Rocketeer Users Guide
Version 1.3.6

March 12, 2014

www.illinoisrocstar.com

License

Rocketeer sources, executables, and this document are the property of Illinois Rocstar LLC. Licensing and support of the software package, including full source access for government, industrial, and academic partners, are arranged on an individual basis. Please contact Illinois Rocstar at

- **tech@illinoisrocstar.com**
- **sales@illinoisrocstar.com**

for support and licensing.

Contents

1 Introduction

The Rocketeer Suite consists of three powerful tools for visualizing 3-D scientific data sets. They were developed by John Norris and Robert Fiedler at CSAR to analyze numerical results from rocket simulations, but they can be used for viewing many types of 2-D and 3-D data. The Rocketeer Suite includes the original serial interactive application (still called *Rocketeer*), an MPI parallel batch mode version called *Voyager* for processing a series of output files from different simulation times, and a client/server implementation (with an MPI parallel server) called *Apollo/Houston* for viewing data on remote systems. These three tools share the same code base, offer almost the same set of features, and this User's Guide applies to all three.

The *Rocketeer* suite is written in C++ and uses the *Visualization Toolkit*, which is based on *OpenGL* for portable graphics acceleration. The GUI uses wxWindows for portability across multiple platforms, including UNIX, Windows, and others. *Rocketeer* is currently known to work on Mac OS X, Linux, Solaris, AIX, and Microsoft Windows.

Rocketeer handles many different types of grids on which data is defined, as well data data defined at a set of points. The grid may be non-uniform, structured, or unstructured, and multiblock. *Rocketeer* can display multiple data sets for multiple materials from multiple files on a single image. It can perform the same set of graphics operations automatically on a series of output dumps to produce frames for animation.

Rocketeer can show the grid on the surface of a computational domain, and/or it can indicate the value of a scalar variable on that surface using a color scale. It can also display scalar variables as multiple isosurfaces and/or on slices across the x, y, and/or z axes. The surface, isosurfaces, and slices can be made translucent and/or they can be cut away at various planes to allow a clearer view of the interior of a 3-D volume. The magnitude of vector variables can be displayed in the same way as scalars, and the vectors themselves can be represented by — cone plots, with oriented cones of variable or fixed size whose color can be used to indicate magnitudes. Tensor variables are also supported and can be represented as ellipsoids shaped and oriented according to the principle axes.

Rocketeer can display 3-D meshes (volumetric or surface) in a variety of ways. The entire volumetric mesh can be displayed, or it can be shown only in the vicinity of points specified by coordinates, grid indices, or a bounding box. Point data can be displayed as glyphs, e.g., spheres for scalars. All these graphical objects can use color scales and translucency to convey additional information.

Rocketeer reads data stored in HDF format (version 4) or CGNS. Attributes (metadata) stored with the data enable *Rocketeer* to determine the time levels, block numbers, grid types, material types, and variable types (scalar/vector/tensor) without additional user input.

2 Release Notes

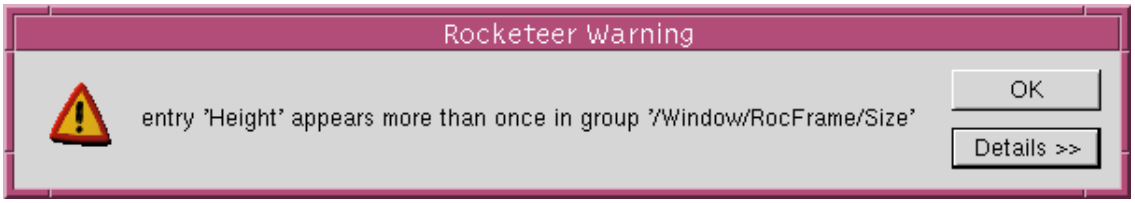
New features for v1.3.6:

- Mesh quality metrics can be used as threshold variables
- Memory usage has been minimized
- Supports Mac OS X
- Supports *Rocstar*3 output dumps
- Ignores blocks with bad data ranges critical for visualizing particles in muliphysics fluid simualtions
- Supports CGNS format data files — can even read ghost cell data from unstructured meshes; we are working with the CGNS developers to extend their standard so that other tools build on CGNS can also read our Rocstar 3 CGNS files.
- Uses the GODIVA interfaces for reading data files (enables data set prefetcing and caching to reduce input latency)
- Includes mesh quality measures (built-in algebraic and user writeable); the ranges are now calculated automatically
- Can display mesh blocks in different colors
- Can display mesh data even without any field variable data
- Vector magnitudes can be used for thresholds

3 Default Settings and Custom Color Tables

When Rocketeer or Apollo is first run on a Unix system, a directory called `.CSAR_Vis` is created in the user's home directory. This directory contains subdirectory "rc" for "rocketeer" and "apollo" default settings files, subdirectory "color_tables" for user-generated color maps, and subdirectory "readers" meant for user-written reader plug-ins.

