

- [Skip to search form](#)
- [Skip to main content](#)
- [Go to the homepage](#)
- [Skip to service alerts](#)



[Boston University Information Services & Technology](#)

[Navigation menu](#)

Search this website

[Get Help](#)

- [Services](#)
- [Support](#)
- [About](#)

Important alerts

- [**Known Issues with BUworks in Procurement as of 12/11**](#)

Please be advised that the following are known issues within BUworks since system maintenance on 12/9/18. We are working to correct these as soon as possible. Procurement: Print Preview for Shopping Carts and Purchase Orders – unavailable in Firefox only (Chrome works) Procurement: Dropdown menus in Chrome do not respond to clicking – Works... [\[read more\]](#)

Information Messages

- [**Planned Maintenance: MyBU Applicant Link will be unavailable on Tuesday, November 11th from 12:00 pm through 5:00 pm**](#)

Planned Maintenance: MyBU Applicant Link will be unavailable on Tuesday, November 11th from 12:00 pm through 5:00 pm... [\[read more\]](#)

Breadcrumb navigation

- [Home](#)
- [Support](#)
- [Research Computing](#)
- [Training & Consulting](#)
- [Self-Paced Tutorials](#)
- [Using ParaView ... Scientific Data](#)

Using ParaView to Visualize Scientific Data (online tutorial)

Table of Contents

[**Introduction**](#)

A brief introduction.

[**Getting started**](#)

A few basic hints that will help you get started.

[**User Interface**](#)

[Menus](#)

[Pipeline Browser](#)

[Object Inspector](#)

[**Visualization Pipeline**](#)

How ParaView transforms informational data into graphical data.

[**Loading Data**](#)

How to load data sets and state files into ParaView.

[**Scalar Visualization Algorithms**](#)

[Color Mapping](#)

[Contouring/Isosurface.](#)

[Vector Visualization Algorithms](#)

[Hedgehogs](#)

[Oriented Glyphs](#)

[Stream Lines](#)

[Modeling Visualization Algorithms](#)

[Cutting/Slicing](#)

[Clipping](#)

[Additional Help](#)

[References](#)

Introduction

ParaView is an open-source, multi-platform data analysis and visualization application built on top of [VTK](#). ParaView offers non-programmers much of the capability of VTK without requiring them to write programs. Through exploration and interaction users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. ParaView provides a comprehensive suite of visualization algorithms and supports many different file formats for both loading and exporting data sets.

This tutorial is organized around a set of ParaView visualization examples which are stored as ParaView state files (filename.pvsm). The examples we will be using were originally developed for the [2008 Workshop on Scientific Visualization](#) and are based upon the example programs and data from Kitware. The workshop examples are available for download here: [Tar file of the ParaView workshop examples](#). See the [ParaView Examples](#) web page for more information.

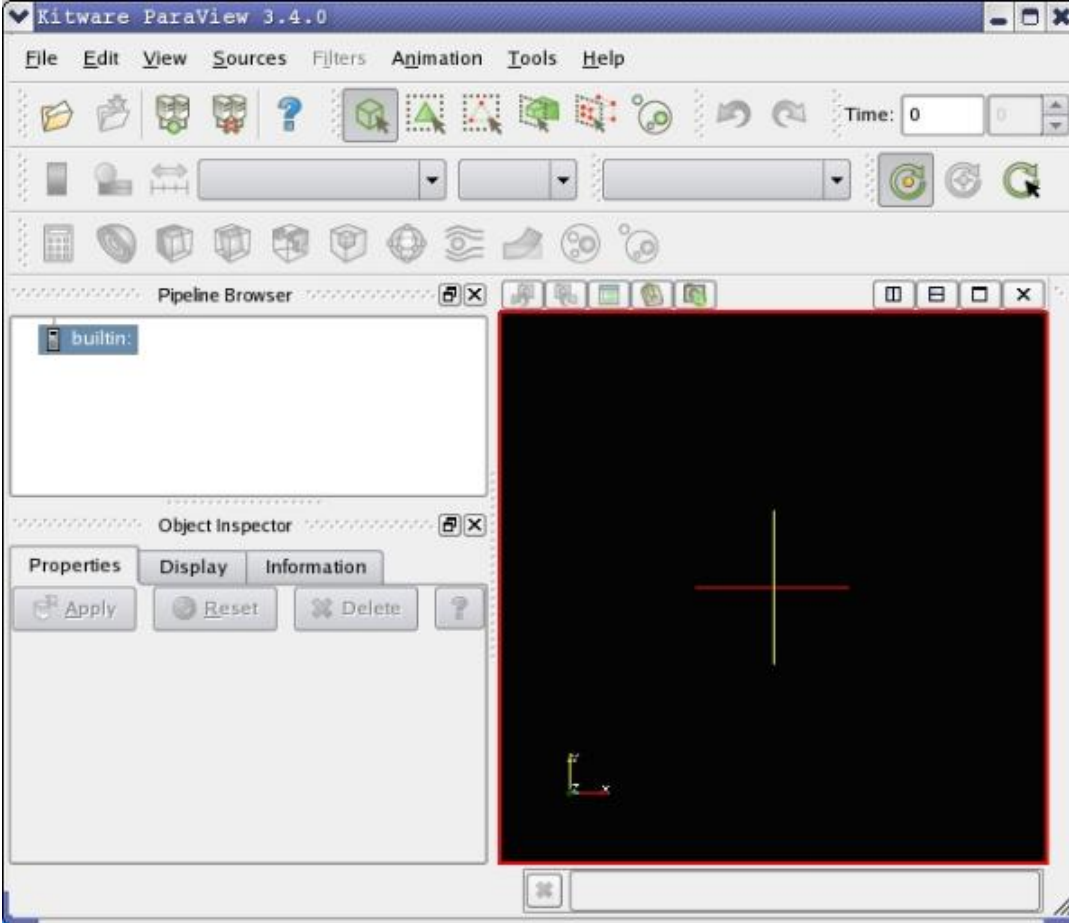
Getting started

For this tutorial we will be running ParaView locally in what is called “stand-alone” mode. ParaView can also be run in a client server configuration, often used for viewing extremely large data sets, but this is beyond the scope of this introductory tutorial. Before starting the tutorial you should download and untar the workshop examples file. Once this has been done, you should have a materials directory with both a Data and a Demos subdirectory. You can **cd** into any of the subdirectories of the materials/Demos directory and load a ParaView example by typing **paraview** followed by **--state=** and then the example’s file name. For example:

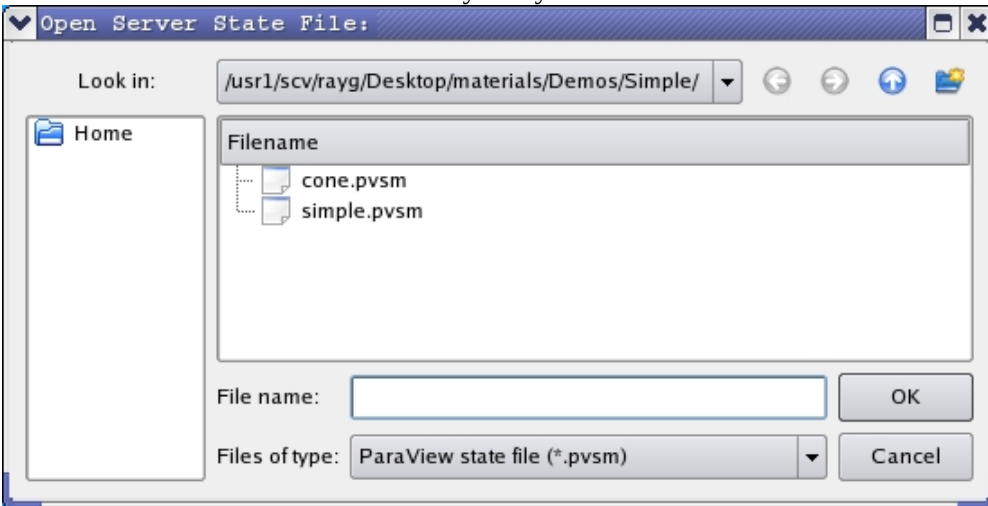
```
sccl% cd materials/Demos/Simple/  
sccl% paraview --state=cone.pvsm
```

Another way to load an example file is to first run ParaView without any arguments:

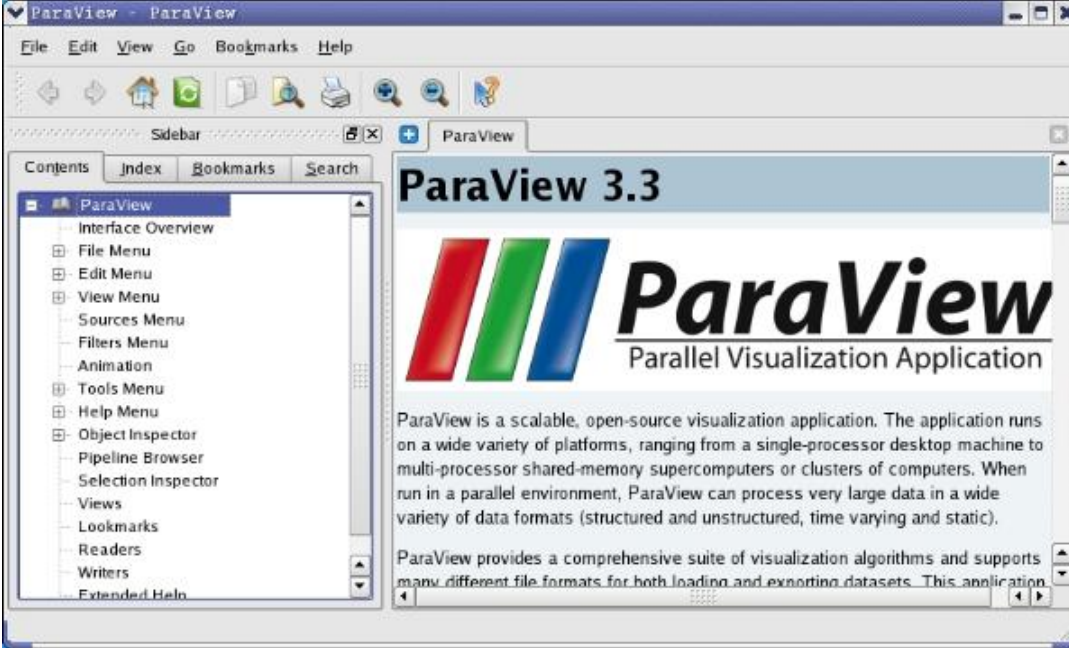
```
scc% cd materials/Demos/Simple/  
sccl% paraview
```



and then from within the ParaView application go to the File menu and select Load State. This will open a file dialog window, providing a list of state files in the current directory that you can load:



The most effective way for you to go through this tutorial is to have ParaView running in a window next to this tutorial. We encourage you to play and experiment with ParaView as we discuss the various topics in this tutorial. It can also be helpful for you to browse the ParaView documentation as we talk about various features of ParaView. To see the documentation from within ParaView go to the Help menu and select Help. This will open a window, providing access to all of the manuals that come with ParaView:



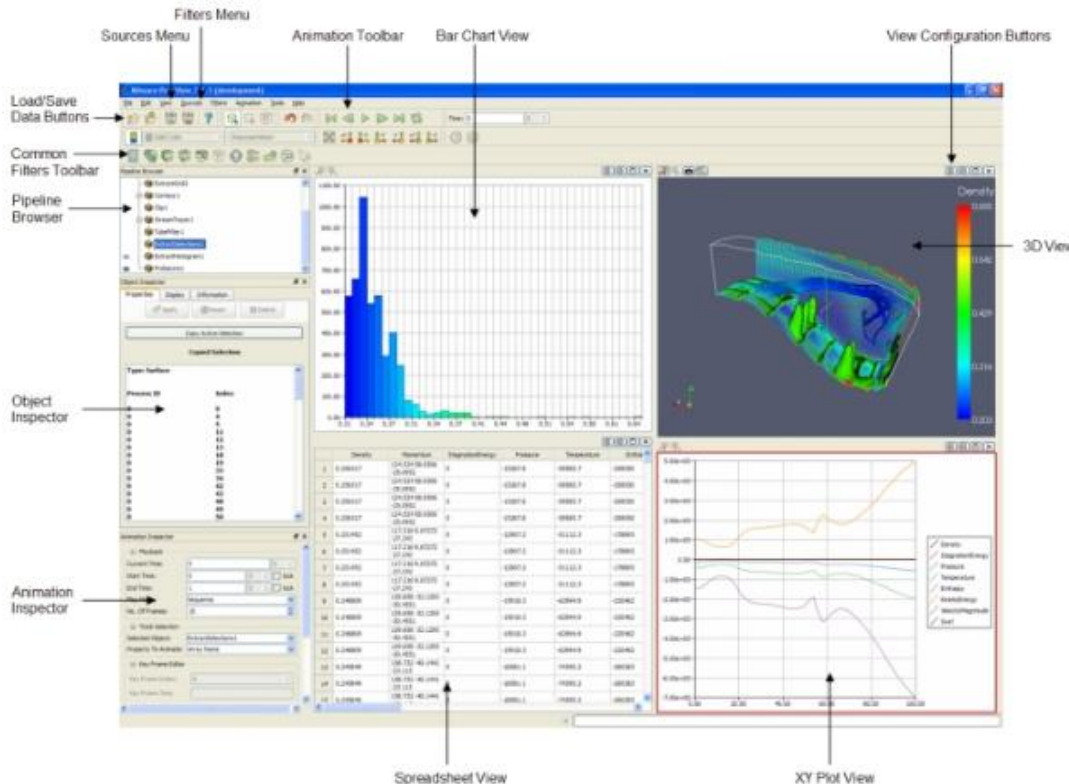
You can also visit the [online version](#) of the ParaView documentation.

Boston University Research Computing users, please go to the [RCS ParaView Help Page](#) for information specific to the ParaView installation at this site. The help page will tell you on what machines ParaView is available, how to set up your environment, how to set your display, and where additional documentation is located.

