元処理のハードウエア・アクセラレーションの進歩によって、高価でないコンピュータでさえ、3 次元画像を高速にレンダリングできるようになりました。しかしながらもちろん、レンダリン

グの速度はレンダリングされるデータの量に比例します。データが大きくなるほど、レンダリング処理は必然的に遅くなります。

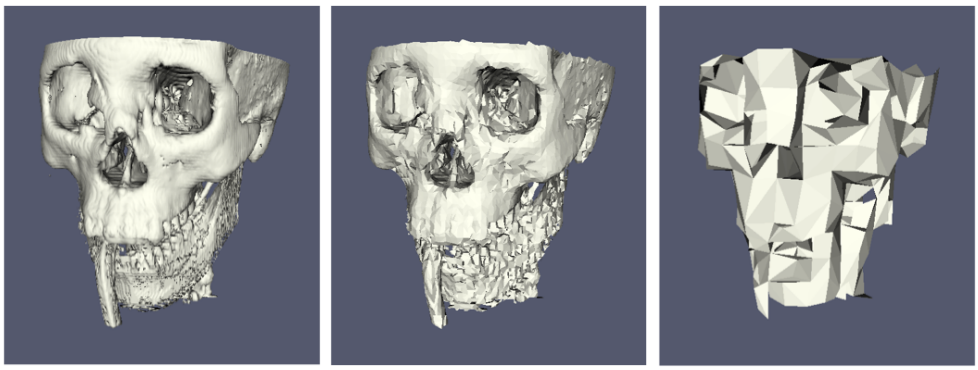
可視化セッションが確実にインタラクティブに実行されるため、ParaView は 2 つのモードをサポートしており、それらのモードは必要に応じて自動的に切り替えられます。**スティル・レンダー**と呼ばれる 1 つ目のモードでは、データは可能な限りのディテールを保持してレンダリングされます。このレンダリング・モードでは、確実にデータの全てが正確に表現されます。**インタラクティブ・レンダー**と呼ばれる 2 つ目のモードでは、正確さよりも速度が優先されます。このレンダリング・モードでは、データの大きさにかかわらず、高速なレンダリングが行われるよう配慮されます。

マウスによる回転、パン、ズームなどの、3D ビューにおけるインタラクションを行っている間は、ParaView はインタラクティブ・レンダーを行います。これはインタラクションの間、これらの機能を実用的に保つため、高いフレームレートが必要であるのと、インタラクションの間それぞれのフレームはすぐに次のフレームで置き換えられるため、このモードにおいては精細なディテールはあまり重要でないためです。3D ビューにおけるインタラクションが行われていない時は、ParaView はデータの全てのディテールが明らかになるよう、スティル・レンダーを使用します。3D ビューでマウスをドラッグしてデータを動かすと、マウスを動かしている間は概略のみがレンダリングされますが、マウスボタンを放すとすぐに完全なディテールまで表現されるのがお判り頂けるでしょう。

インタラクティブ・レンダーは速度と正確さの妥協の産物です。したがって、いつ、どのように粗いディテールが使用されるかについては、多くの設定が関係します。

3.10.1 基本的な設定

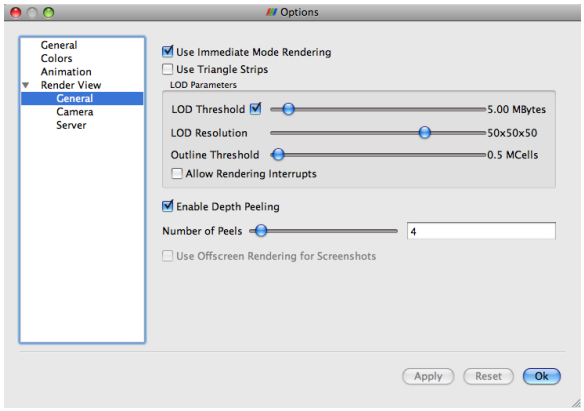
最も重要なレンダリング設定の幾つかは、LOD に関する設定です。インタラクティブなレンダリングの間、形状は低い**ディテールのレベル**(**LOD**)、すなわちより少ないポリゴンで表現された近似的な形状に置き換えられることがあります。



形状の近似化の解像度は、変更することができます。形状簡略化のアルゴリズムは、ポリゴンを粗い格子に沿って配置するように動作します。上の画像では、左の

画像は完全な解像度です。中央の画像は 50^3 分割の格子による形状簡略化の結果で、右の画像は 10^3 分割の格子による形状簡略化の結果です。

3 次元レンダリングの設定は、メニューの Edit → Settings (Mac では ParaView → Preferences) で現れる設定ダイアログボックスにあります。ダイアログを開くと、基本的なレンダリング設定は Render View → General 以下にあります。



レンダリング性能に関する設定は、以下のような意味があります。

Use Immediate Mode Rendering (イミディエイト・モード・レンダリングを使用する) これをチェックすると、形状はグラフィックス・カードに送られてイミディエイト・レンダリングが行われます。チェックを外すと、形状はより効率的なレンダリングのためにディスプレイ・リストにまとめられます。通常はディスプレイ・リストの方が高速にレンダリングされますが、最初のフレームのレンダリングの間にディスプレイ・リストを作成する時間、およびそれを保持するメモリが必要となります。

Use Triangle Strips (トライアングル・ストリップを使用する) チェックを外すと、データはポリ・データによって定義されたとおりにレンダリングされます。チェックすると、データはトライアングル・ストリップに変換されます。トライアングル・ストリップはグラフィックス・カードに効率的に転送され、より高速にレンダリングされることもありますが、通常はそうではありません。

LOD Threshold (LOD しきい値) どのような時に形状を簡略化された形状で置き換えるかを制御します。チェックボックスは、この機能自体をオンまたはオフにします。オンの場合は、スライダによってこの機能の動作するしきい値を与えます。形状のデータ量がこのしきい値未満であれば、その形状はレンダリングするのに充分小さいと見なされます。形状のデータ量がこのしきい値より大きければ、簡略化された形状がレンダリングに使用されます。

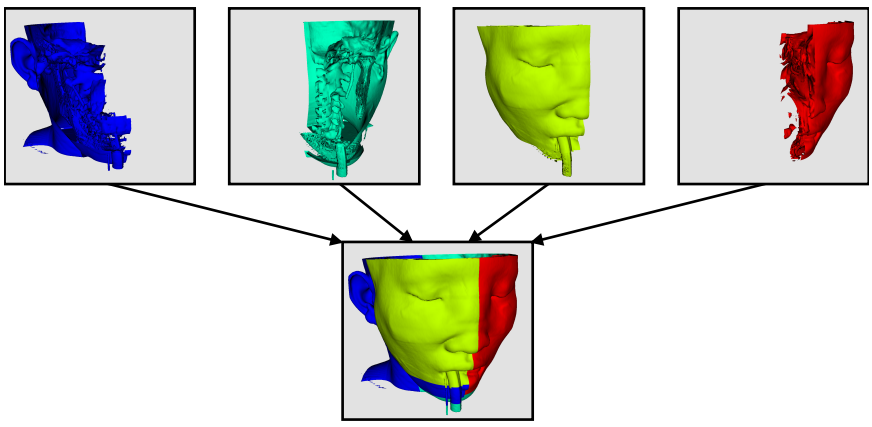
Allow Rendering Interrupts (レンダリングの中断を許可する) チェックされている場合、スティル・レンダーを中断してインタラクティブ・レンダーを行えるようになります。