If you are upgrading from *Rocketeer* v1.3, when v1.3.6 starts up it may complain:



To avoid this warning, remove the "rocketeer" and/or "apollo" settings files in your `.CSAR_Vis/rc director`
Rocketeer will write new files to record your image size preference in subsequent sessions.

Under MS Windows, the default settings are stored in the registry.

Users can provide their own color maps (also known as color tables or color bars) under both Unix and MS Windows to customize the display of data. If *Rocketeer* is installed in directory $INSTALLDIR, you will find 8 color tables in $INSTALLDIR/colorbars:

```
00rainbow
10inverted_rainbow
20starlight
30carnation
40hot_metal
50purple_haze
60rose
99grayscale
```

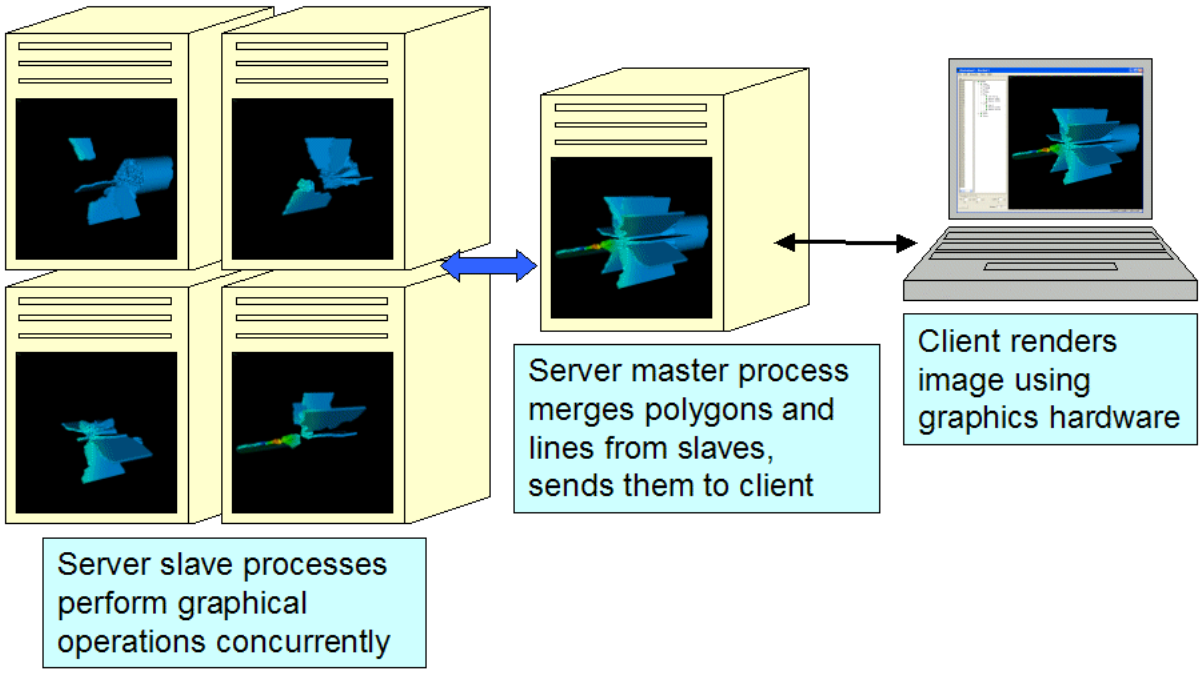
The user color tables (to be placed in $HOME/.CSAR_Vis/colorbars under Unix, but in $INSTALLDIR/colorbars under MS Windows) can take any unused numbers from 0 to 99. They have the same form as the color table file 00rainbow:

Rainbow

0	0.5	0	1
7168	0	1	1
21504	0	1	1
22528	0	1	1
36864	0	1	0
51200	1	1	0
65536	1	0	0

The color table includes a colorbar name and 65537 rows (65536 plus one extra so that the highest level is not black) specifying the intensity of red, green, and blue on a scale from 0 to 1. Linear interpolation is performed to obtain intensity values for any missing rows.