Before we can talk about how to visualize data sets with ParaView, we first need to talk about ParaView's user interface and ParaView's visualization pipeline.

User Interface

As an application, ParaView comes with a complete graphical user interface. Shown below is an example of a ParaView start-up state with an example data set and visualization pipeline. The parts of the main window are labeled and most of the major ones will be discussed below.



Menus

There are eight menus in the menu bar at the top of the main window. These are File, Edit, View, Sources, Filters, Animation, Tools,

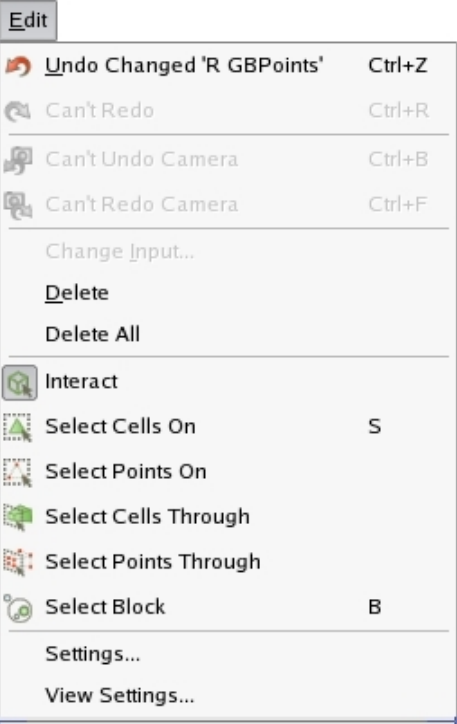
and Help.

File Edit View Sources Filters Animation Tools Help

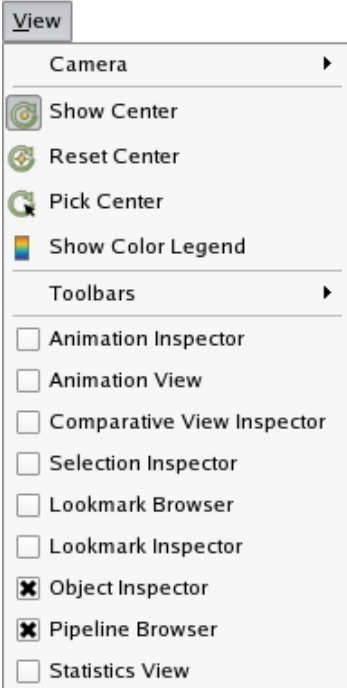
The File menu handles various tasks such as opening data files, saving data files, loading state files, saving state files, saving screenshots, saving animations, and fileserver connections.



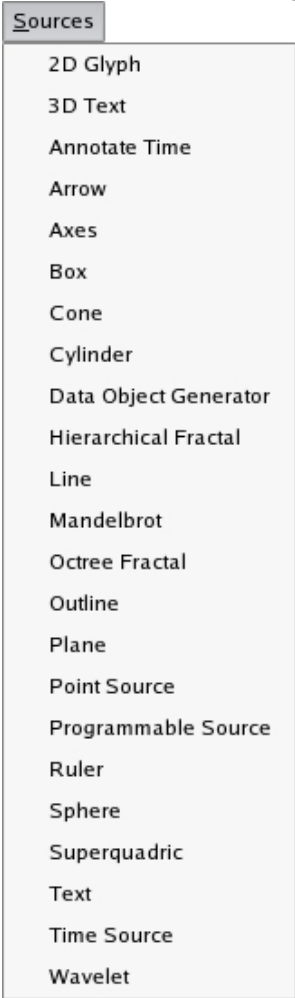
The Edit menu provides control settings for ParaView, undo and redo functions, allows you to change pipeline topology, and allows you to configure how the mouse interacts with the 3D view.



The View menu allows you to modify the camera and center of rotation for the 3D view. The view menu also allows you to toggle the visibility of the toolbars, inspectors, and views.

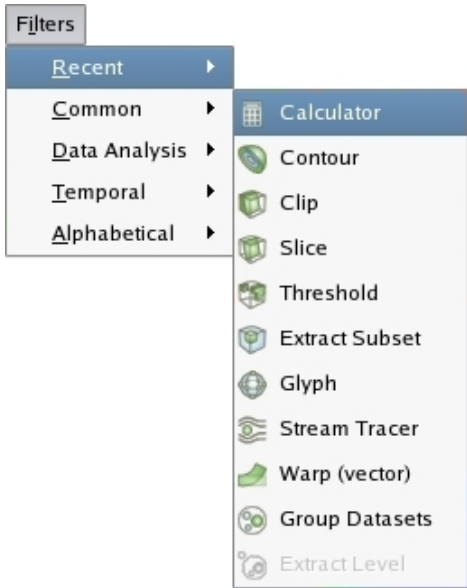


The Sources menu shows the various sources you can use to create a data set from within ParaView itself. A source is an object that creates data without using another data set or a data file as input.

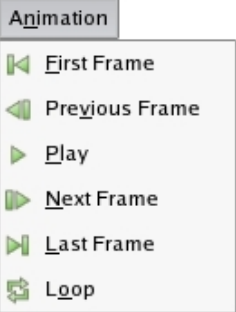


The Filters menu provides a list of available filters you can use to process data sets. The lists are organized by recent, common, data analysis, temporal, and alphabetical. The most commonly used filters, located under the *Common* subdirectory, are also located on the Common Filters Toolbar. The filters are context sensitive and will only be available for selection if an appropriate data set has been

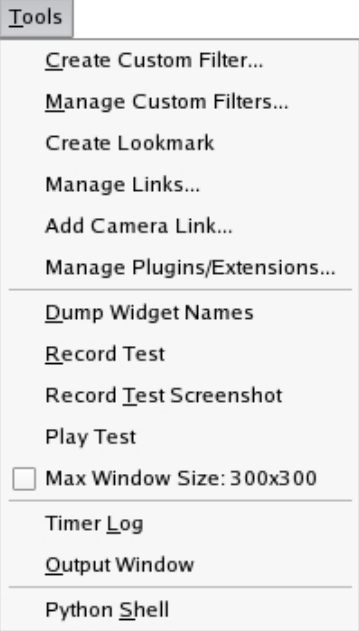
loaded first and selected in the **Pipeline Browser**.



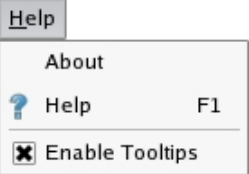
The Animation menu provides controls for moving between frames in an animation that has been previously created.



The Tools menu provides access to various tasks such as creating and managing custom filters, creating a lookmark (a particular camera view), linking properties of one object to another object, linking the camera in one view window to a camera in another view window, managing plugins, testing, and debugging.

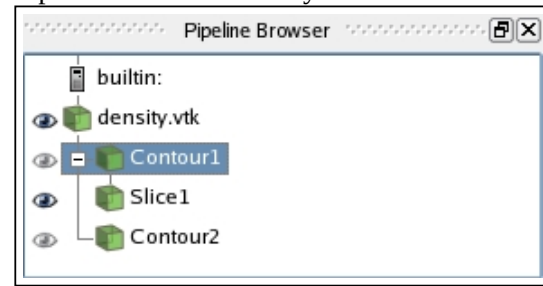


The Help menu provides information on the ParaView version, information on client server connections, and provides access to the online manual.



Pipeline Browser

The Pipeline Browser, located in the upper left corner of the user interface, allows you to both build a visualization pipeline and to see information about the current visualization pipeline. By allowing you to select, add, and delete various readers, sources, and filters, the Pipeline Browser allows you to interact with the current visualization pipeline.



At the top of the pipeline browser is the name of the server to which ParaView is connected. If you are running ParaView in stand-alone mode (which you should be doing for this tutorial) you will see the server name of “builtin.” Below the server name is a tree structure representing each of the reader, source, and filter objects that are in the visualization pipeline. In the example above, **density.vtk** represents the reader for the data set. **Contour1** and **Contour2** are filters, each applied separately to the output of the reader (as shown by the lines connecting each of the filters directly to the reader density.vtk). **Slice1** is a filter that is applied to the output of the filter named Contour1 (as shown by the line connecting them).

To the left of each object in the pipeline is an “eye” toggle icon which both indicates and controls (by left-clicking on the icon) whether or not the object is shown in the view window. In the above example density.vtk and Slice1 are shown in the view window but the other objects are not.

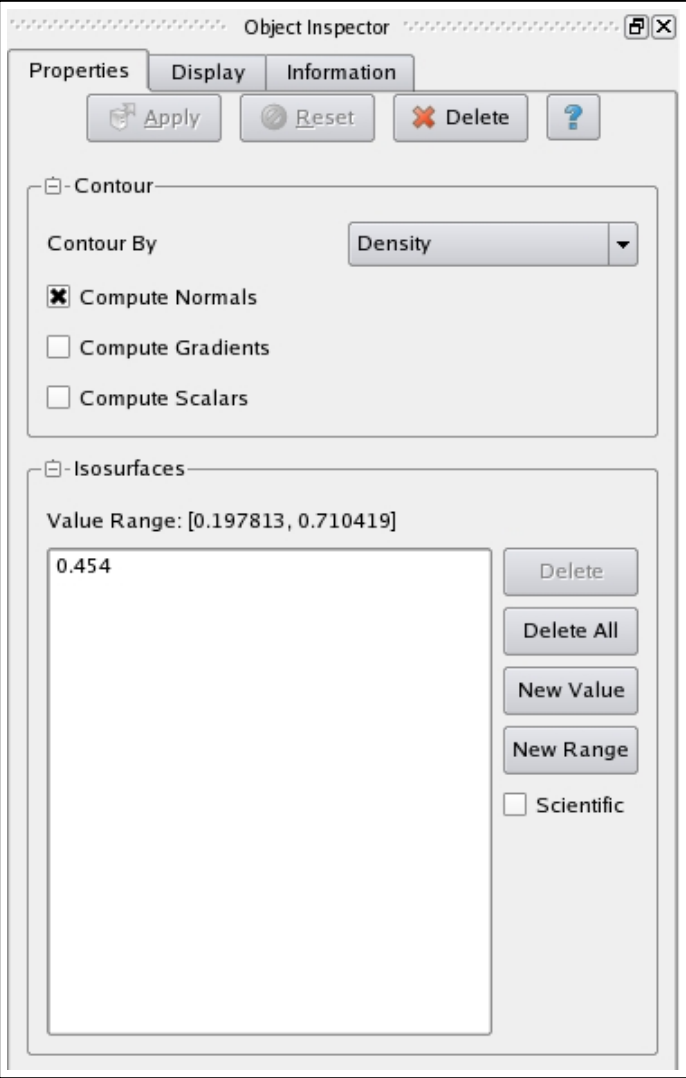
To apply a filter to an object, first select the object in the Pipeline Browser and then choose a filter from the Filters menu. After selecting a filter from the Filters menu, an icon representing the filter will show up in the browser and will be connected to the object to which it is being applied. Left-clicking on one of the entries in the pipeline selects that object and displays the appropriate Properties, Display, and Information tabs in the Object Inspector (see below). In the above example Contour1 is currently selected.

See [Visualization Pipeline](#) for more details on how ParaView creates and uses a visualization pipeline.

Object Inspector

The Object Inspector, located beneath the Pipeline Browser in the user interface, contains controls and information for the reader, source, or filter object selected in the Pipeline Browser. The content changes based upon the specific object selected. There are three tabs in the Object Inspector: Properties, Display, and Information.

The **Properties Tab** contains controls for specifying various parameters of the object selected in the Pipeline Browser. Here is an example of what is shown in the Properties Tab for a Contour filter:



The **Display Tab** contains controls for setting the appearance of the object selected in the Pipeline Browser. These controls are grouped into several sections: View, Color, Slice, Style, Edge Style, Annotation, Lighting, and Transformation. Here is an example of what is

shown in the Display Tab for a Contour filter:

Object Inspector

Properties

Display

Information

View

☐ Visible

Zoom To Data

Color

☒ Interpolate Colors

☒ Map Scalars

Apply Texture

None

Color by

☒ Solid Color

☐ Set Solid Color...

Slice

Slice Direction

Slice

0

Style

Representation

Surface

Interpolation

Gouraud

Material

None

Point size

5.00

Line width

1.00

Opacity

1.00

Volume mapper

Edge Style

☒ Set Edge Color...