Enable Depth Peeling (デプス・ピーリングを使用する) ParaView では、半透明のサーフェイスを正しくレンダリングするために、デプス・ピーリングと呼ばれるアルゴリズムを使用します。このアルゴリズムでは、まず最前面のサーフェイスがレンダリングされ、そして次にその下のサーフェスがレンダリングされるよう「剥がされる」プロセスが繰り返されます。もしサーフェイスを半透明にすることで動作速度が大幅に落ちたり、レンダリング結果が正しくないようなら、グラフィックス用ハードウエアにおけるデプス・ピーリング拡張の実装が良くない可能性があります。また、デプス・ピールの数を調整することもできます。ピールの数を増やすことで、複雑な深さを正しくレンダリングできるようになりますが、逆に減らすことで、レンダリングの速度を上げることができます。

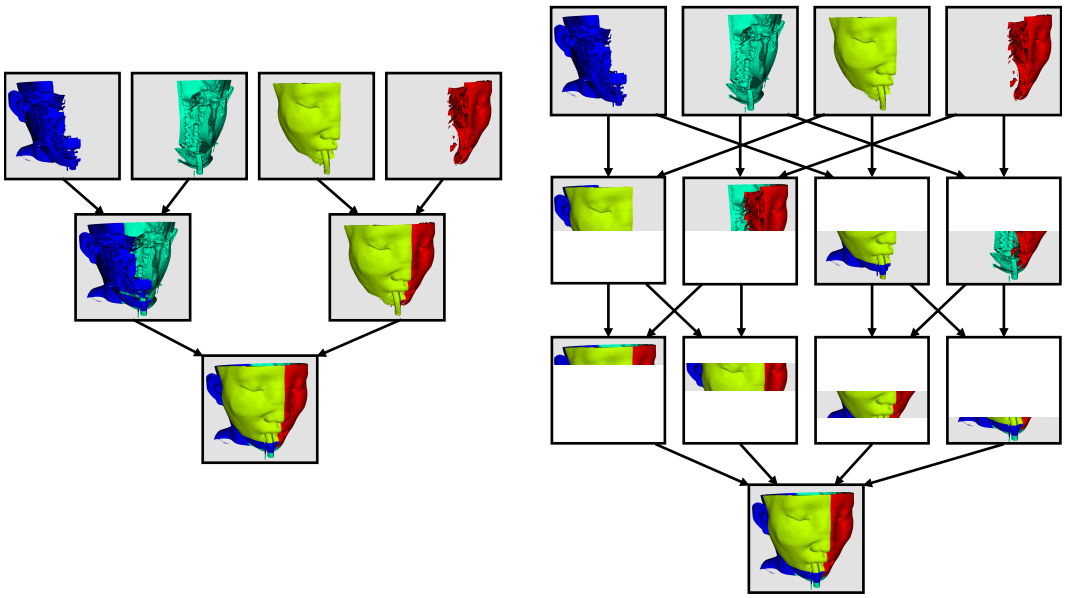
Use Offscreen Rendering for Screenshots (スクリーンショットにはオフスクリーン・レンダリングを使用する) ParaView では、デスクトップ上の ParaView 以外のウインドウが画像に影響を与えるのを防ぐため、スクリーンショットの作成には通常、オフスクリーン・レンダリングが使用されます。しかしながら、グラフィックス・ハードウエアの中には、オフスクリーン・レンダリング画像にノイズが混入するような制約を有するものがあります。もし保存された画像がスクリーン上の画像と異なって見える (例えば、暗くなっているなど) ようなら、この機能をオフにしてみてください。

3.10.2 基本的な並列レンダリング

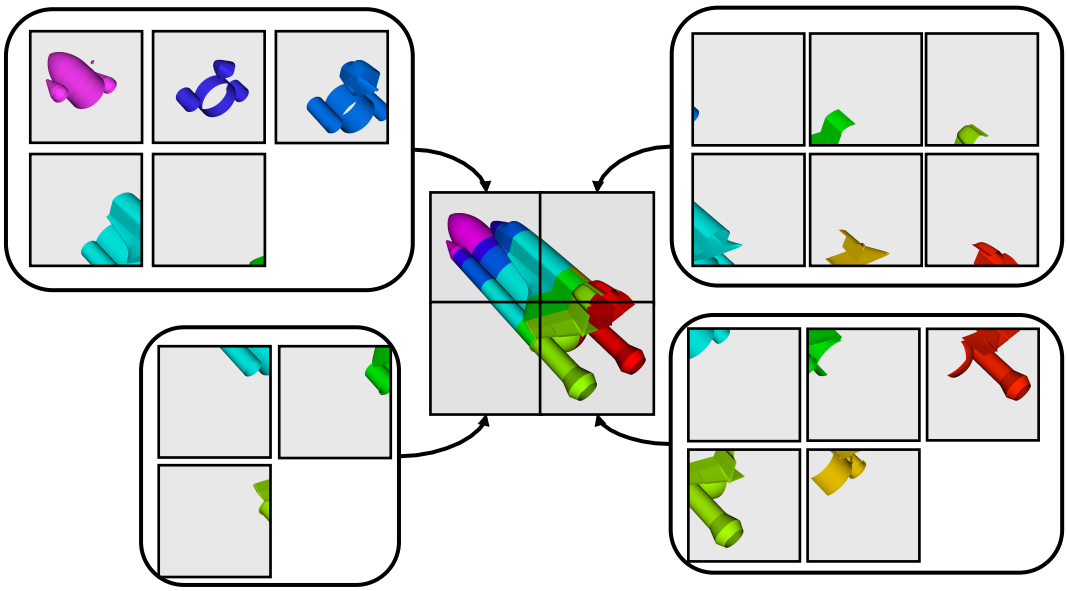
並列可視化を行う際には、レンダリングに至るまで、およびレンダリング自体の全てのプロセスにおいて、データが分割されているよう注意します。ParaView では、**IceT** と呼ばれる並列レンダリング・ライブラリが使用されます。IceT では、**ソート・ラスト**・アルゴリズムが、並列レンダリングに使用されます。この並列レンダリング・アルゴリズムによって、それぞれのプロセスはそれぞれに与えられた分割された形状を独立にレンダリングし、その結果の部分的な画像を**重畳**して最終的な画像を生成することができます。



上の図は大幅に簡略化したものです。IceT には、**2 分木法**や**バイナリ・スワップ法**のような、いくつかの段階を用いて処理を効率的にプロセス間に分割する、複数の並列化された画像重畳アルゴリズムが含まれています。



ソート・ラスト法による並列レンダリングの素晴らしい点は、その効率がレンダリングされるデータの量に全く依存しないことです。そのため、この手法は非常にスケーラブルで、大規模なデータに適しています。しかしながら、並列レンダリングのオーバーヘッドは画像中の画素数に対して線形に増加します。したがって、レンダリング設定の幾つかは画像の大きさに関するものです。



IceT は、タイリング表示されたディスプレイ (多数の並べられたモニタやプロジェクタによって構成される、大型・高解像度のディスプレイ) を駆動する能力があり

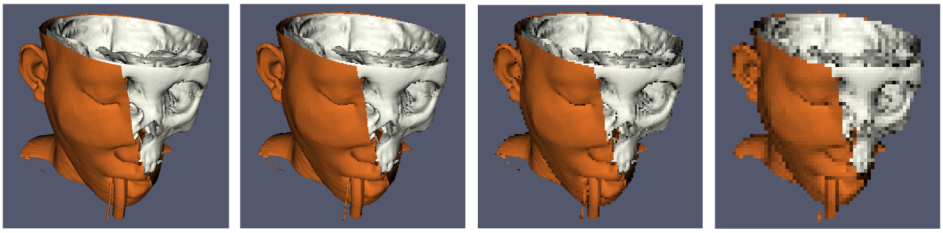
ます。ソート・ラスト・アルゴリズムをタイリング・ディスプレイに使用するのは、重畳すべき画素数が非常に多くなるため、やや直感的ではありません。しかしながら IceT では、それぞれのプロセスに存在するデータの特定の局所性を利用して、必要な重畳処理を大幅に低減するようになっています。この空間的な局所性は、データに対し D3 フィルタを適用することで強制することができます。