4 Running the Client/Server Version



The client (Apollo) runs on your desktop workstation (Sun, linux, AIX, Mac OS X, or Windows), while the server (Houston) runs on Solaris, linux, Mac OS X, or AIX. The data that you will be able to visualize resides on the server, but the images you create will be saved on your desktop, since it is there that they are rendered.

Houston is a MPMD parallel application, with one master process (HoustonMaster) and any number of slave processes (HoustonSlave). The master process communicates with the client using TCP/IP. As it is currently implemented, each slave must have at least one data file to read (per snapshot) or the application will abort.

On the Turing cluster, Houston is run on the compute nodes.

To start Houston, run the script ~rfiedler/bin/Houston. It will submit an interactive batch job, wait for it to start, and then remind you how to connect using Apollo. You can specify a time limit and the number of CPUs; the default is 30 minutes on 2 CPUs (1 slave process). When your job starts, you will have to ssh from your local system to `turing.cse.uiuc.edu` again, this time forwarding port 4041 from your local system to the "home node", i.e., the Turing compute node where the Houston master process is running in the batch job. The script will tell you which node is the home node, and you will connect using a command like:

```
% ssh -L 4041:(home_node):4041 turing.cse.uiuc.edu
```

where (home_node) is of the form tur?-??.

On a MS Windows system using SSH, you have to open a new SSH session, and before connecting go to the "Edit Profiles" menu, select the entry for Turing, and on the "tunneling" tab add an entry to forward port 4041 to the home node. This is not pretty because you will have to edit this setting whenever your job runs on a different home node.

On most other systems, both the master and slave processes typically run on the front-end system, although they could be run in batch. The master process primarily relays data between the slave processes and the client. Most of the computational work is done by the slaves. Communication between these components of Houston is performed using the MPI library to pass messages. Since Houston is an MPMD application, we use the MPICH version of MPI over ethernet.

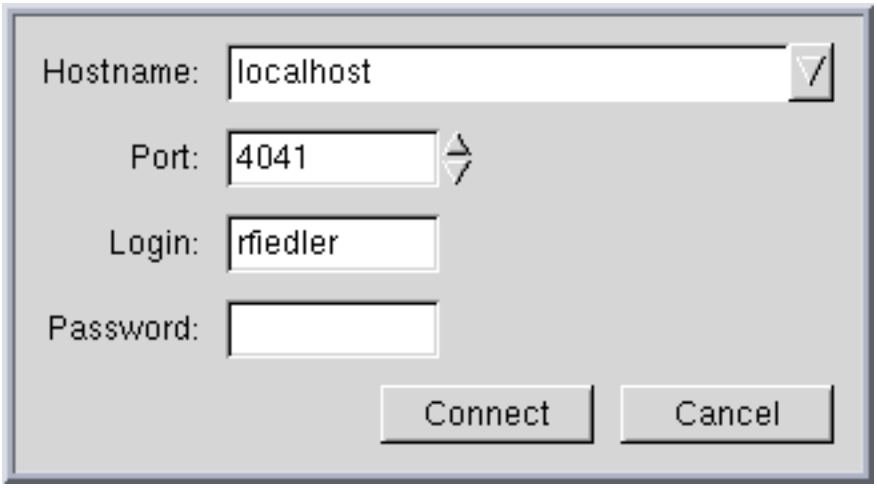
Running Houston on the IBM SP (AIX) is similar to but simpler than the procedure above for Mac clusters. See the Readme file for details.

Running Houston on the CSAR Suns is also similar to the above description, but you do not have to do any port forwarding. Just log onto the CSAR Sun server of your choice and run the launching script:

```
raphson > ~rfiedler/bin/Houston [n_slaves]
```

Note that the procgroup file must contain the name of the system on which you want the Houston master and slave processes to run. The script will show you the existing procgroup file (called `${HOME}/Houston_pgfile`) if it exists, or it can generate one for you automatically.

When Houston starts on any system, it issues a message saying that it is listening on port 4041 (or higher if another Houston process is already running on the system — on server systems with firewalls you may have to forward additional ports by including additional strings such as `-L4042:(remote host):4042` on the ssh command line). At this point Apollo can connect to Houston. When you start Apollo on your local system, a dialog box appears:



in which you specify the name of the server system (i.e., the system running HoustonMaster; here the server system is actually `turing.cs.uiuc.edu`, but since you must use port forwarding, choose `localhost` instead) and the port number (the default is 4041; it must match the one output by Houston shortly after it starts up). You can set up your account as described above using ssh-keygen so that you do not have to provide a password here. If the connection is successful, Apollo remembers the server name for subsequent sessions. Once Apollo connects to Houston, its behavior is quite similar to that of the basic version of *Rocketeer* .