Annotation

☐ Show cube axes

Edit

Lighting

Specular Intensity

0.10

Specular Focus

100

☒ Specular White

Transformation

Translate

0

0

0

Scale

1

1

1

Orientation

0

0

0

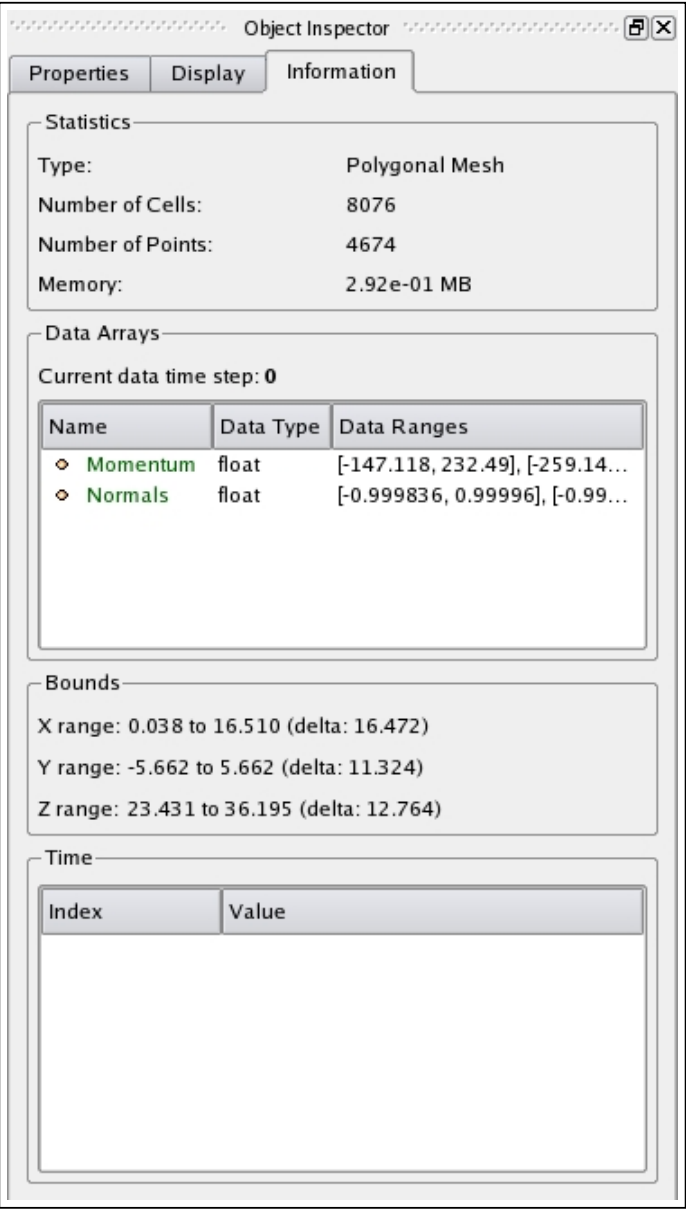
Origin

0

0

0

The **Information Tab** contains statistical information about the output of the object selected in the Pipeline Browser. Here is an example of what is shown in the Information Tab for a Contour filter:

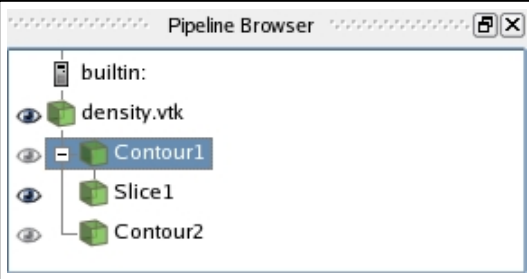


Visualization Pipeline

The underlying visualization pipeline that ParaView uses comes from [VTK](#). The visualization pipeline is responsible for constructing a geometric representation of the data set that is then rendered by the graphics hardware. The visualization pipeline transforms informational data into graphical data. ParaView like VTK uses a data flow approach to transform informational data into graphical data. There are two basic types of objects involved:

- **DataObjects**
 - represent data of various types: Image data, Rectilinear Grid, Structured Grid, Unstructured Grid, Polygonal Data
 - consist of geometry (point coordinates), topology (points or cells), and data attributes (Scalars, Vectors, Normals, Texture Coordinates, Tensors)
 - provides access to data
 - represented in ParaView by **readers** or **sources**
- **ProcessObjects**
 - filters which operate on data objects to produce new data objects
 - represent visualization algorithms
 - represented in ParaView by **filters**

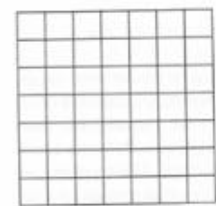
Data Objects and Process Objects are connected together to form a visualization pipeline which can be seen in the ParaView Pipeline Browser.



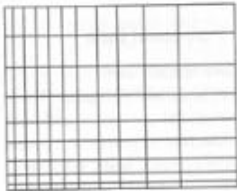
Data Set Types

ParaView as well as VTK data objects have both topological and geometrical structure and can represent several types of data:

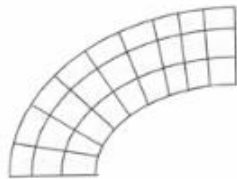
- **(a) Image/Volume Data**
 - regular in both topology and geometry
 - examples: lines, pixels, voxels
 - applications: imaging CT, MRI
- **(b) Rectilinear Grid**
 - regular topology but geometry only partially regular
 - examples: pixels, voxels
- **(c) Structured Grid**
 - regular topology and irregular geometry
 - examples: quadrilaterals, hexahedron
 - applications: fluid flow, heat transfer
- **(d) Unstructured Points**
 - no topology and irregular geometry
 - examples: vertex, polyvertex
 - applications: data with no inherent structure
- **(e) Polygonal Data**
 - irregular in both topology and geometry
 - examples: vertices, polyvertices, lines, polylines, polygons, triangle strips
 - applications: modelling
- **(f) Unstructured Grid**
 - irregular in both topology and geometry
 - examples: any combination of cells
 - applications: finite element analysis, structural design, vibration



(a) Image Data



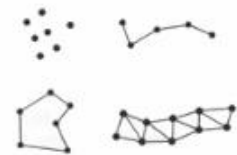
(b) Rectilinear Grid



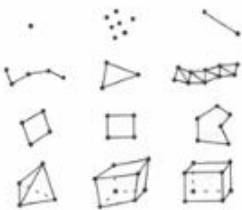
(c) Structured Grid



(d) Unstructured Points



(e) Polygonal Data



(f) Unstructured Grid

Data Attributes

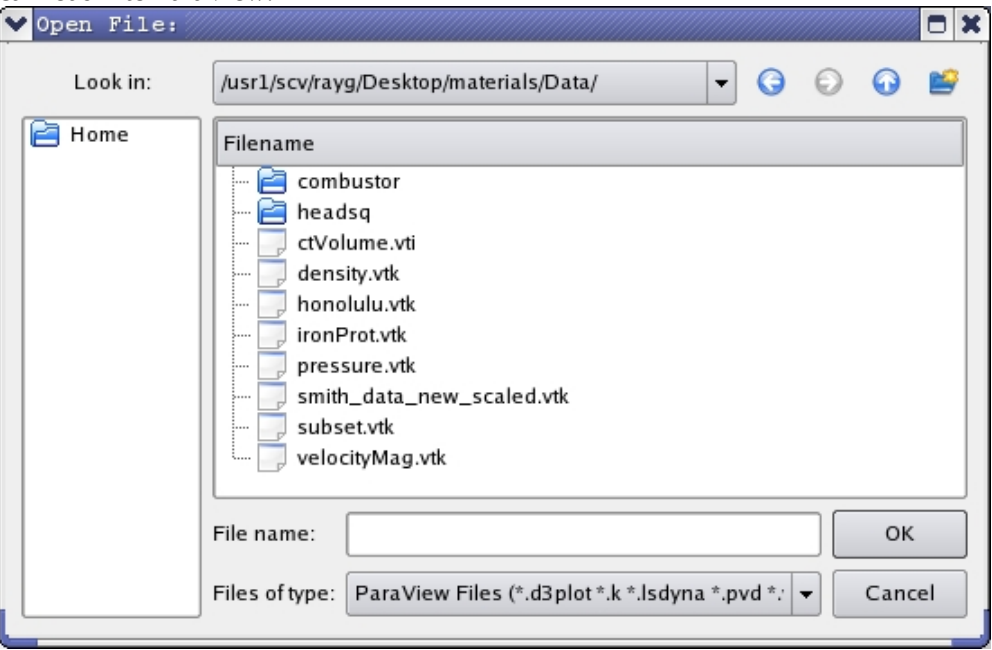
ParaView as well as VTK data objects also have attributes associated with the topological and geometrical organizing structure:

- **Scalars**
 - single valued
 - examples: temperature, pressure, density, elevation
- **Vectors**
 - magnitude and direction
 - examples: velocity, momentum
- **Normals**
 - direction vectors (have a magnitude of 1)
 - used for shading
- **Texture Coordinates**
 - used to map a point in Cartesian space into 1, 2, or 3D texture space
 - used for texture mapping of polygonal surfaces
 - volume visualization can use 3D textures
- **Tensors**
 - rank 0 (scalar), rank 1 (vector), rank 2 (matrix), rank 3 (3D rectangular array)

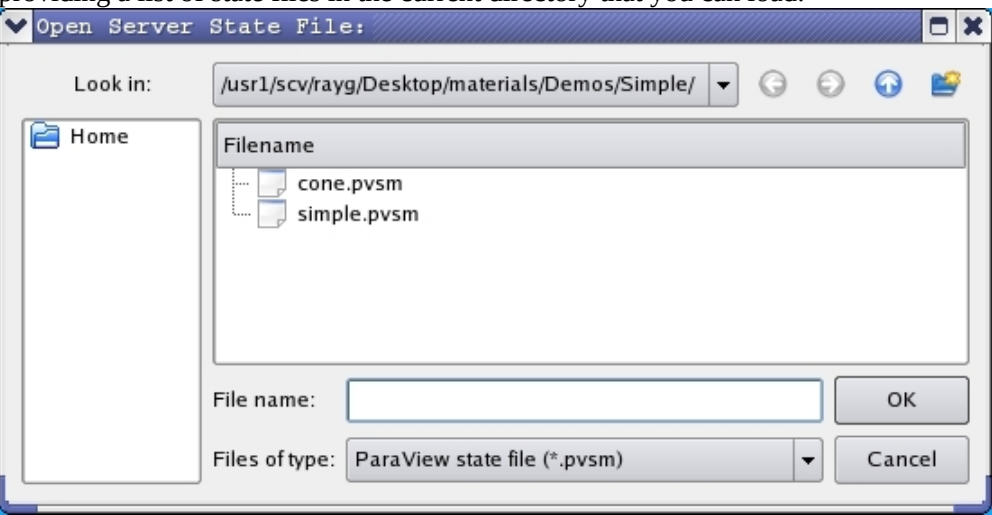
- examples: stress, strain

Loading Data

The first step in visualizing data with ParaView is to load the data set. This is easily accomplished using the **Open** option in the **File** menu. Go to the **File** menu and select **Open**. This will open a file dialog window, providing a list of data files in the current directory that you can load into ParaView:



Another option is to load a previously saved ParaView visualization pipeline which is called a ParaView state file. To load a ParaView state file, use the **Load State** option in the **File** menu. Go to the **File** menu and select **Load State**. This will open a file dialog window, providing a list of state files in the current directory that you can load:



ParaView supports many different data file formats. To see the list of available **readers**, look in the ParaView online documentation, use the ? option under the **Help** menu. For the purposes of this tutorial we will be using ParaView state files to demonstrate various visualization techniques. In the materials/Data directory of the Workshop files you untared, you will find several additional VTK/ParaView data files with which to experiment.

Scalar Visualization Algorithms

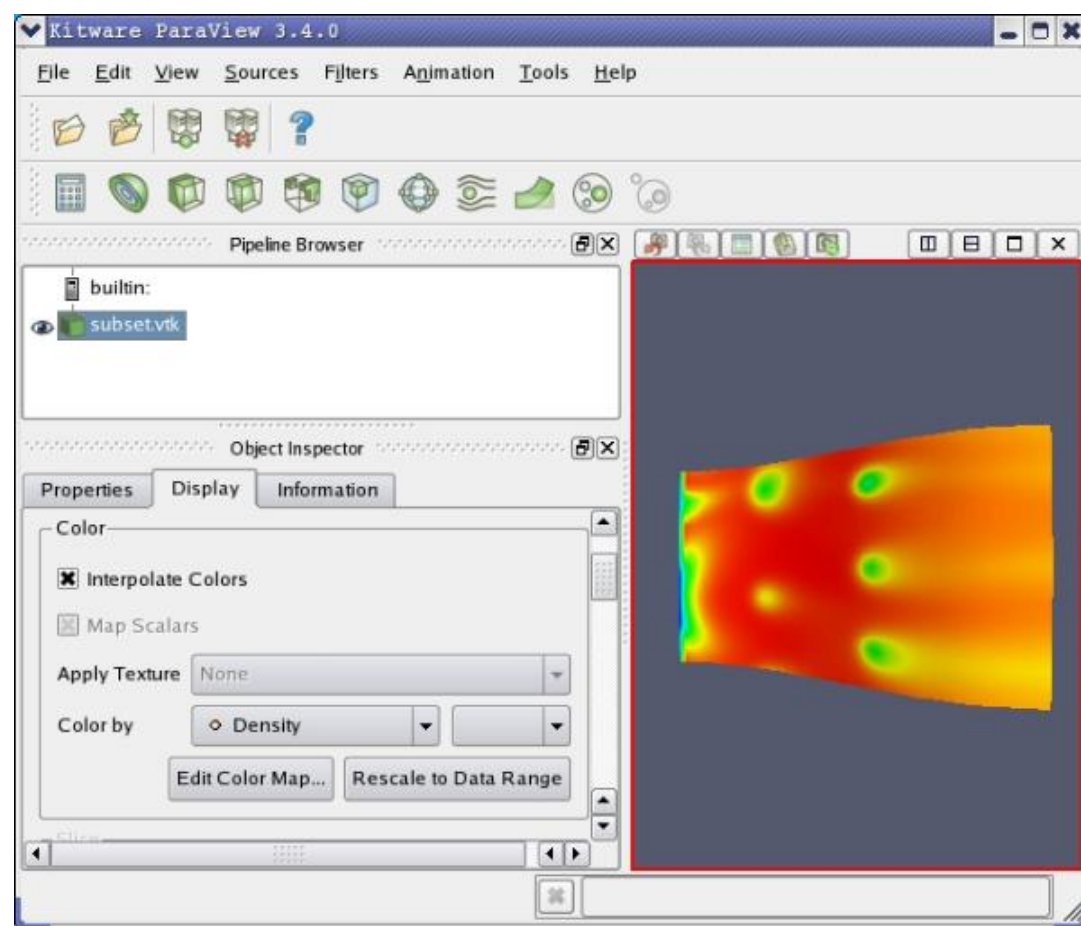
Scalars are single data values associated with each point/cell in the data set. There are many different algorithms to visualize scalar data. Two common algorithms are Color Mapping and Contouring.