並列レンダリングにはオーバーヘッドが存在するため、ParaView ではいつでも並列レンダリングをオフにすることができます。並列レンダリングがオフの時は、形状データが実際に表示が行われる所に転送されます。明らかに、これはレンダリングするデータが小さいときにのみ行うべきです。

3.10.3 画像ディテールのレベル

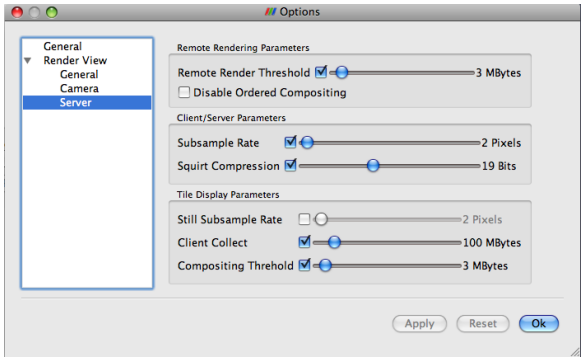
並列レンダリングのアルゴリズムによって発生するオーバーヘッドは、生成される画像の大きさに比例します。また、サーバ上で生成された画像はクライアントに転送する必要があり、そのコストも画像の大きさに比例します。ユーザによるインタラクションの間のフレームレート向上を補助するため、ParaView には画像の大きさを制御する新たな LOD パラメータが導入されています。

ParaView では並列レンダリング中のインタラクションの間、画像を間引くことができます。すなわち、ParaView がインタラクションの間、画像の縦横各方向解像度をある割合で減らします。間引かれた画像がレンダリングされ、重畳され、そしてクライアントへ転送されます。クライアントでは、転送された画像が GUI の画面サイズに合わせて拡大されます。



間引かれた画像の解像度は、各方向の解像度を割る除数によって制御されます。上の画像では、左の画像が完全な解像度となっています。それに続いて、それぞれ 2、4、8 分の 1 に間引かれた解像度でレンダリングされた画像となっています。

3.10.4 並列レンダリングの設定



他の 3 次元レンダリングの設定と同じように、並列レンダリングの設定も Edit → Settings (Mac では ParaView → Preferences) によって現れる設定ダイアログボックスに配置されています。ダイアログを開くと、並列レンダリングの設定は Render View → Server 以下にあります。それぞれの設定は、以下のような意味があります。

Remote Render Threshold (リモート・レンダリングを行うしきい値) このチェックボックスは、リモート・レンダリングをオンまたはオフにします。スライダによって、並列レンダリングを行うしきい値を変更することができます。形状のデータ量がこのしきい値未満であれば、形状データが表示が行われる所 (通常はクライアント) に転送されます。

Suppress Ordered Compositing (順序付け重畳を行わない) ボリューム・レンダリングおよび透明なポリゴンが正しく表示されるには、順序付け重畳と呼ばれる特別な並列レンダリング・モードが必要です。しかしながら、このモードには追加の計算処理とメモリが必要です。このチェックボックスをオンにすると、順序付け重畳を行わなくなります。

Interactive Subsample Rate (インタラクション時の間引き率) 並列レンダリングのオーバーヘッドは、生成された画像の大きさに比例します。したがって、インタラクティブなレンダリングは、画像を間引く割合を指定することで高速化することができます。このチェックボックスがチェックされている時は、インタラクティブなレンダリングでは低解像度の画像を生成し、表示される時にその画像を拡大するようになります。この設定はインタラクティブなレンダリングの時のみ使用されます。スティル・レンダーの時は、常に完全な解像度の画像が使用されます。

Image Compression (画像圧縮) 画像がサーバからクライアントに転送される前に、その画像を 2 つのアルゴリズムのいずれかによって圧縮することができます。すなわち **SQUIRT** または **Zlib** です。圧縮をさらに効果的にするために、いずれのアルゴリズムでも圧縮前に画像の色深度を削減することができます。スライダによって、削減後に保持される色深度のビット数を指定します。スティル・レンダーの時は、常にフルカラーの色深度が使用されます。

Still Subsample Rate (スティル・レンダーの間引き率) タイリング・ディスプレイは色々な用途に使われます。例えば、小規模な共同作業や大規模なプレゼンテーションなどです。大規模なプレゼンテーションでは、観衆がディスプレイの全ての画素を見ることはまずありません。そのような場合には、このオプションによってスティル・レンダーの画像を間引いて、レンダリングの時間を大幅に短縮することができます。

Client Collect (クライアントに集約する) タイリング・ディスプレイ・モードでは、並列レンダリングによる画像はデスクトップでなく、タイリングされたディスプレイに送られます。したがって、クライアントは全てのデータをローカルにレンダリングしなければなりません。この設定では、クライアントに送られるデータ量の上限を指定します。データが指定されたしきい値より大きければ、クライアントにはデータが収まるバウンディング・ボックスのみが表示されます。

Compositing Threshold (重畳を行うしきい値) 重畳を行うしきい値とは、タイリング・ディスプレイに対するリモート・レンダリングを行うしきい値と同等のものです。ただし、重畳を行うことによる得失は異なります。タイリングされたディスプレイは解像度が高いために重畳のオーバーヘッドは大きく、そのかわり通常は高速なネットワーク内で行われるため、形状の集約のオーバーヘッドは小さくなります。したがって、重畳を行うしきい値をリモート・レンダリングを行うしきい値よりも高くすると、効果のあることがあります。形状のデータ量がこの重畳を行うしきい値より小さければ、表示される形状が全てのレンダリング・ノードに送られ、それぞれのレンダリング・ノードはそれぞれが受け持つタイリング・ディスプレイの部分に直接レンダリングされます。

3.10.5 大規模データのための設定

デフォルトのレンダリング設定は、ほとんどのユーザに適しています。しかしながら、非常に大規模なデータを扱う時は、設定を追い込むことが有効であることがあります。最適な設定はデータおよびParaViewが実行されるハードウエアによって変わりますが、以下のような助言に従うと良いでしょう。

1. もしレンダリング・クラスタにレンダリングのためのハードウエアが特に装備されていなければ、Use Immediate Mode Renderingをオンにしてください。この設定をオンにすると、グラフィック・システムがレンダリングのための特殊なデータ構造を作成しなくなります。グラフィックスのためのハードウエアが装備されている場合は、これらのレンダリングのためのデータ構造は、GPUにデータを十分高速に供給するために重要です。しかしながら、GPUが装備されていなければ、これらのレンダリングのためのデータ構造は大して役に立ちません。

2. Use Triangle Strips をオフにしてください。この設定はデータを効率的にレンダリングできる構造に変換するためのものです。しかしながら、データが大きいときには、変換処理の負荷とメモリ使用量のオーバーヘッドが、変換によって得られる効率化に見合いません。実際、メモリ容量が限界いっぱいのときには、この設定によって性能が悪化することがあります。

3. LOD Threshold を**オフ**にしてみてください。大規模データにおいては、形状の簡略化には長い時間がかかり、データに極端な曲がり部や特殊なコネクティビティがあれば、良い結果が得られないことがあります。LOD によって性能が改善されるのであれば、LOD Resolution の設定スライダを一番右 $(10 \times 10 \times 10)$ に動かしてみて下さい。