Since Houston may not terminate automatically when Apollo quits, the Houston script prompts you to kill all of the Houston processes simply by typing "q". We also provide a utility called "KillMaster" (in `~rfiedler/bin`) to terminate conveniently any Houston master or slave processes. On AIX, these scripts are in `Houston_SP/utils`.

5 Using the Batch Mode Tool

Voyager processes in parallel a series of HDF files, applying the same set of graphical operations to each data file. It is particularly useful when the data to be visualized in each snapshot fits in the memory available to one CPU.

Voyager is known to run on both the CSAR Suns and the now defunct old CSE Linux cluster. It uses the LAM MPI implementation on the Suns and Myrinet MPI on the Linux cluster.

Voyager requires an X server with *OpenGL* on each processor that it uses. It could be made to work on a linux cluster that has XFree86/Mesa on each node, but on a large heterogeneous cluster, installing XFree86/Mesa can be very labor intensive for the system administrator. For the defunct CSE Linux cluster, we had obtained a commercial quality X server that runs in a virtual frame buffer from Xi Graphics.

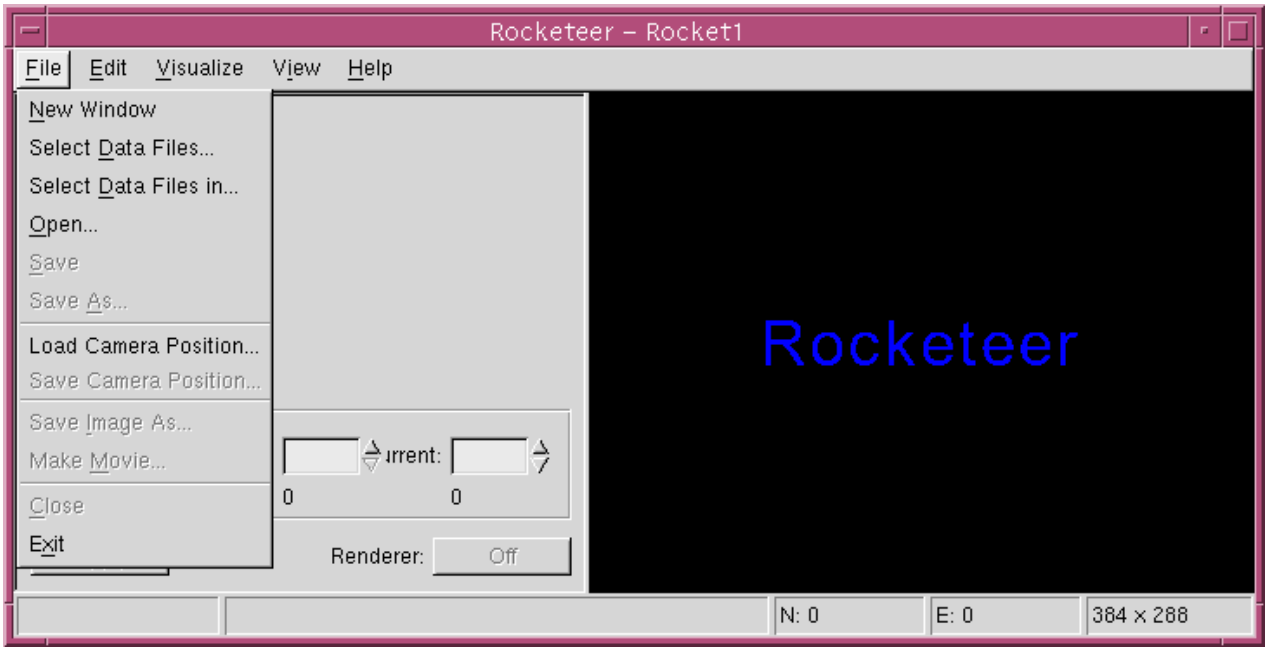
Since the rendering of images is done in software, Voyager does not benefit from hardware graphics acceleration. However, the main bottleneck when processing images from many large HDF files is I/O bandwidth, not the speed at which the final images are rendered.

The input needed by Voyager to process a series of HDF files is generated and saved during a session of one of the interactive versions (see below). For details on using Voyager, see the Voyager web page.