Color Mapping

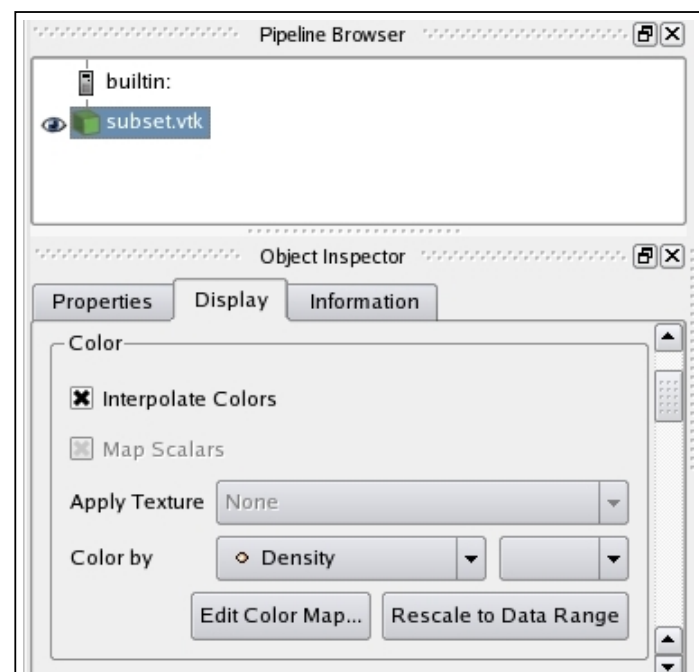
In color mapping, scalar values are mapped through a lookup table to a specific color. Scalar values are used as an index into a color lookup table. The basic idea is to specify a HSVA (Hue-Saturation-Value-Alpha) ramp and then generate the colors in the table by using linear interpolation into the HSVA space. By using the ParaView [Color Tools](#), you can set both the number of colors (size of the table) and the hue range for the color table.

To start load the colormap.pvsm statefile. The statefile is located in the materials/Demos/ColorMaps/ directory.

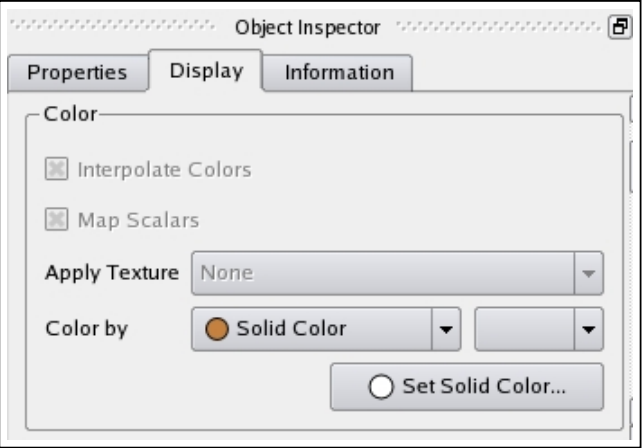
```
sccl% cd materials/Demos/ColorMaps
sccl% paraview --state=colormap.pvsm
```



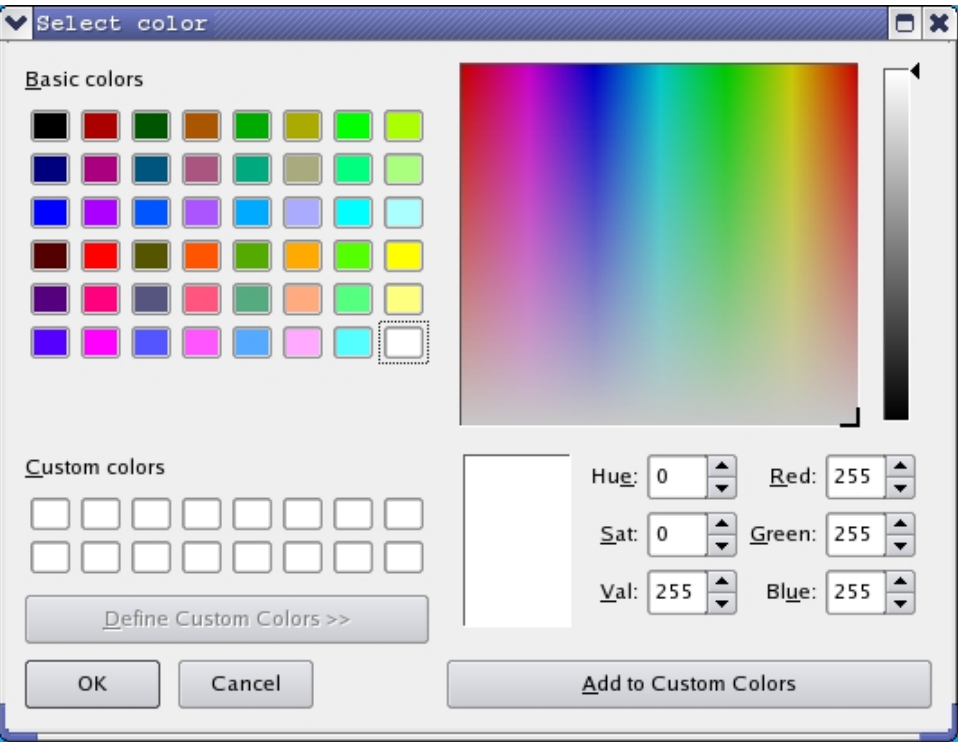
To control how color is mapped to the data set, first select the data reader (named subset.vtk) in the Pipeline Browser, and then go to the the color section in the Display Tab in the Object Inspector.



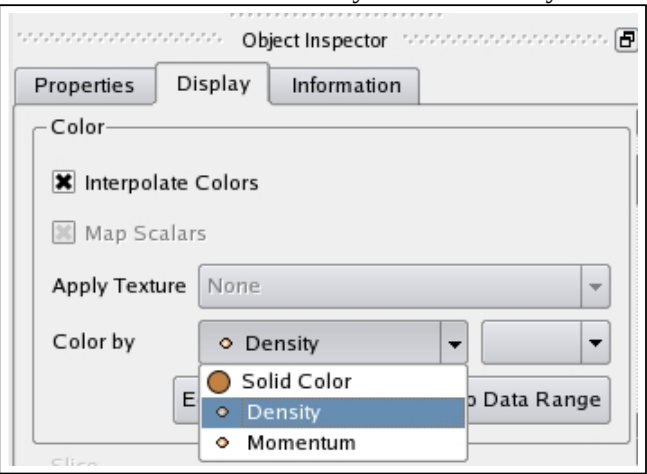
There are two options available for coloring a dataset. To use the same color over an entire dataset, select Solid Color from the “Color by” menu.



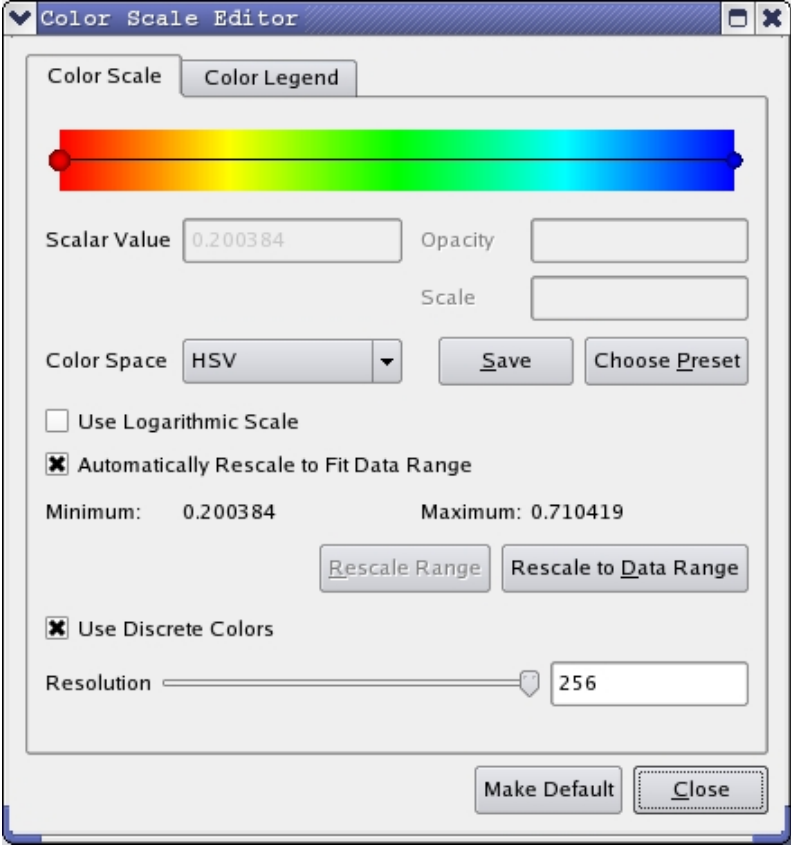
Clicking the **Set Solid Color** button below the “Color by” menu displays a color chooser window from which a particular color may be selected.



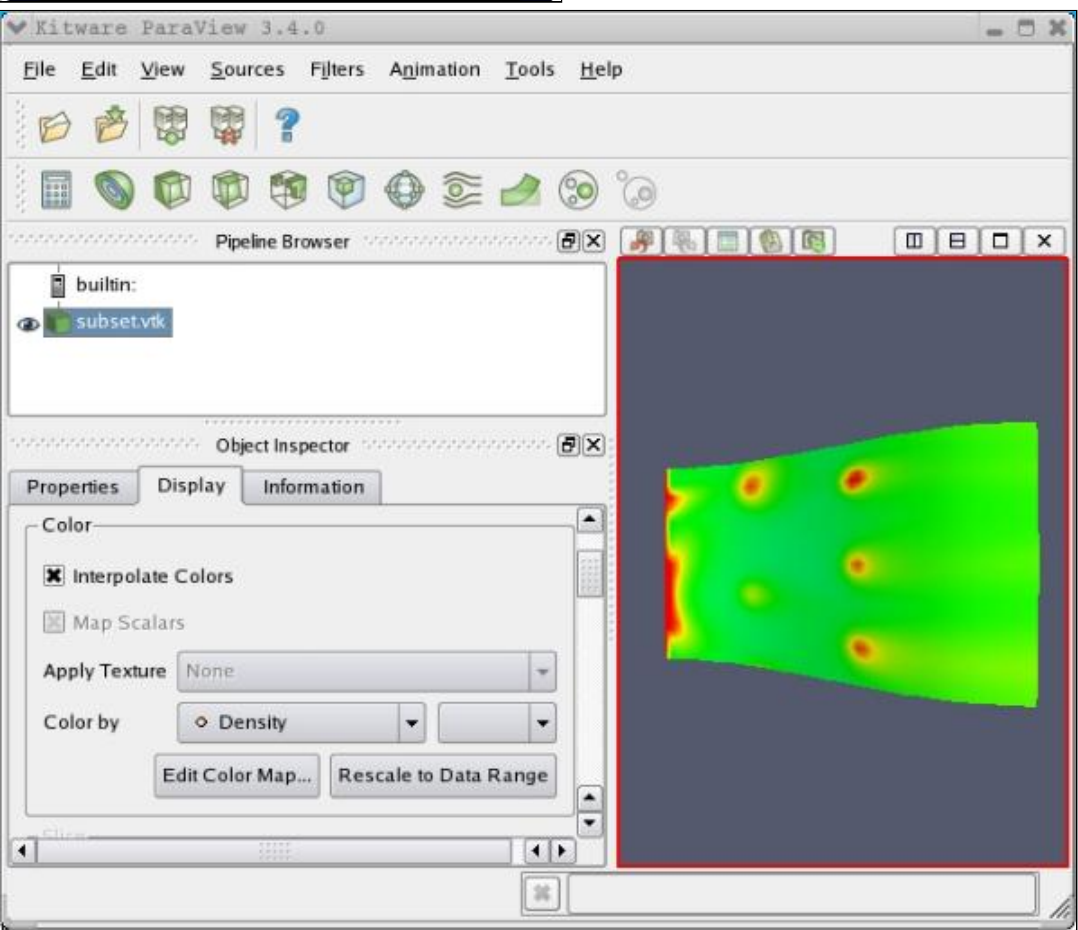
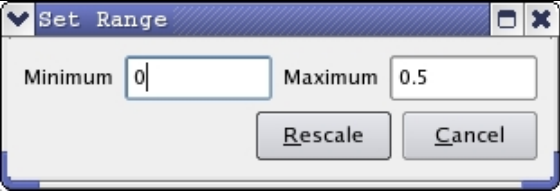
If the data set contains data attributes such as scalars or vectors, then the “Color by” menu also lists the names of the attribute arrays. Selecting an array name causes the dataset to be colored based on the underlying scalar values in that array. If you select a vector array, a scalar array will need to be created for the mapping. This can be accomplished by selecting Magnitude in the second “Color by” menu, which will generate a scalar array of vector magnitudes. In the **subset.vtk** data set, which is loaded in the current example state file, there are two attributes: a scalar array named “Density” and a vector array named “Momentum.”



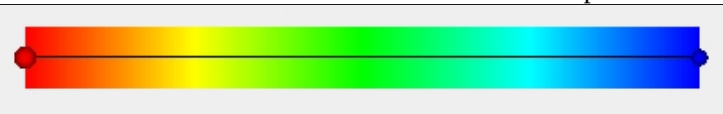
The mapping from data values to colors is determined by the color map. The color map may be edited in the Color Scale Editor window which appears when you click the **Edit Color Map** button in the Color section of the Display Tab..



One way to change the mapping of data values to colors is by setting the Data Range. By default the Data Range is set from the minimum data value in the data set to the maximum data value. By Clicking on the **Rescale Range** button (after first deselecting the **Automatically Rescale to Fit Data Range** toggle) you can explicitly set these values. The values between the minimum and maximum are then linearly interpolated into the color table. Here is an example of setting the Minimum value to 0.0 and the Maximum value to 0.5:



Another way of affecting the mapping of data values to colors is to change the color ramp in the color table. At the top of the Color Scale Editor window is a color map editor:



The color map editor displays the mapping between scalar values and colors, and provides the means for making modifications to this mapping. The nodes in the color map editor (shown as colored spheres) indicate the mapping from a scalar value to a color. The colors are interpolated linearly between the nodes using the color space chosen in the Color Space menu below the color map editor. More nodes may be added simply by left-clicking within the editor.