4. 常にリモート・レンダリングをオンにしておいてください (Remote Render Threshold の隣のチェックボックスで切り替えられます)。リモート・レンダリングによってサーバ全体のレンダリング能力が使われ、クライアントに画像が転送されます。リモート・レンダリングがオフであれば、形状データがクライアントに転送されます。大規模なデータに対しては、形状データを転送するよりも画像を転送する方が必ず高速です。

5. 間引き処理をオンにして、Subsample Rate を必要に応じて調整して下さい。画像の重畳に時間がかかる場合は、クライアントとサーバの間の接続が狭帯域であるか、非常に大きな画像をレンダリングしているときには間引き率を大きくすると、インタラクティブ・レンダリングの性能が大幅に改善されることがあります。

6. Image Compression がオンであることを確認して下さい。この設定はデスクトップへの画像表示性能に大きな影響があります。また、この設定によって引き起こされる画質低下は最小限であり、かつ影響を受けるのはインタラクティブ・レンダリング時のみです。

第4章　Pythonによるバッチ・スクリプティング

ParaViewにおけるPythonによるスクリプティングは、主に2つの用途に活用することができます。1つはGUIでユーザが行うのと同じことを行って、設定や可視化の実行を自動化することです。もう1つは、Pythonスクリプトをパイプライン・オブジェクトの中で走らせ、それによって並列可視化アルゴリズムを実行することです。本章では、最初のモード、すなわち可視化の自動化のためのバッチ的なスクリプティングを述べます。

バッチ・スクリプティングは日常的な、あるいは繰返し行う処理の自動化に向いていますが、同時に、GUIの使用が望ましくない、あるいは使用できない状況でParaViewを使用するには必須の手法です。Pythonスクリプトを使用した自動化によって、ParaViewをスケーラブルな並列ポストプロセッシングのためのフレームワークとして活用することができます。また筆者らは、シミュレーション・コード内で、シミュレーションと同時に (*in-situ*) 可視化を行うためにも、Pythonによるスクリプティングを使用しています。

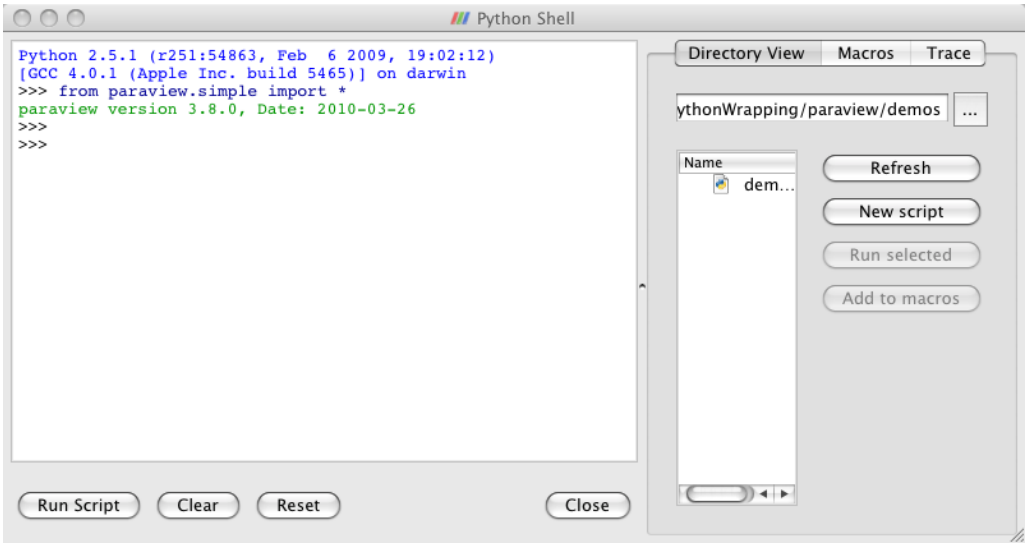
このチュートリアルでは、Pythonによるスクリプティングの、ほんの導入的な解説のみを行います。最新かつ、より詳しい解説は、ParaView Wikiに掲載されています。

http://www.paraview.org/Wiki/ParaView/Python_Scripting

4.1　Pythonインタプリタの起動

Pythonインタプリタを起動するには、幾つかの方法があります。どの方法を使用すべきかは、スクリプティングをどのように使用するかによります。Pythonインタプリタを使用するための、最も簡単で、かつこのチュートリアルで利用する方法は、メニューから Tools → Python Shell を選択することです。そうすると、ParaViewのPythonシェルのためのコントロールを有するダイアログが開きます。ダイアログの左側はPythonインタプリタで、ここにコマンドを以下に述べるインタフェイスに従って入力することで、ParaViewを直接制御することが出来ます。右側はディスクにPythonスクリプトを保存したり、ディスクから読み出したり、ParViewのメニュー・バーから直接アクセスできるスクリプトである**マクロ**を管理したり、ParaViewの

通常の GUI インタフェイスによって行われた操作を再現するためのスクリプトである**トレース**を記録するためのコントロールがあります。



もし、とにかく今すぐスクリプトを書いてみたいのであれば、以下のスクリプティングを起動するための他の方法を飛ばして、次節まで進んで頂いても構いません。

ParaView には、Python スクリプトを実行するためのコマンドライン・プログラムが、2 種類付属しています。すなわち、`pvpython` と `pvbatch` です。それらは、コマンドラインまたはファイルから Python スクリプトを読込んで、それを Python インタプリタに渡す点で、Python の配布物に付属の `python` 実行ファイルと似ています。

`pvpython` と `pvbatch` の違いは、可視化を行う際の、微妙な手法の違いによるものです。`pvpython` は、`paraview` のクライアント GUI における GUI を、Python インタプリタに置き換えたものと概ね同等と言えます。`pvpython` はシリアル実行され、1 つの ParaView サーバ (ビルトイン、またはリモートのいずれも可) に接続されるアプリケーションです。それに対して `pvbatch` は、クライアントとのソケット通信の代わりに Python スクリプトによって動作を指示される点を除いて、`pvserver` と概ね同等です。`pvbatch` は (MPI サポート付きでコンパイルされていれば) `mpirun` から起動することが可能な並列アプリケーションです。従って別のサーバに接続することはできず、それ自身がサーバとなっています。一般的には、インタプリタを対話式に使うなら `pvpython` を、並列に実行したい場合は `pvbatch` を使用します。

ParaView の Python モジュールを、ParaView の外部のプログラムから使用することも可能です。そのためには、`PYTHONPATH` 環境変数に ParaView のライブラリおよび Python モジュールへのパスを含め、`LD_LIBRARY_PATH` (Unix/Linux/Mac)[1] または `PATH` (Windows) 環境変数を、ParaView のライブラリへのパスを含むように設定します。このようにして Python スクリプトを実行することで、IDLE のようなサード・パーティのアプリケーションを使用することが可能となります。環境設定に関するさらなる情報については、ParaView Wiki をご覧下さい。

[1] 訳注: Mac OS X では `DYLD_LIBRARY_PATH`

4.2 ParaView の状態をトレースする

Python スクリプティングの機能について深く解説する前に、Python スクリプトを作成するための自動化機能についてちょっと調べてみましょう。ParaView の GUI 上の Python Trace 機能によって、多くの一般的な操作について、とても簡単に Python スクリプトを作成することができます。Trace を使用するには、GUI の Python Shell ダイアログからトレースの記録を開始し、ParaView の GUI を用いて可視化を作成し、そしてトレースの記録を停止します。これによって、GUI で行った操作を再現する Python スクリプトが作成されます。このスクリプトには、今から述べるのと全く同じ記法によるプログラムが含まれています。したがって、Trace の記録は Python インタフェイスによって同じ操作を行う方法を見出すための良い見本であり、逆に以下の解説はあらゆる Trace スクリプトの中身を理解するのに役立ちます。

演習 4.1: Python スクリプトによるトレースを作成する

もし前節から演習を続けているのであれば、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

1. もしまだ行っていなければ、メニューから Tools → Python Shell を選択し、ParaView GUI の Python シェルを開きます。
2. Python シェルの Trace タブをクリックします。
3. Start Trace をクリックします。
4. Python シェルを最小化し、ParaView GUI の主ウインドウで簡単なパイプラインを作成します。例えば、スフィア・ソースを作成し、それをクリップします。
5. 最小化していた Python シェルを元に戻し、Stop Trace をクリックします。
6. Show Trace をクリックします。今トレースされた Python スクリプトが、Python のインタラクティブ・シェルに現れます。
7. Save Trace をクリックします。ハードディスクのどこかに適宜、Python スクリプトを保存します。

もし ParaView の Python バインディングをまだ学んでいないとしても、トレースされたスクリプトで実行されるコマンドには見覚えがあるでしょう。ひとたびハードディスクに保存されれば、お好みのエディタでスクリプトを編集することももちろん可能です。最終的なスクリプトは、`pvpython` または `pvbatch` に読込ませ、完全に自動化された可視化を行うことができます。Python シェルのダイアログ中の Run Script ボタンをクリックすることで、このスクリプトを実行することもできますが、このスクリプトをマクロに割当てて、ボタンのクリック一発でいつでも実行可能とすることもできます。 ♦

4.3 マクロ

ParaView の動作をカスタマイズするための簡単ながら強力な方法は、**マクロ**を作成することです。マクロとは、メニュー・バーのエントリやツール・バーのボタンによって起動することのできる、自動化されたスクリプトです。Python シェルによって、あらゆる Python スクリプトをマクロとして割当てることが可能です。

演習 4.2: マクロの追加

この演習は、演習 4.1の続きです。この演習を始める前に、演習 4.1を完了させておく必要があります。

1. もし現在開いていなければ、メニューから Tools → Python Shell を選択して ParaView の GUI 上の Python シェルを開いて下さい。
2. Python シェルの Directory View タブをクリックします。
3. ... ボタンをクリックし、演習 4.1でスクリプトを保存したディレクトリを開きます。
4. 保存したスクリプトを開き、Add to macros をクリックします。

この時点で、ツール・バーにマクロが追加されたことがお判りでしょう。デフォルトでは、マクロのツールバー・ボタンは左端の中段に置かれます。GUI 上のスペースが足りないようなら、このボタンを見るのにツールバーを適宜動かさなければならないかもしれません。また、Macros メニューに追加されたことも、お判りいただけるでしょう。

5. Python シェルを閉じます。
6. メニューから Edit → Delete All を選ぶか、  ボタンを押して、作成したパイプラインを削除します。
7. ツールバー・ボタンをクリックするか、Macros メニューで選択し、マクロをアクティブにします。

この例では、マクロににによってまっさらな状態から何かを作成しました。これは、毎度同様にして何らかのデータを読込む場合には有用です。また、既に存在するデータに対して適用されるフィルタの作成をトレースすることもできます。このような種類のトレースから作成されるマクロでは、異なるデータからの同じ可視化を自動化することができます。 ♦

4.4 パイプラインの作成

まず、ParaView GUI の Python Trace 機能によって、多くの一般的な操作に対して Python スクリプトをとても簡単に作成できることに注意してください。Trace を使用するには、GUI の Python Shell ダイアログからトレースの記録を開始し、ParaView の GUI を用いて可視化を作成し、そしてトレースの記録を停止します。これによって、GUI で行った操作を再現する Python スクリプトが作成されます。このスクリプトには、今から述べるのと全く同じ種類の操作が含まれています。したがって、Trace の記録は Python インタフェイスによって同じ操作を行う方法を見出すための良い見本であり、逆に以下の解説はあらゆる Trace スクリプトの中身を理解するのに役立ちます。[2]

ParaView のためのあらゆる Python スクリプトが行わなければならない最初のことは、`paraview.simple` モジュールの読込みです。これは、以下を実行することによって行われます。

```
from paraview.simple import *
```

一般に、このコマンドはあらゆる ParaView の Python バッチ・スクリプトで行う必要があります。このコマンドは ParaView からスクリプティングのダイアログを起動した時には自動的に実行されますが、(`pvpython` と `pvbatch` を含む) その他のプログラムで Python インタプリタを使用する際には、明示的に行う必要があります。

`paraview.simple` モジュールによって、ParaView で定義されているあらゆるソース、読込みオブジェクト、フィルタ、および書出しオブジェクトに対応する Python の関数が定義されます。この関数名は、GUI のメニューで表示されるそれらの名前から、空白と特殊文字を除いたものとなります。例えば、`Sphere` 関数は GUI の Sources → Sphere と対応し、`PlotOverLine` 関数は Filters → Data Analysis → Plot Over Line に対応します。それぞれの関数によってパイプライン・オブジェクトが作成され、(書出しオブジェクトを除いて) それがパイプライン・ブラウザに表示され、さらにそのパイプライン・オブジェクトのプロパティを取得、または操作するためのプロキシ・オブジェクトを返します。

`paraview.simple` モジュールには、その他の操作を行うための関数もあります。例えば、`Show` と `Hide` の対になった関数は、パイプライン・オブジェクトのビュー内での表示・非表示を切替えます。`Render` 関数は、ビューの再描画を行います。

演習 4.3: ソースの作成と表示

前節からの演習を行っていた場合は、ParaView をリセットする良い機会です。そのためには、  ボタンを押すのが最も簡単です。

もし、まだ行っていなければ、メニューから Tools → Python Shell を選択し、ParaView の GUI から Python シェルを開いて下さい。この操作によって Python シェルを起動した場合、

[2]訳注: 4.2節の重複と思われますが、再掲しておきます。

```
from paraview.simple import *
```

は既に実行されています。

以下を Python シェルに入力して、Sphere ソースを作成および表示させて下さい。

```
sphere = Sphere()
Show()
Render()
```

Sphere コマンドによって、スフィア (球) のパイプライン・オブジェクトが作成されます。Sphere コマンドを実行したら、パイプライン・ブラウザに項目が作成されたのがお判り頂けると思います。このパイプライン・オブジェクトのプロキシは、sphere 変数に格納されています。この変数は (まだ) 使用されていませんが、このようにしてパイプライン・オブジェクトへの参照を保持しておくのは良い習慣と言えます。

それに続く Show コマンドによって、ビュー内でのこのオブジェクトが表示状態となり、ついで Render によって、その結果が見えるようになります。この時点で、再び GUI を直接インタラクティブに操作できるようになります。ビューの視点の角度を、マウスによって変更してみて下さい。 ♦

演習 4.4: フィルタの作成および表示

フィルタの作成は、ソースの作成とほとんど同様です。デフォルトでは、GUI でフィルタを作成する時と同様に、最後に作成されたパイプライン・オブジェクトが、新たに作成されたフィルタへの入力としてセットされます。

この演習は、 演習 4.3の続きです。この演習を始める前に、演習 4.3を完了させておく必要があります。

Python シェルに、以下のスクリプトを入力して下さい。このスクリプトは、球を非表示にしてシュリンク・フィルタを追加し、それを表示します。

```
Hide()
shrink = Shrink()
Show()
Render()
```

球は、Shrink フィルタの出力によって置き換えられます。それによって球を構成する全てのポリゴンが縮小され、爆発したような見掛けになります。 ♦

ここまで、パイプラインを構成する際には、デフォルトの設定を適用してきました。しかしながら多くの場合、2章の演習で見られたように、オブジェクト・インスペクタを用いて設定を変更する必要があります。