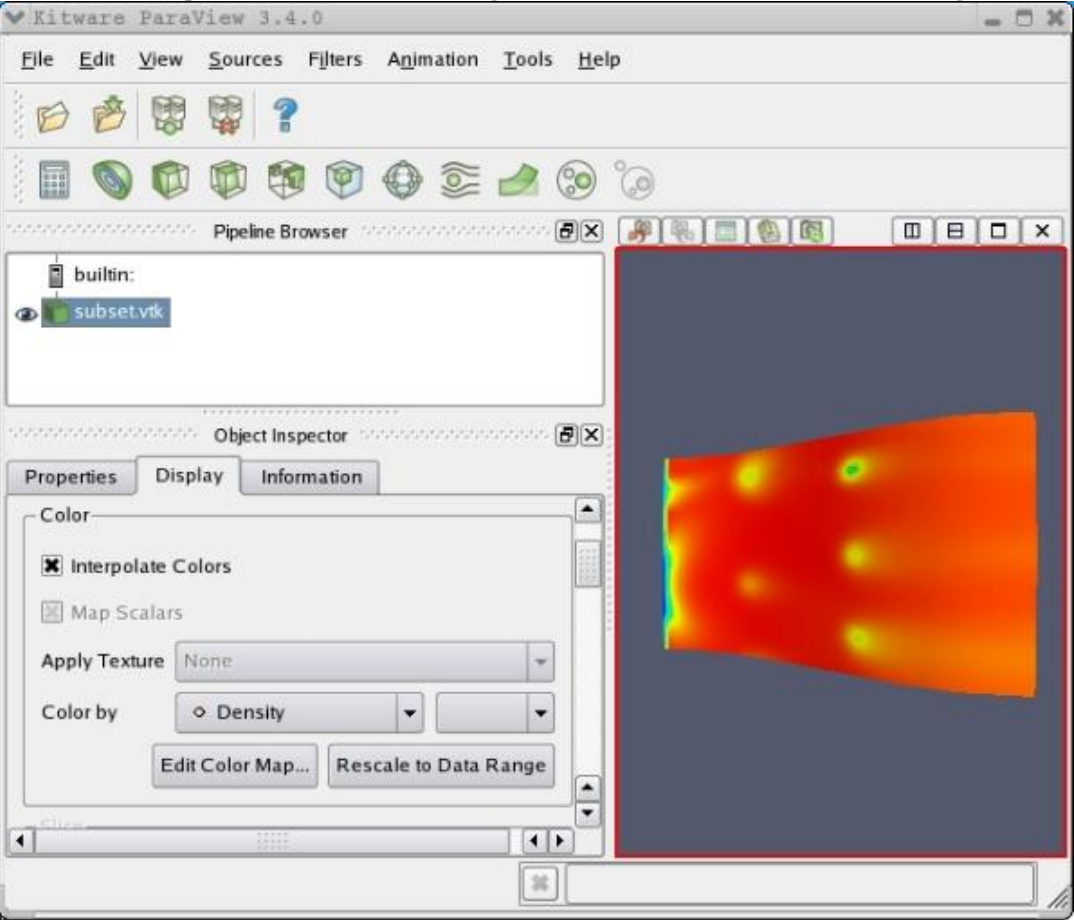
A node in the color map editor becomes selected when it is left-clicked. The selected node is indicated as a larger sphere in the editor. Pressing either the 'd' or Delete key removes the selected node from the editor. Clicking the currently selected node or double-clicking any node causes a color chooser to be displayed, from which a new color may be selected for the node. The scalar value of the selected node may be interactively changed by dragging the node either left (decreasing scalar value) or right (increasing scalar value). The node's color remains the same during this process.



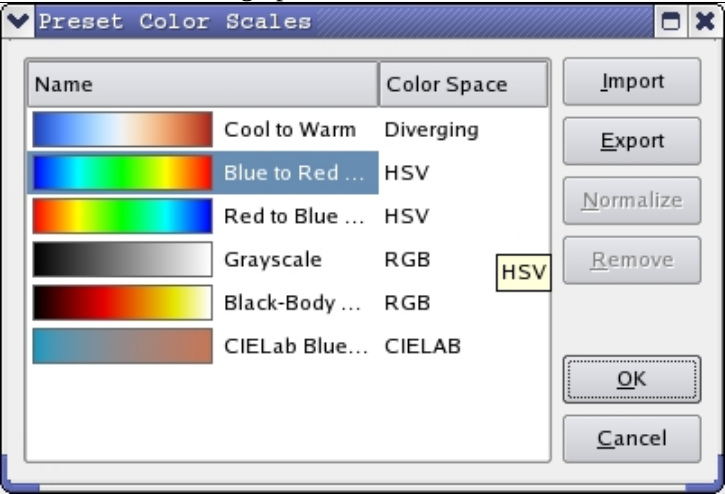
The scalar value for the selected node may also be changed manually by entering a new value in the Scalar Value entry box.



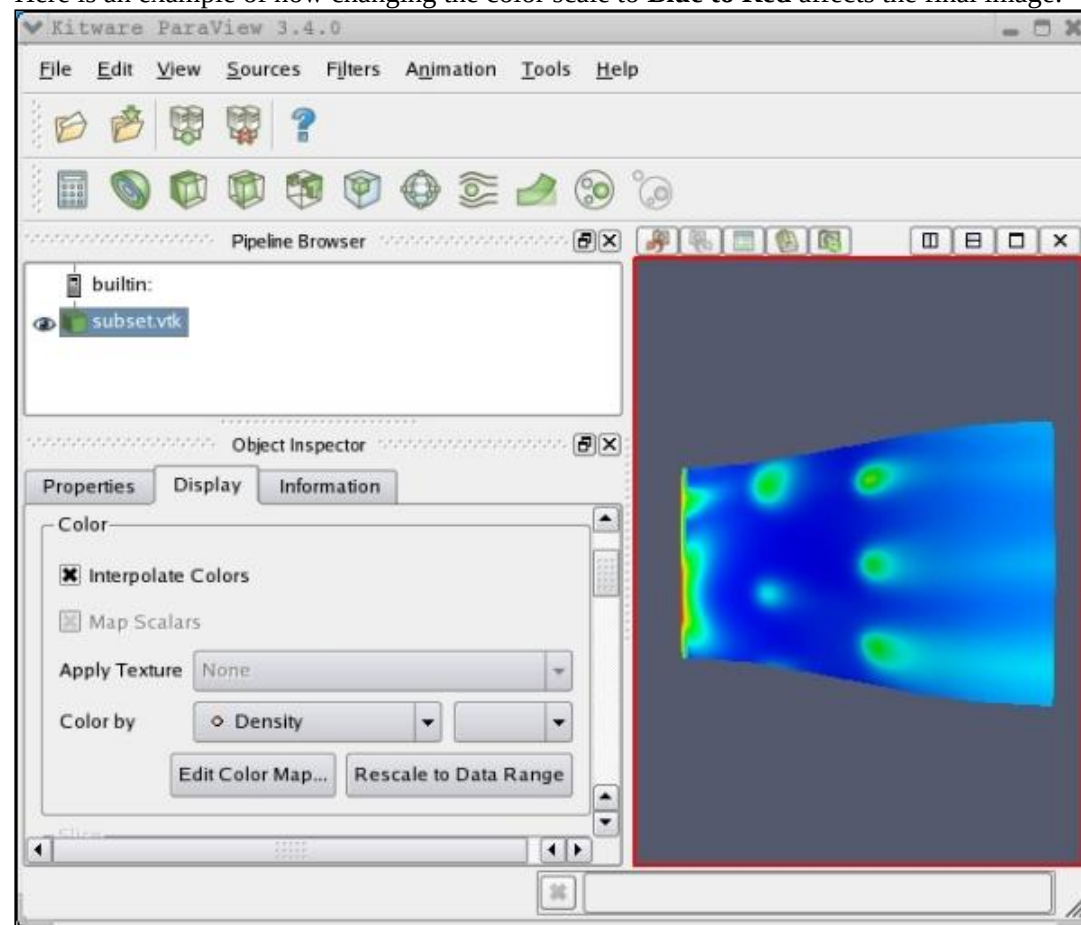
Here is an example of how the above changes to the color table affect the final image:



Several preset Color Maps are also available for your use and can be selected by clicking on the **Choose Preset** button in the Color Scale Editor. This will bring up the **Preset Color Scales** window.



Here is an example of how changing the color scale to **Blue to Red** affects the final image:



Modifications made to the color map may also be saved for future use by clicking the **Save** button below the color map editor.

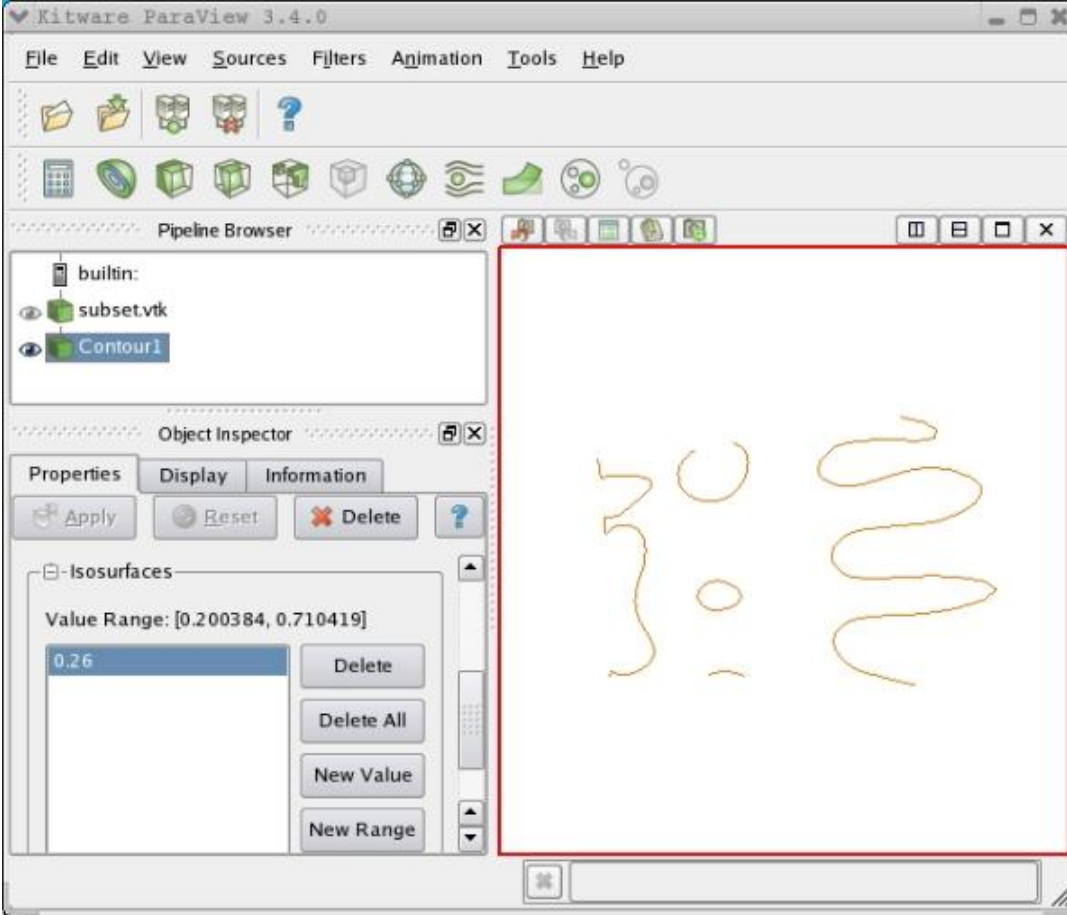
Several additional examples of using color mapping to visualize scalar data are located in the `materials/Demos/ColorMaps` directory.

Contours / Isosurfaces

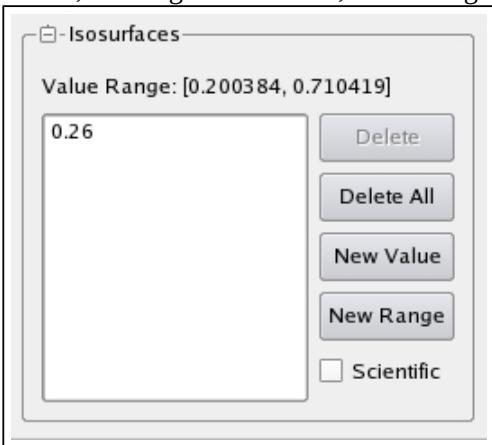
Contouring is a technique where one constructs a boundary between distinct regions in the data. Contours are lines or surfaces of constant scalar value. This is a natural extension from color mapping as our eyes instinctively separate similarly colored areas into distinct regions. The first step in contouring is to explore the data space to find points near a contour or region of interest. Once found these points are then connected into either contour lines (**isolines**) for two-dimensional (2D) data or into surfaces (**isosurfaces**) for three-dimensional (3D) data. The lines or surfaces can be color mapped using the scalar data. Several examples of using color mapping to visualize scalar data are located in the `materials/Demos/Contours` directory.

To start load the `contour.single.pvsm` statefile. The statefile is located in the `materials/Demos/Contours/` directory.

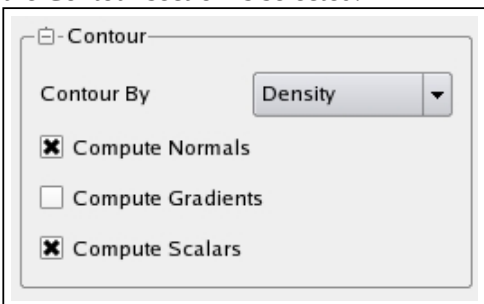
```
sccl% cd materials/Demos/Contours
sccl% paraview --state=contour.single.pvsm
```



The primary ParaView filter used for contouring is the [Contour](#) filter. To apply a Contour filter you would first select the object to which you want to apply the filter and then choose the Contour filter from the Filters menu. In the current example, we want to see contours in the scalar attribute data of the 2D data set. We first select **subset.vtk** which is the data reader for our 2D data set, and then we add a contour filter which is called **Contour1**. In the object inspector under the Properties Tab for the Contour filter, we can set the scalar value or values that will be used to generate the contours. In the Isosurfaces section of the Properties Tab, there are buttons for adding scalar values, deleting scalar values, and setting up a range of values which will be used to generate the contour lines.