Python によるスクリプティングでは、オブジェクトを作成する関数から返されるプロキシを使用して、パイプライン・オブジェクトを操作します。これらのプロキ

シは実際のところ、オブジェクト・インスペクタで設定するプロパティに対応するクラス属性 (プロパティ) を有する、Python のオブジェクトです。それらのプロパティは、オブジェクト・インスペクタの設定名から、スペースや Python オブジェクト名に使用できない文字を除いた名前を有します。それらのプロパティを設定するには、単に設定したい値を代入します。

演習 4.5: パイプライン・オブジェクトのプロパティを変更する

この演習は、演習 4.3と 4.4の続きです。この演習を始める前に、演習 4.3と 4.4を完了させておく必要があります。

ここまでに、sphere と shrink の 2 つの Python 変数を作成しました。これらは対応するパイプライン・オブジェクトのプロキシです。それではまず、以下のコマンドを Python シェルに入力し、球の Theta Resolution プロパティの値を取得します。

```
print sphere.ThetaResolution
```

Python インタプリタによって、8 の値が返される筈です。(なお、Python シェルは全てのコマンドの実行結果を出力するため、引数の Python オブジェクトを標準出力に表示させる print キーワードは本来、不要です。) このプロパティの値を変えて、球の赤道方向のポリゴンの数を倍にしてみましょう。

```
sphere.ThetaResolution = 16
Render()
```

シュリンク・フィルタが持っているプロパティはただ一つ、Shrink Factor です。この設定を調整すると、ポリゴンのサイズが大きく、または小さくなります。この値を変更して、ポリゴンを小さくしてみましょう。

```
shrink.ShrinkFactor = 0.25
Render()
```

Python のコマンドを入力してパイプライン・オブジェクトのプロパティを変更するのに合わせて、GUI のオブジェクト・インスペクタが更新されるのがお判り頂けると思います。 ♦

ここまで、分岐の無いパイプラインだけを作成してきました。これは単純でありながら一般的なケースであり、paraview.simple モジュールの他の多数の機能と同様、このようなケースにおける作業量を最小限にとどめつつ、さらに複雑なケースにも対応できるような明確な道筋を提供できるよう設計されています。ここまで分岐の無いパイプラインを構成してきたように、直前に作成されたオブジェクトは、ParaView によって自動的に次のフィルタの入力として接続されてきました。それによって、スクリプトはそれが行う操作の一連の流れとして読むことができるように

なっています。しかしながら、パイプラインに分岐が存在する場合には、フィルタの入力を明確に指定する必要があります。

演習 4.6: パイプラインを分岐する

この演習は、演習 4.3から 4.5までの続きです。この演習を始める前に、演習 4.3から 4.4までを完了させておく必要があります (演習 4.5は任意です)。

今までのところ、sphere と shrink という、2 つの Python 変数を作成しました。これらは、対応するパイプライン・オブジェクトのプロキシとなっています。ここで、球のワイヤーフレームを抽出する 2 つめのフィルタをスフィア・ソースに追加しましょう。以下を Python シェルに入力してください。

```
wireframe = ExtractEdges(Input=sphere)
Show()
Render()
```

スフィア・ソースに、Extract Edges フィルタが追加されました。元の球のワイヤーフレームと、ポリゴンが縮小された球が同時に表示されている筈です。

ここで、ExtractEdges 関数への引数として Input=sphere を与えることで、Extract Edges フィルタへの入力を明示的に指定していることに注意してください。ここで実際に行われていることは、オブジェクトを作成する際に、Input プロパティを与えることです。デフォルトの入力でオブジェクトを作成し、後で与えることも可能ですが、推奨されません。それは、全てのフィルタが全ての種類の入力を受付ける訳ではないためです[3]。もし最初に誤った種類の入力によってフィルタを作成した場合、Input プロパティを正しい種類の入力に変更する機会を得る前に、エラーメッセージが表示されることとなる可能性があります。

出力に 2 つのフィルタが接続されているスフィア・ソースは、**扇型に広がる**パイプラインの例です。このように 1 つの出力に複数のフィルタを接続することは、常に可能です。一方で、全てではないものの、いくつかのフィルタは入力に複数のフィルタを接続することが可能です。複数のフィルタを入力に接続することは、**扇型に束ねる**接続と言えます。ParaView の Python スクリプティングでは、扇型に束ねることは扇型に広げることと同様に、フィルタへの入力を明示的に指定することで行います。(1 つの入力ポートに) 複数の入力を接続するには、今まで 1 つの入力を指定してきたところに、かわりにパイプライン・オブジェクトのリストを指定します。例えば、シュリンク・フィルタとエッジの抽出フィルタの出力を、Group Datasets フィルタを使って束ねてみましょう。以下の行を Python シェルに入力して下さい。

```
group = GroupDatasets(Input=[shrink,wireframe])
Show()
```

[3]訳注: たとえば、ParticleTracer フィルタへの入力は、Temporal タイプのデータセットである必要があります。

シュリンク・フィルタとエッジの抽出フィルタの結果を表示させておく理由はもはやありませんので、非表示にしましょう。デフォルトでは、Show 関数と Hide 関数は、(フィルタを作成する際のデフォルトの入力と同様に) 最後に作成されたパイプライン・オブジェクトに対して作用しますが、引数に与えることで明示的にオブジェクトを選択することができます。シュリンク・フィルタとエッジの抽出フィルタの結果を非表示とするには、以下を Python シェルに入力してください[4]。

```
Hide(shrink)
Hide(wireframe)
Render()
```

♦

前述の演習では、オブジェクトを作成する関数の引数を Input=⟨**入力オブジェクト**⟩ とすることで、Input プロパティを指定できることを述べました。一般的にはどのようなプロパティでも、⟨**プロパティ名**⟩=⟨**プロパティの値**⟩ と指定することで、オブジェクトの作成時に指定することができます。例えば、以下のような行によって、球の作成時に Theta Resolution と Phi Resolution の双方を指定することができます。

```
sphere = Sphere(ThetaResolution=360, PhiResolution=180)
```

4.5 アクティブなオブジェクト

ParaView の GUI を使った経験が少しでもあれば、アクティブなオブジェクトの概念には既に慣れていることでしょう。GUI によって可視化パイプラインの作成や操作を行うには、まずパイプライン・ブラウザでオブジェクトを選択する必要があります。それ以外のオブジェクト・インスペクタのような GUI パネルは、アクティブなオブジェクトが何であるかによって変わります。アクティブなオブジェクトは、フィルタの追加のような操作でも、デフォルトのオブジェクトとして使用されます。

Python によるバッチ・スクリプティングにおいても、アクティブ・オブジェクトの概念があります。実際、GUI と Python インタプリタを同時に使用する場合には、両者は同じアクティブなオブジェクトを共有しています。前節でフィルタを作成した際、それらのフィルタに与えられるデフォルトの入力が、実はアクティブなオブジェクトです。新たなパイプライン・オブジェクトを作成したとき、(GUI でオブジェクトを作成した時と同じように) その新たなオブジェクトがアクティブとなりました。

アクティブなオブジェクトは、GetActiveSource および SetActiveSource 関数によって、それぞれ取得および設定することができます。また、GetSources 関数によって、全てのパイプライン・オブジェクトのリストを取得することも可能です。GUI のパイプライン・ブラウザで新たなオブジェクトをクリックすると、Python でのアクティ

[4]訳注: Group Datasets の実行によって二重に表示されていたオブジェクトが非表示となるだけなので、ビューの見掛けは変わりません。

ブなオブジェクトもそのオブジェクトに変更されます。同様に、SetActiveSource を Python から呼び出すと、パイプライン・ブラウザでは対応する項目がハイライトされます。

演習 4.7: アクティブなパイプライン・オブジェクトを試す

この演習は、前節の演習のつづきです。ただし、好みに応じて、いずれかのオブジェクトを適宜作成して、この続きを行うことも可能です。

以下によって、アクティブ・オブジェクトについて色々試してみて下さい。

- GetSources() を呼び出して、オブジェクトのリストを取得して下さい。そのリストの中から、作成したソースとフィルタを探して下さい。
- GetActiveSource() を呼び出して、アクティブなオブジェクトを取得して下さい。それを、パイプライン・ブラウザで選択されているオブジェクトと比べて下さい。
- パイプライン・ブラウザでいずれかの新たなオブジェクトを選択し、GetActiveSource() を再び呼び出して下さい。
- SetActiveSource 関数によって、アクティブなオブジェクトを変更して下さい。パイプライン・ブラウザで、その変更を観察して下さい。

♦

アクティブなパイプライン・オブジェクトを保持するのに加えて、ParaView の Python スクリプティングでは、アクティブなビューも保持しています。ParaView のユーザであるということで、既に複数ビューとアクティブ・ビューの概念にも慣れていることでしょう。アクティブなビューは、GUI では青いボーダーラインでマーキングされています。Python 関数の GetActiveView と SetActiveView によって、アクティブなビューを問い合わせたり、変更することができます。パイプライン・オブジェクトと同様に、アクティブなビューは GUI と Python インタプリタの間で同期しています。

4.6 オンライン・ヘルプ

The ParaView Book や ParaView Wiki での似たような使用説明と同様に、このチュートリアルは、Python によるバッチ処理スクリプトを理解し、また作成するために必要な重要な概念を提示することを目的に作成されています。利用可能な関数、クラス、プロパティの一覧などを含む詳細なドキュメントは、ParaView をビルドする過程において作成され、ParaView アプリケーションのオンラインヘルプとして提供されています。このようにして、ドキュメントがいかなるバージョンの ParaView であっても最新であり、容易にアクセス可能であることを確実としています。

ParaView の Python バインディングでは、help 内蔵関数を利用しています。この関数はあらゆる Python オブジェクトを引数に取り、そのオブジェクトに関する何らかのドキュメントを返します。例えば、以下を入力すると、

```
help(paraview.simple)
```

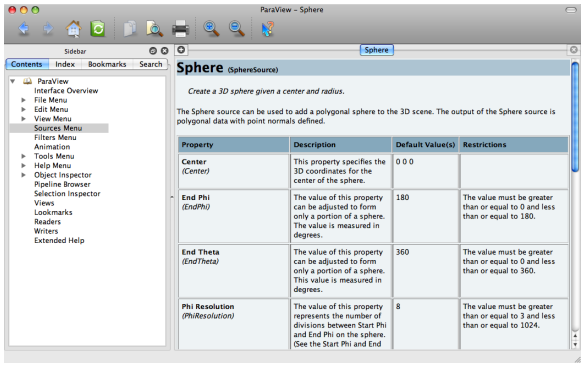
paraview.simple についての簡単な記述と、paraview.simple モジュールに含まれる全ての関数のリストを、それぞれの概要とともに返します。さらに、たとえば、

```
help(Sphere)
sphere = Sphere()
help(sphere)
```

とすると、最初の行では Sphere 関数に関するヘルプを表示し、つぎに Sphere 関数を使用してオブジェクトを作成し、さらに Sphere 関数によって返されたオブジェクトに関するヘルプを (そのプロキシの全てのプロパティのリストとともに) 表示します。

オブジェクト・インスペクタの Properties タブに表示されるパネルのほとんどは、Python クラスを構築するのと同じ仕組みによって、自動的に生成されます。(使いやすさを向上させるため、少数ながら例外的に、カスタム・パネルを使用しているものもあります。) したがって、オブジェクト・インスペクタ上の名前が付けられたウイジェットには多くの場合、Python オブジェクトにも同じ名前の対応するプロパティが存在しています。

GUI にパイプライン・オブジェクトのカスタム・パネルが含まれるか否かにかかわらず、そのオブジェクトのプロパティに関する情報を、GUI のオンライン・ヘルプで得ることができます。いつものように、ツールバー・ボタンの  によって、ヘルプを表示させてください。ヘルプの Contents の中の Sources Menu、Filters Menu、Readers および Writers 項目の下には、全ての利用可能なオブジェクトに関するドキュメントがあります。それぞれの項目には、そのタイプのオブジェクトのリストがあります。いずれのオブジェクトをクリックしても、Python から設定可能なプロパティのリストを得ることができます。



4.7 ファイルからの読込み

ParaView の GUI でファイルを開くことと同等の処理は、Python スクリプティングでは読込みオブジェクトを作成することです。読込みオブジェクトは、ソースやフィルタとほぼ同様にして作成します。`paraview.simple` には、それぞれの読込みオブジェクトの種類について、パイプライン・オブジェクトを作成してプロキシ・オブジェクトを返す関数が含まれています。あらゆる読込みオブジェクトは、以下のようにして、あるいは、さらに簡単には `reader = OpenDataFile(`⟨*filename*⟩`)` を呼び出すことによってインスタンス化することができます。

全ての読込みオブジェクトには少なくとも、ファイル名を指定するための (GUI からは見えない) プロパティがあります。このプロパティは慣習的に、`FileName` または `FileNames` と呼ばれています。読込みオブジェクトを作成する際には、オブジェクトのコンストラクタへの引数で `FileName=`⟨**ファイルへのフル・パス**⟩ のように記述することで、必ず有効なファイル名を与える必要があります。有効なファイル名が与えられないと、読込みオブジェクトは多くの場合、正しく初期化されません。

演習 4.8: 読込みオブジェクトの作成

本演習ではまっさらな状態から可視化を行いますので、ここまでの演習を行ってこられた場合は、ParaView を再起動するのにちょうど良いタイミングでしょう。そのための最も簡単な方法は、  ボタンを押すことです。また、Python シェルも必要です。もしまだ行っていなければ、メニューから Tools → Python Shell として Python シェルを開いて下さい。

この演習では、Python シェルから disk_out_ref.ex2 を読込みます。コンピュータ内の disk_out_ref.ex2 ファイルを探し、そのパスを Python シェルに入力するかコピーできるようにして下さい。このファイルを以後、⟨*path*⟩`/disk_out_ref.ex2` とします。

以下を Python シェルに入力し、ファイル名を指定しつつ読込みオブジェクトを作成します。

```
reader = OpenDataFile('⟨path⟩/disk_out_ref.ex2')
Show()
Render()
```

♦

4.8 フィールドの属性を問合せる

全てのパイプライン・オブジェクトのためのプロキシは、そのクラスに特定のプロパティに加えて、共通のプロパティやメソッドを持っています。そのようなプロパティの中でも特に重要な 2 つのプロパティは、`PointData` プロパティと `CellData` プ

ロパティです。これらのプロパティは、Python における連想配列型である**辞書**のように振舞います。すなわち、(文字列で表された) 変数名を、フィールドの何らかの特性を保持する ArrayInformation オブジェクトにマッピングします。ここで特記しておくべきは、ArrayInformation のメソッドである GetName で、これはフィールドの名前を返します。GetNumberOfComponents メソッドは、それぞれのフィールド値の成分の数 (スカラなら 1、ベクトルならそれ以上) を返します。GetRange メソッドは、ある特定の成分の最小値および最大値を返します。

演習 4.9: フィールド情報の取得

この演習は演習 4.8のつづきです。この演習を始める前に、演習 4.8を完了する必要があります。

まず、節点型データへのハンドルを取得し、データが持つ全ての節点型フィールドを表示します。

```
pd = reader.PointData
print pd.keys()
```