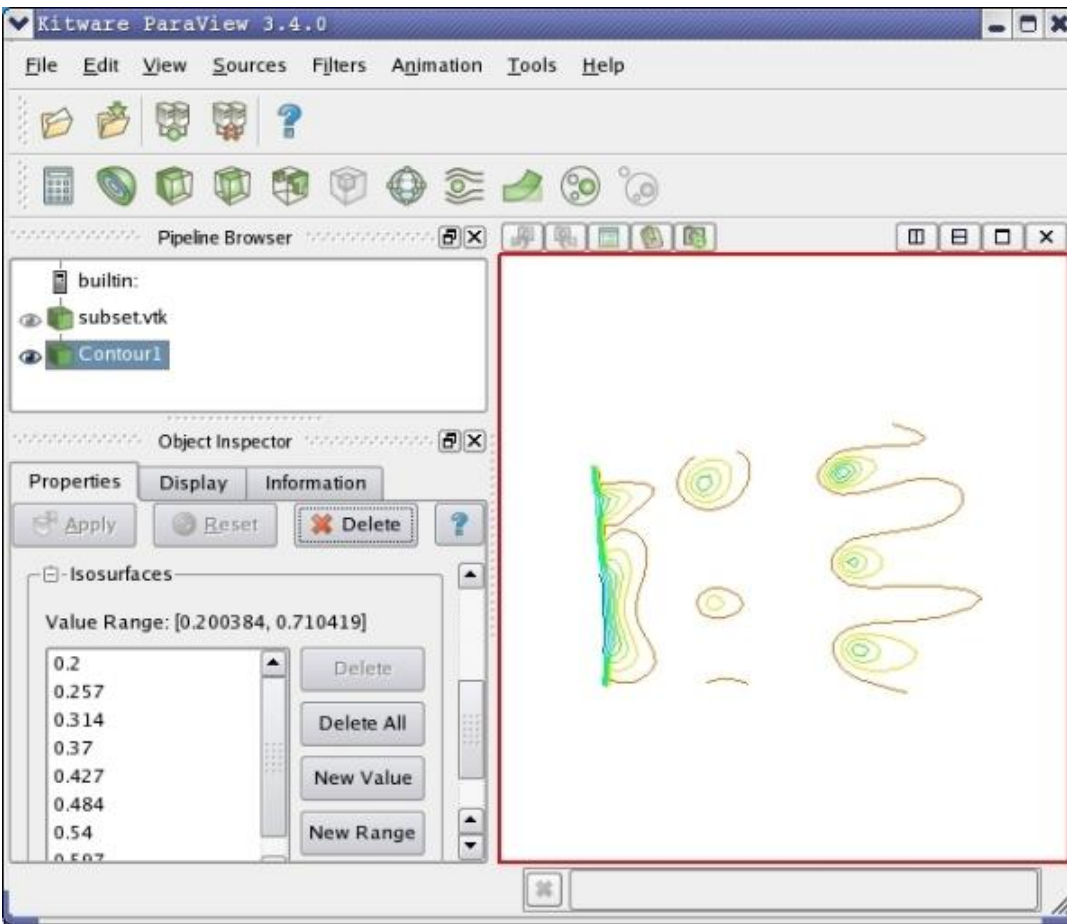


If you wish to color the contour line based upon its scalar value and the current color map, make sure the **Compute Scalars** checkbox in the Contour section is selected.



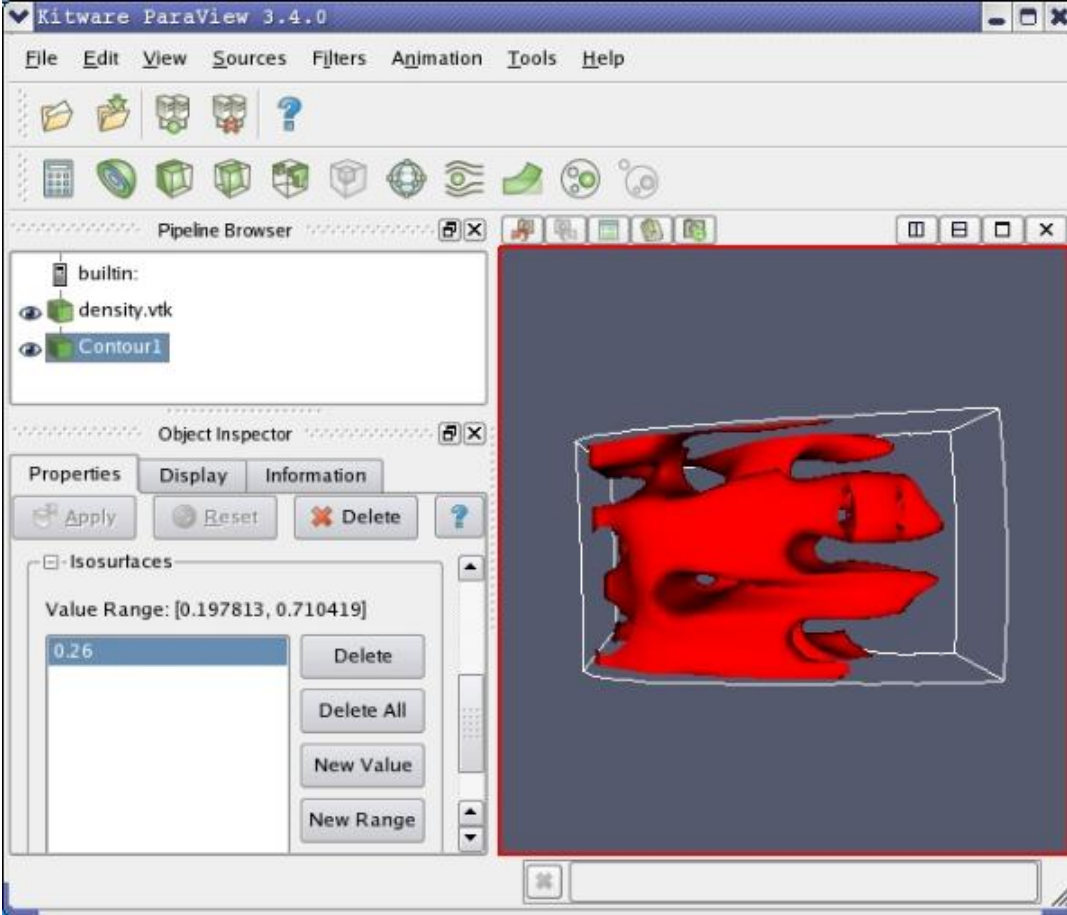
The `contour.multi.pvsm` statefile shows an example of multiple colored contour lines.

```
sccl% cd materials/Demos/Contours
sccl% paraview --state=contour.multi.pvsm
```



For three-dimensional data instead of generating contours or isolines we generate isosurfaces. We use exactly the same technique as used for generating contours in two-dimensional data. The only difference is we are reading in a three-dimensional data set (**density.vtk**) instead of a two-dimensional data set (**subset.vtk**). Here is an example:

```
sccl% cd materials/Demos/Contours
sccl% paraview --state=isosurface.pvsm
```



More examples of creating isosurfaces and contours are located in the `materials/Demos/Contours` directory. `isosurface.colorbyVector.pvsm` and `isosurface.colorbyScalar2.pvsm` show you how to color the isosurface using additional vector or scalar data.

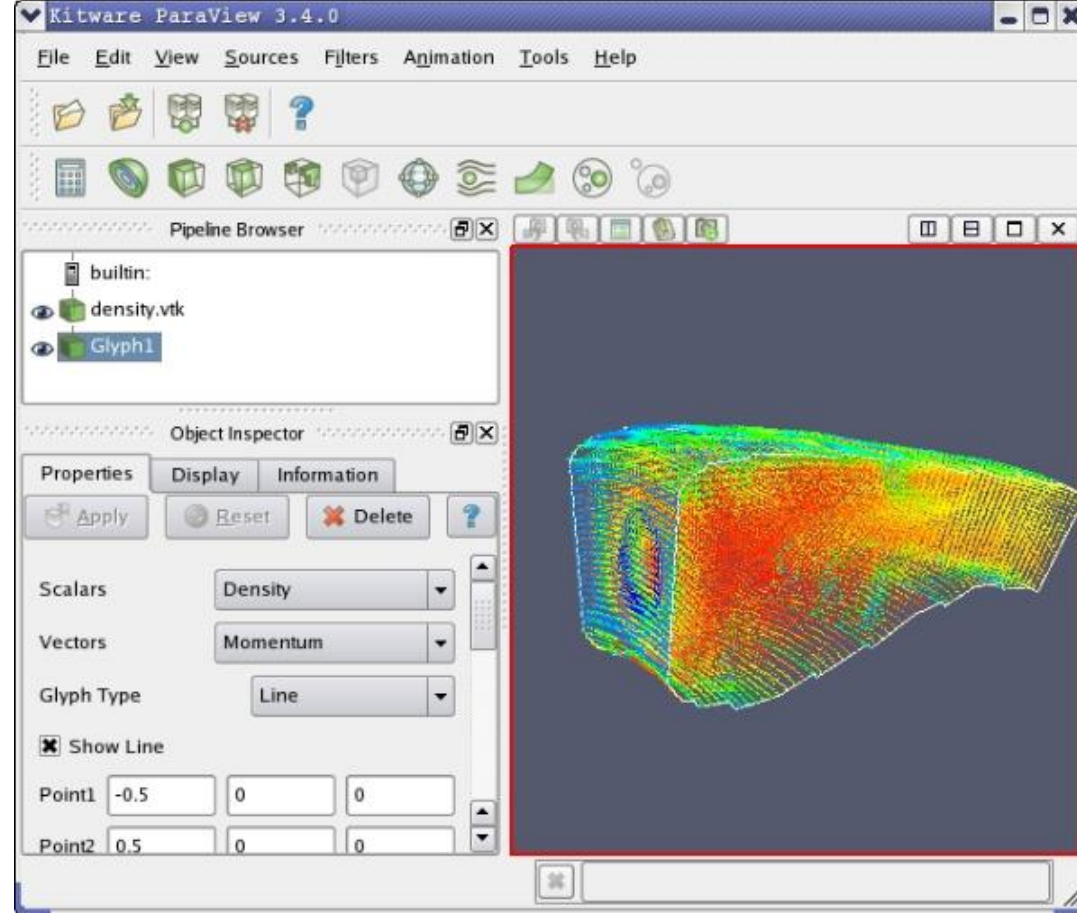
Vector Visualization Algorithms

Vector data is a three-dimensional representation of direction and magnitude associated with each point/cell in the data set. Vector data is often used to describe the rate of change of some quantity. For example vectors can be used to describe fluid flow. There are several algorithms that can be used to visualize vector data.

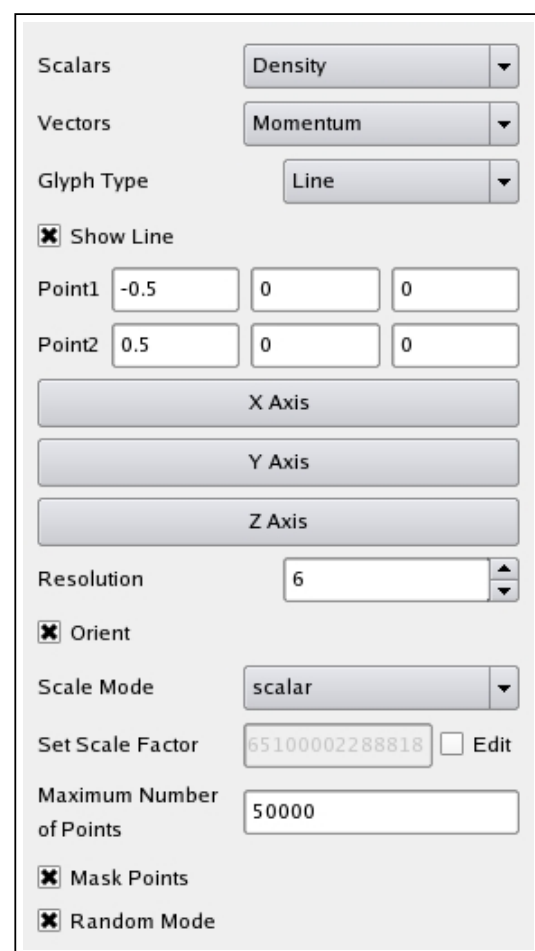
Hedgehogs

One visualization technique used for vector data is to draw an oriented, scaled line for each vector. The lines may be colored according to vector magnitude or some other scalar quantity (e.g. temperature or pressure). The result looks similar to a bristly Hedgehog. Often you will need to adjust the scaling of the lines to control the size of its visual representation. To start load the `hedgehog.pvsm` statefile which is located in the `materials/Demos/Hedgehogs/` directory.

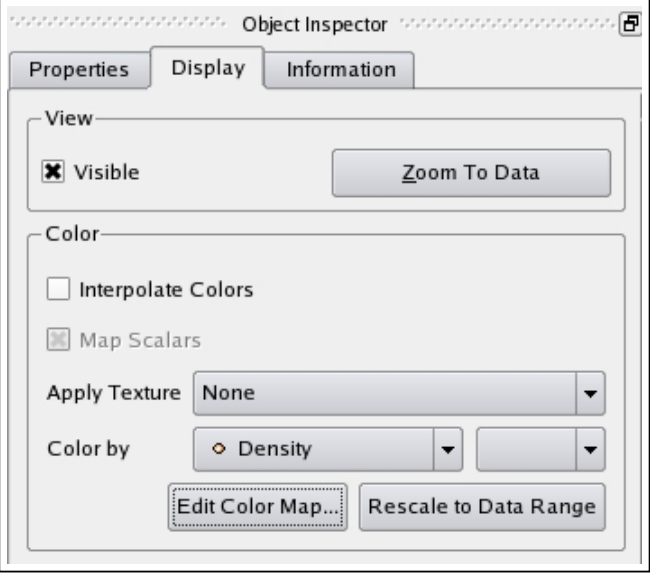
```
sccl% cd materials/Demos/Hedgehogs
sccl% paraview --state=hedgehog.pvsm
```

The primary ParaView filter used for creating a Hedgehog is the [Glyph](#) filter. In the current example, we see oriented, scaled, colored lines for each of the Momentum vectors in the data set. In the Object Inspector under the Properties Tab for the Glyph filter, we set the “Scalars” menu to **Density**, set the “Vectors” menu to **Momentum**, set the “Glyph Type” to **Line**, checked “Orient”, set the “Scale Mode” to **scalar**, and set the “Maximum Number of Points” to **50000**.



The lines in the Hedgehog are colored by the density scalar attributes of the data set. This is done by setting the “Color by” menu to **Density** in the Color section of the Display Tab of the Glyph filter.

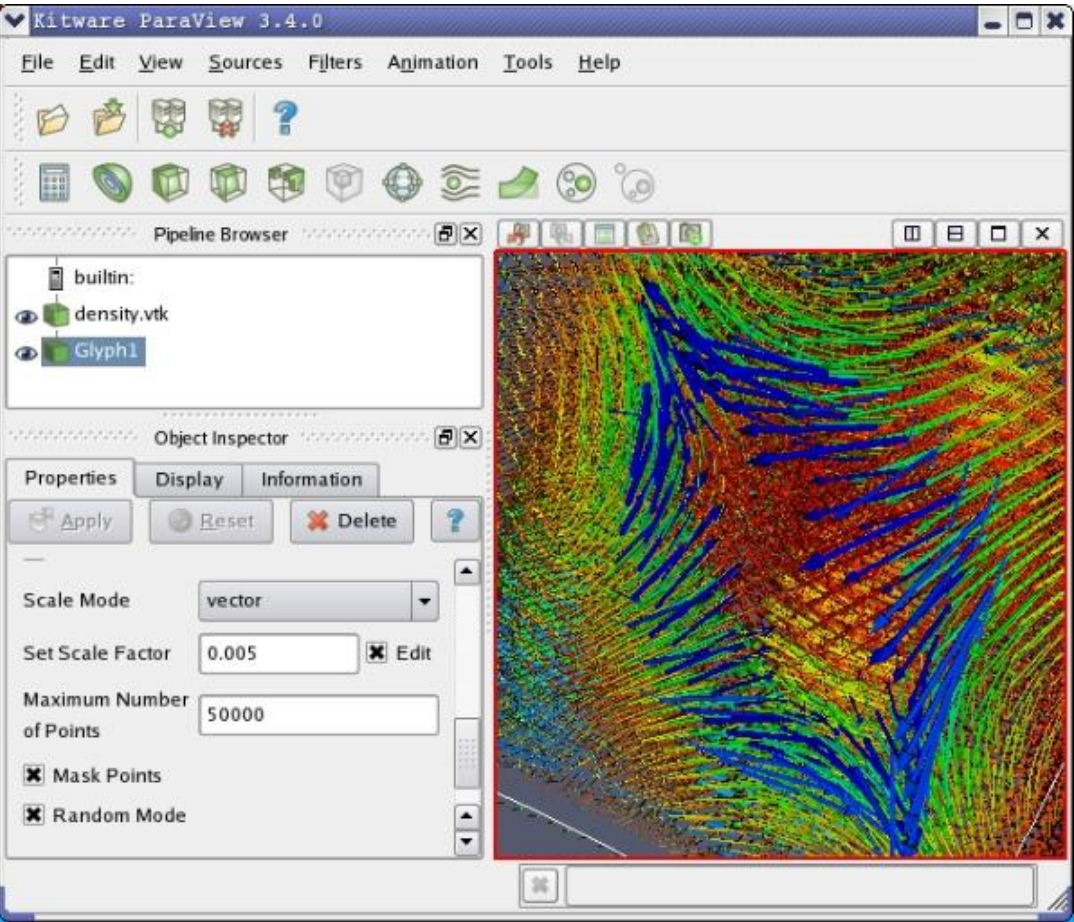


Oriented Glyphs

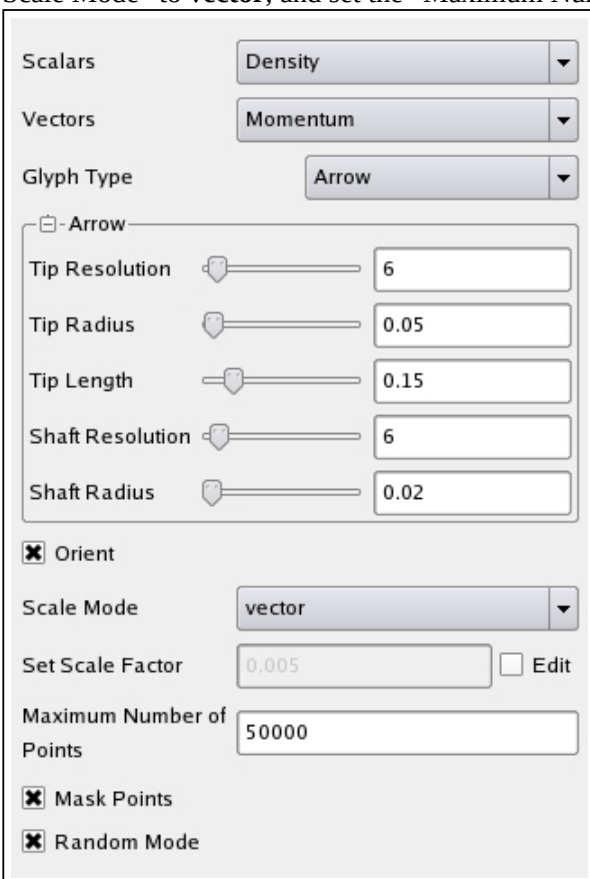
A similar technique to using Hedgehogs to visualize vector data is to use Oriented Glyphs. Glyphs are polygonal objects such as a cone or an arrow and can be used in place of the lines of the Hedgehog. Glyphs like Hedgehogs can be colored, scaled, and oriented. The orientation and scale of the glyphs are often used to indicate the direction and magnitude of the vectors. Scalar attributes, such as temperature or density, are used to color the glyphs.

To start load the `glyph.scale.pvsm` statefile which is located in the `materials/Demos/Glyphs/` directory.

```
sccl% cd materials/Demos/Glyphs
sccl% paraview --state=glyph.scale.pvsm
```



The primary ParaView filter used for creating Oriented Glyphs is the [Glyph](#) filter. The main difference from the Hedgehog setup is that instead of a **line** we use an **arrow**, **cone**, **box**, **cylinder**, or **sphere** for the glyph type. In the current example, we see oriented, scaled, colored arrows for each of the Momentum vectors in the data set. In the Object Inspector under the Properties Tab for the Glyph filter, we set the “Scalars” menu to **Density**, set the “Vectors” menu to **Momentum**, set the “Glyph Type” to **Arrow**, checked “Orient”, “ set the Scale Mode” to **vector**, and set the “Maximum Number of Points” to **50000**.

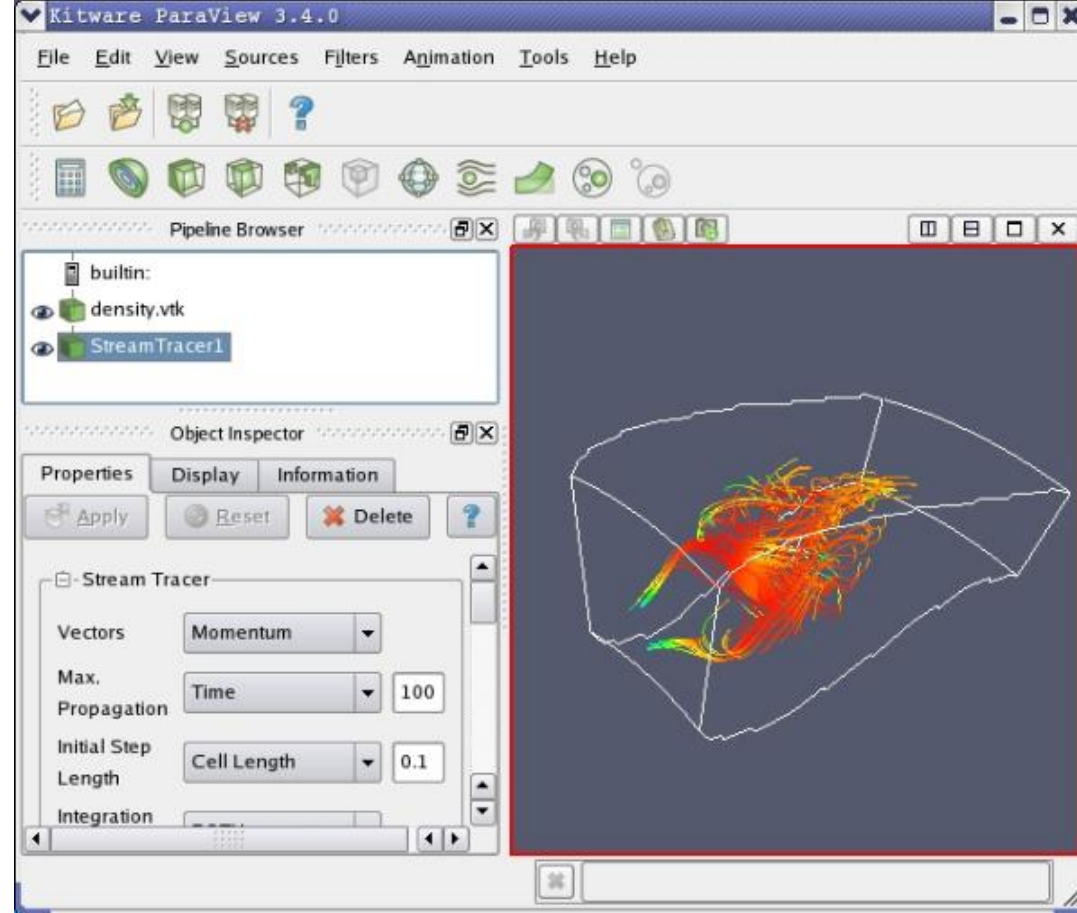


Streamlines

A streamline can be thought of as the path a massless particle takes flowing through a velocity field (i.e. vector field). Streamlines can be used to convey the structure of a vector field by providing a snapshot of the flow at a given instant in time. Multiple streamlines can be created to explore interesting features in the field. Streamlines are computed via numerical integration (integrating the product of velocity times delta T).

To start load the `streamLines.pvsm` statefile which is located in the `materials/Demos/Streamlines/` directory.

```
sccl% cd materials/Demos/Streamlines
sccl% paraview --state=streamLines.pvsm
```



The primary ParaView filter used for creating Streamlines is the [StreamTracer](#) filter. The Stream Tracer filter generates streamlines in the vector field from a collection of seed points. Production of streamlines stops if a streamline crosses the exterior boundary of the input data set. When configuring the Stream Tracer, you will first need to set up the integrator to do the numerical integration. Next you will need to specify the seeds points. In the current example under the Properties Tab for the StreamTracer filter (in the Stream Tracer Section), we set the “Vectors” menu to **Momentum**, set the “Max. Propagation” to **Time** with a value of **100**, set the “Initial Step Length” to **Cell Length** with a value of **0.1**, set “Integration Direction” to **Both** with a Maximum number of steps set to **1000**, and set the “Integrator Type” to **Runge-Kutta 4**. To set up the seed points, we chose a **Point Source** for “Seed Type,” **Center on Bounds**, with **100** points and a “Radius”

of 3 in the Seeds Section of the Property Tab.

Stream Tracer

Vectors

Momentum

Max. Propagation

Time

100

Initial Step Length

Cell Length

0.1

Integration Direction

BOTH

Max. Steps

1000

Term. Speed

1e-12

Integrator Type

Runge-Kutta 4

Minimum Step Length

Cell Length

0.01

Maximum Step Length

Cell Length

0.01

Maximum Error

1e-06

Seeds

Seed Type

Point Source

☒ Show Point

Center on Bounds

Point

8.255000114

0

29.76309967

Number of Points

100

Radius

3

There are more examples of creating Streamlines in the materials/Demos/Streamlines directory. `streamRibbon.pvsm` shows you how to create ribbons instead of lines and `streamTubes.varyRadius.pvsm` show you how to create tubes with varying radii.

Modeling Visualization Algorithms

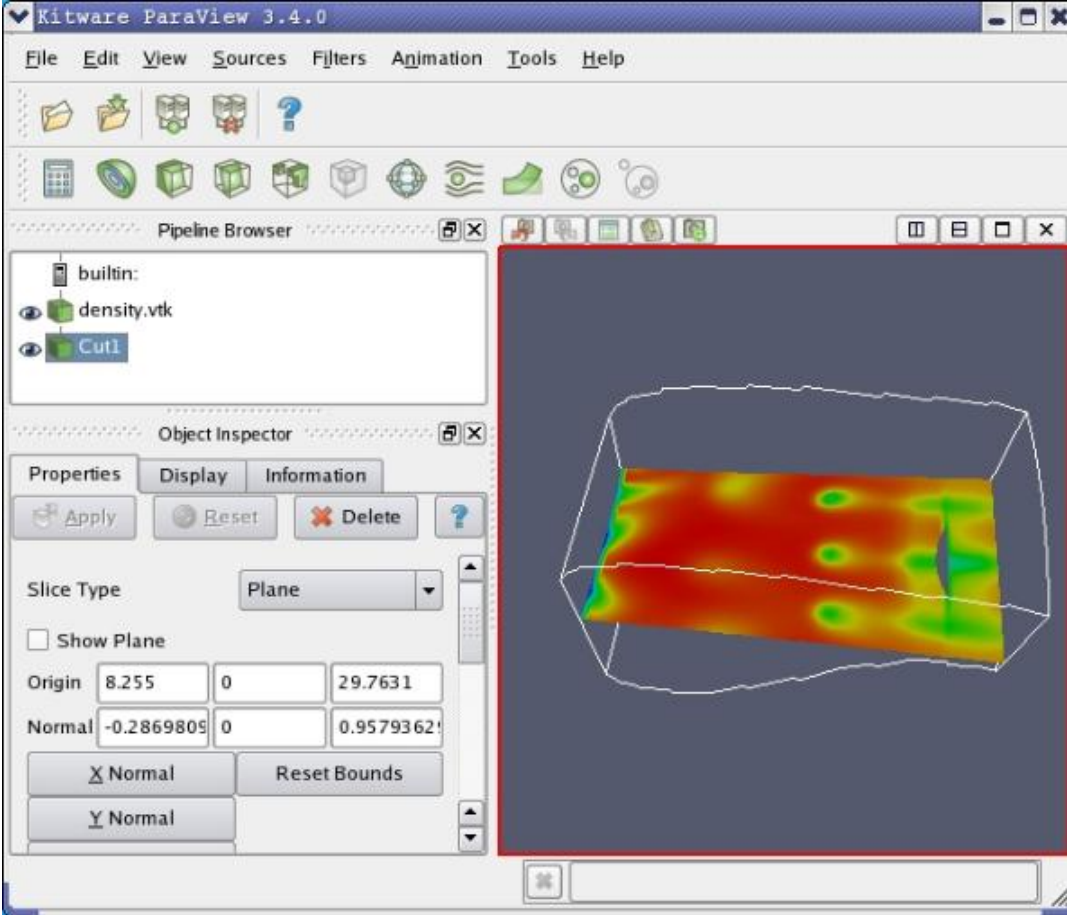
Modeling algorithms change data set geometry or topology. They are often used to reveal internal details of a data set. Two common modeling visualization techniques are cutting (slicing) and clipping.