“Pres” と “V” フィールドに関する情報を取得します。

```
print pd['Pres'].GetNumberOfComponents()
print pd['Pres'].GetRange()
print pd['V'].GetNumberOfComponents()
```

それでは、もう少し凝ったことを試してみましょう。Python の for 構文を使って、全ての配列の全ての成分について、値の範囲を表示させます。

```
for ai in pd.values():
    print ai.GetName(), ai.GetNumberOfComponents(),
    for i in xrange(ai.GetNumberOfComponents()):
        print ai.GetRange(i),
    print
```

♦

4.9 レプレゼンテーション

レプレゼンテーションは、パイプライン・オブジェクト内のデータとビューを結びつける役割を果たします。レプレゼンテーションによって、どのようにデータセットがビュー内で描画されるかが管理されます。レプレゼンテーションによって、配色やライティングのようなレンダリングにおけるプロパティに加え、データの描画に用いられる内部的なレンダリング・オブジェクトが定義および管理されます。GUI

の Display パネルにおいて設定可能なパラメータは、レプレゼンテーションによって決まります。全てのパイプライン・オブジェクトとビューのペアについて、個別のレプレゼンテーション・オブジェクトのインスタンスが存在します。これは、それぞれのビューがデータを異なったレプレゼンテーションで表示できるようにするためです。

レプレゼンテーションは、自動的に作成されます。GetRepresentation 関数によって、レプレゼンテーション・オブジェクトへのプロキシを取得することができます。引数が何も無ければ、この関数はアクティブなパイプライン・オブジェクトおよびアクティブなビューに関するレプレゼンテーションを返します。パイプライン・オブジェクトまたはビュー、あるいはそれら両方を指定することも可能です。

演習 4.10: データの色づけ

この演習は、演習 4.8 (および、もし行っていた場合は、演習 4.9) のつづきです。まだ演習 4.8での Exodus ファイルを開いていなければ、この演習の前に演習 4.8を完了させて下さい。

形状の色を青色に変更し、鏡面反射によるハイライトをかなり強めに与えます (つまり、その形状に強い光沢を与えます)。Python シェルに以下を入力し、レプレゼンテーションを取得して材質の特性を変更します。

```
readerRep = GetRepresentation()
readerRep.DiffuseColor = [0, 0, 1]
readerRep.SpecularColor = [1, 1, 1]
readerRep.SpecularPower = 128
readerRep.Specular = 1
Render()
```

GUI 上でマウスによってカメラを回転し、鏡面反射によるハイライトの効果を確かめて下さい。

レプレゼンテーションは、フィールド変数によって色付けするためにも使うことができます。以下を Python シェルに入力し、“Pres” フィールド変数によってメッシュを色付けして下さい。

```
readerRep.ColorArrayName = 'Pres'
readerRep.LookupTable = MakeBlueToRedLT(0, 0.03)
Render()
```

♦

第5章　さらに情報を得るには

このチュートリアルに参加頂き、ありがとうございました。ParaViewを使って、大規模データの可視化を始めるのに充分なだけ勉強されたことでしょう。以下には、さらに情報を得るための情報源を挙げます。

The ParaView Guideは、ParaViewと一緒に持っておくのにとても良い情報源です。ここで学んだ以外の多くの使用法や、多くの機能に関するより詳細な説明が得られます。

Amy Henderson Squillacote. *The ParaView Guide*. Kitware, Inc., 2008. ISBN-10 1-930934-21-1.

ParaView Wikiは、ParaViewをセットアップして使用するのに役立つ情報が満載です。特に、並列ParaViewサーバをインストールしたい方は、そのためのビルドおよびインストールのページを必ずご参照ください。

http://www.paraview.org/Wiki/ParaView

http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server

可視化や、ParaViewで利用可能なフィルタに関する詳細事項をさらに学ぶことに興味がある場合は、以下の可視化に関するテキストを入手することをご一考下さい。

Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit*. Kitware, Inc., fourth edition, 2006. ISBN 1-930934-19-X.

ParaViewをカスタマイズしようとされているのであれば、上記の本およびウエブ・ページに多くの情報があります。ParaViewが依存する可視化ライブラリのVTK、GUIライブラリのQtの使用法に関する情報については、下記の本が参考となります。

Kitware Inc. *The VTK Users Guide*. Kitware, Inc., 2006.

Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall, 2006. ISBN 0-13-187249-4 (邦訳: 杵渕 聡、杉田 研治 訳「入門 Qt 4 プログラミング」オライリー・ジャパン、2007年、ISBN 978-4-87311-344-9).

もし並列可視化の設計や VTK パイプラインの機能に興味があるのであれば、以下の技術論文があります。

James Ahrens, Charles Law, Will Schroeder, Ken Martin, and Michael Papka. “A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets.” Technical Report #LAUR-00-1620, Los Alamos National Laboratory, 2000.

James Ahrens, Kristi Brislawn, Ken Martin, Berk Geveci, C. Charles Law, and Michael Papka. “Large-Scale Data Visualization Using Parallel Data Streaming.” *IEEE Computer Graphics and Applications*, 21(4): 34–41, July/August 2001.

Andy Cedilnik, Berk Geveci, Kenneth Moreland, James Ahrens, and Jean Farve. “Remote Large Data Visualization in the ParaView Framework.” *Eurographics Parallel Graphics and Visualization 2006*, pg. 163–170, May 2006.

James P. Ahrens, Nehal Desai, Patrick S. McCormic, Ken Martin, and Jonathan Woodring. “A Modular, Extensible Visualization System Architecture for Culled, Prioritized Data Streaming.” *Visualization and Data Analysis 2007, Proceedings of SPIE-IS&T Electronic Imaging*, pg 64950I-1–12, January 2007.

John Biddiscombe, Berk Geveci, Ken Martin, Kenneth Moreland, and David Thompson. “Time Dependent Processing in a Parallel Pipeline Architecture.” *IEEE Visualization 2007*. October 2007.

もし ParaView の並列レンダリングのアルゴリズムや構造に興味があるのであれば、それらに関する技術論文もまた多数あります。

Kenneth Moreland, Brian Wylie, and Constantine Pavlakos. “Sort-Last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays.” *Proceedings of IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pg. 85–92, October 2001.

Kenneth Moreland and David Thompson. “From Cluster to Wall with VTK.” *Proceddings of IEEE 2003 Symposium on Parallel and Large-Data Visualization and Graphics*, pg. 25–31, October 2003.

Kenneth Moreland, Lisa Avila, and Lee Ann Fisk. “Parallel Unstructured Volume Rendering in ParaView.” *Visualization and Data Analysis 2007, Proceedings of SPIE-IS&T Electronic Imaging*, pg. 64950F-1–12, January 2007.

謝辞

Amy Squillacote 氏と David DeMarle 氏には、このチュートリアルに教材を提供して下さったことに感謝いたします。そしてもちろん、Kitware、サンディア、CSimSoft の各位には、多大なる労力によって ParaView を現在のようにして下さったことに感謝申し上げます。

サンディアは、DE-AC04-94AL85000 の契約に基づいて、米国エネルギー省国家核安全保障局のために、ロッキード・マーチン系列のサンディア・コーポレーションによって運営される、多数のプロジェクトからなる研究所です。

日本語版 謝辞

このドキュメントは、Kenneth Moreland 氏による、“The ParaView Tutorial version 3.8” の日本語訳です。原著者には、本ドキュメントの前々作となる “Large Scale Visualization with ParaView: Supercomputing 2008 Tutorial” の日本語訳をきっかけに、原文の LaTeX ソース及び画像ファイル一式を公開頂きました。感謝いたします。

索 引