Cutting/Slicing

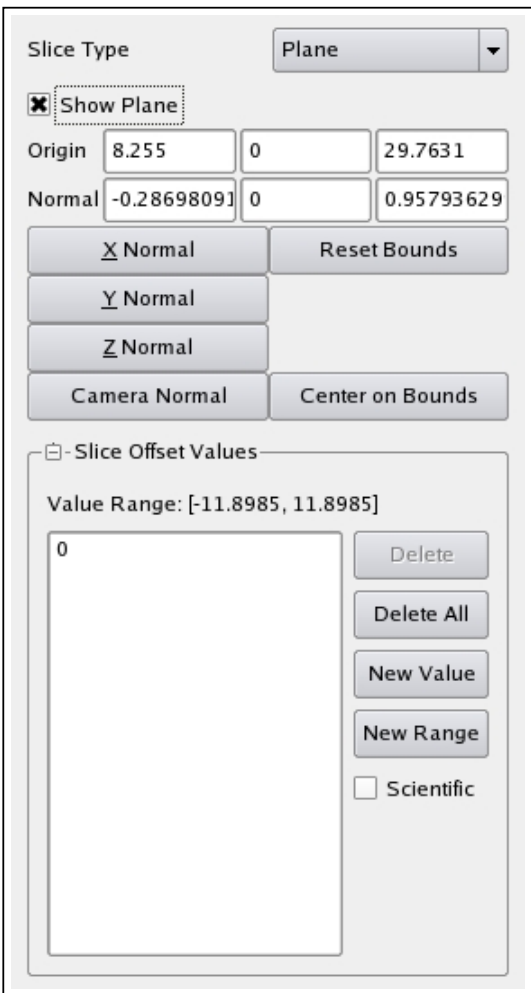
Cutting also called slicing entails creating a “cross-section” of the data set. Cutting uses an implicit function to define a surface with which to cut the data set. Any type of implicit function may be used to create the surface. One simple technique is to use a plane to define the cutting surface thereby creating a planar cut. The cutting surface interpolates the data as it cuts and can be visualized using any of the standard visualization techniques.

To start load the `cutplane.pvsm` statefile which is located in the materials/Demos/Cutplanes/ directory.

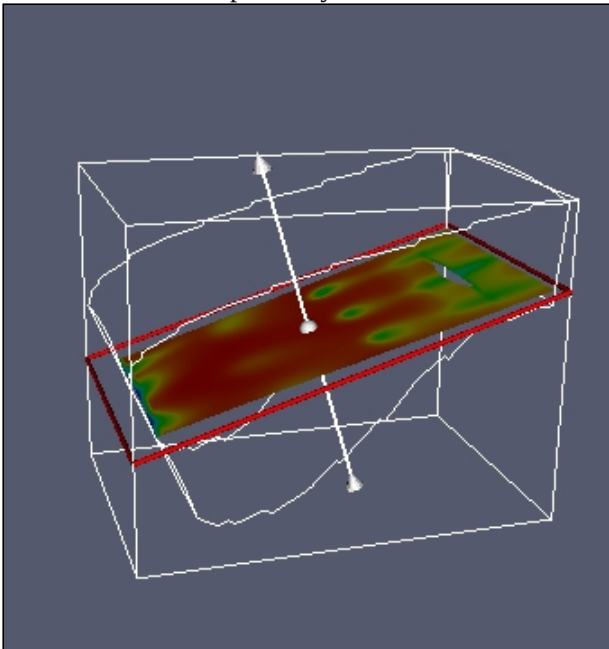
```
sccl% cd materials/Demos/Cutplanes
sccl% paraview --state=cutplane.pvsm
```



The primary ParaView filter used for creating a cut or slice is the [Cut](#) filter. The Cut filter slices the data set with a plane. When configuring the Cut filter you will need to specify the plane that is used to slice the data set. The easiest way to do this is to turn on “Show Plane” in the Properties Tab for the Cut filter.



ParaView will then present you with a 3D Plane Widget in the View window which you can then use to define the plane.



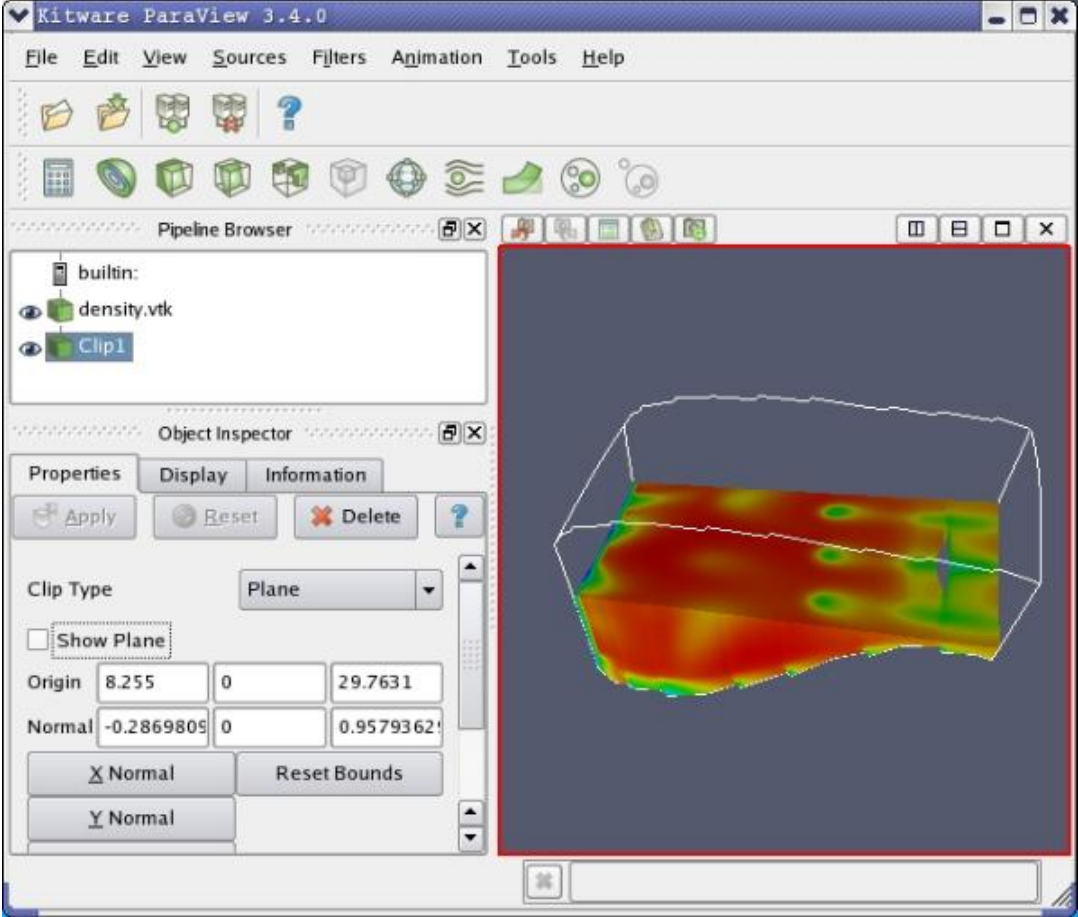
After defining the plane we used Color Mapping to color the plane with the density scalar attributes of the data set.

Clipping

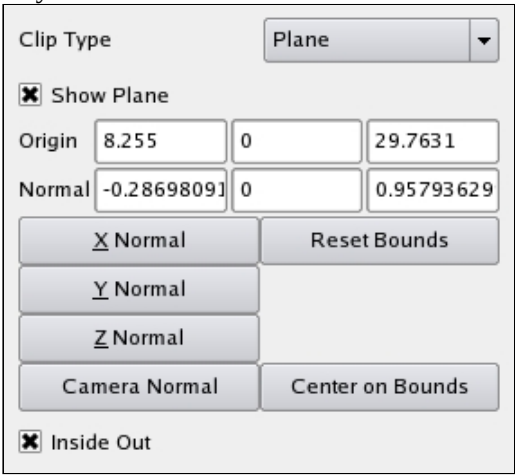
Clipping, like cutting, also uses an implicit function to define a surface. But instead of creating a cross-section of the data set, clipping uses the surface to “cut” through the cells of the data set, returning everything inside of the specified implicit function. The result of clipping is an unstructured grid. As we did with cutting in the previous example, to keep things simple we will also use a plane as our implicit function to create the surface for clipping the data set.

To start load the `clipping.pvsm` statefile which is located in the `materials/Demos/Clipping/` directory.

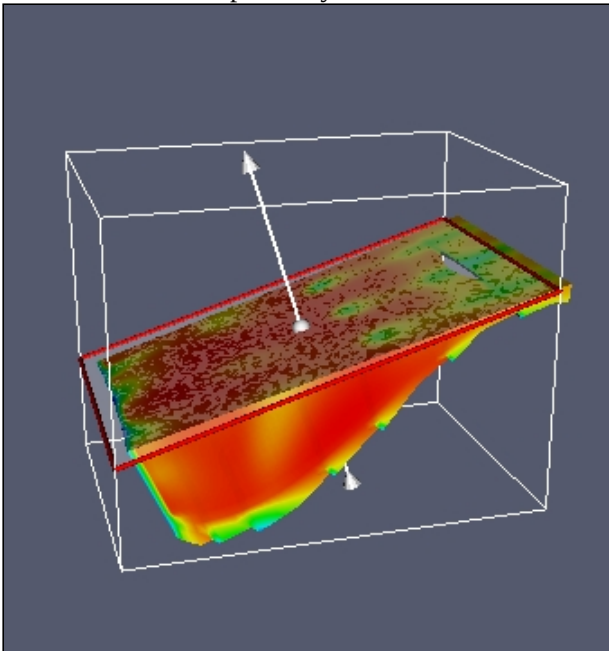
```
sccl% cd materials/Demos/Clipping
sccl% paraview --state=clipping.pvsm
```



The primary ParaView filter used for clipping is the [Clip](#) filter. The Clip filter cuts away a portion of the input data set using an implicit plane. When configuring the Clip filter, you will need to specify the plane that is used to clip the data set. Like the Cut filter the easiest way to do this is to turn on “Show Plane” in the Properties Tab for the Clip filter.



ParaView will then present you with a 3D Plane Widget in the View window which you can then use to define the plane.



After defining the clipping plane, we used Color Mapping to color the resulting unstructured grid with the density scalar attributes of the data set.

Additional Help

For more information on ParaView, visit either the [ParaView website](#) or the [ParaView Wiki](#). A [ParaView FAQ](#) is available online, along with an [overview](#) of the package. If you are just starting to learn ParaView, we recommend that you get the [ParaView Guide](#) which is available from Kitware.

For specifics on running ParaView on Research Computing systems see our [ParaView Help Page](#).

As part of our 2008 [Workshop on Scientific Visualization](#), a talk on [VTK and ParaView](#) was presented which you might find useful. The visualizations which were used during the workshop are available for download on the [ParaView Examples](#) web page.

An updated talk on [Scientific Visualization using Paraview](#) was presented in February, 2010. [Demos](#) and [data](#) used in the tutorial are also available.

And finally there are several excellent ParaView tutorials available externally:

1. [Sandia National Laboratories ParaView 3 Tutorials](#)
2. [SC08 ParaView Tutorial](#)
3. [IEEE Vis08 ParaView Tutorial](#)

References

The Visualization Toolkit, 3rd Edition, Will Schroeder, Pearson Education, Inc, 2002.

The ParaView Guide, Amy Henderson Squillacote, Kitware, 2007.

Kitware: www.paraview.org

ParaView online help files


- [Image Files](#)
- [Introduction to Scientific Visualization Tutorial](#)
- [IDL Tutorial](#)
- [MPI Tutorial](#)
- [Introduction to MATLAB](#)
- [Using MATLAB to Visualize Scientific Data](#)
- [MATLAB Parallel Computing Toolbox Tutorial](#)
- [Using VTK to Visualize Scientific Data](#)

- [Using ParaView to Visualize Scientific Data](#)
- [Plotting Packages Tutorial](#)
- [Archive](#)

Contact Us

- ☎617-353-HELP (4357)
- ☎ithelp@bu.edu
- •[Contact Us](#)
- •[Feedback](#)
- •[Privacy Statement](#)

- 
[RSS](#)
[ATOM](#)

- 
[@buinfosec](#)
[@buihelp](#)
[@bumcit](#)
[@edtechbu](#)

- 