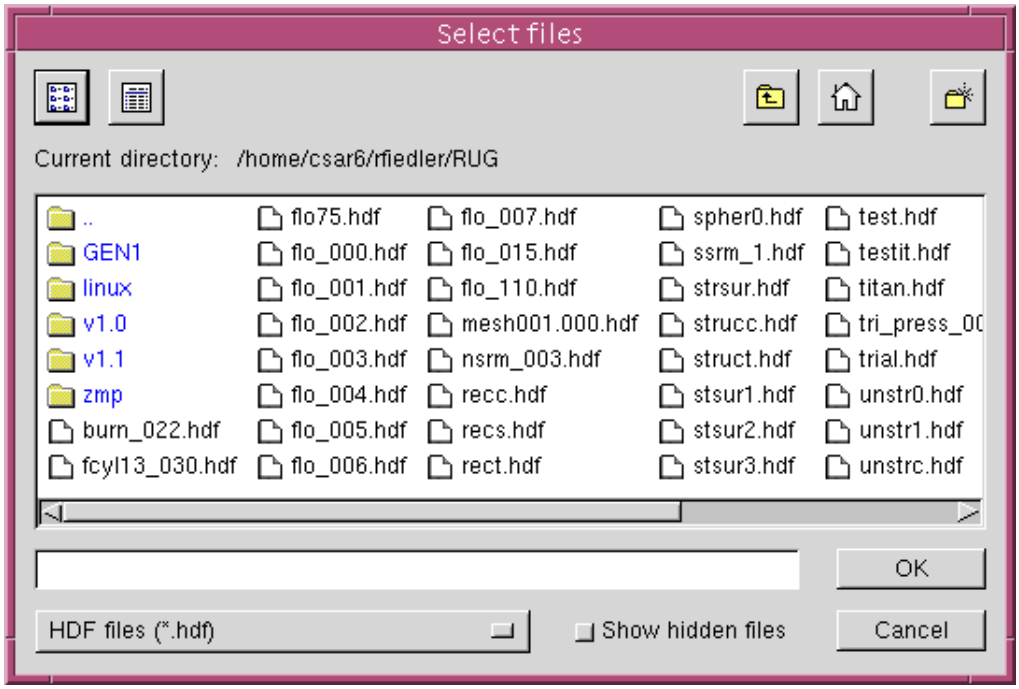
6 Basic Use of Rocketeer

6.1 Opening Data Files

Download the two example HDF files, solid_08.400000_000.hdf and solid_08.400000_001.hdf. Start Rocketeer by typing "Rocketeer", provided your environment is set up as described above. If you click on the "File" menu, you should see something like the following window:



Go to the "File" menu and choose "Select Data Files...". Choosing "Open" instead of "Select Data Files" tells Rocketeer to open a session file that you would have saved during a previous use of Rocketeer, rather than data files. After choosing "Select Data Files", you will see a dialog box for your current directory:

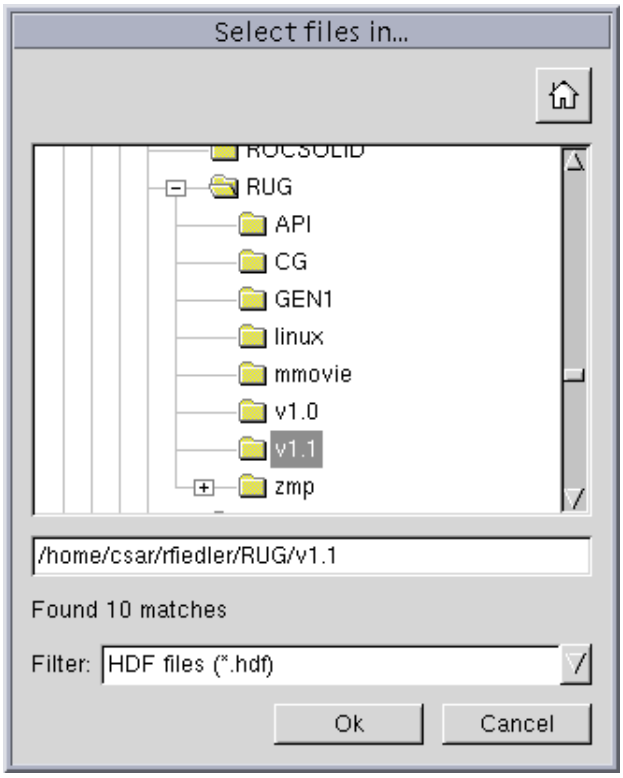


If you hold the mouse over one of the icons along the top, some information about its function will appear. You can choose a summary or detailed listing by clicking on one of the two icons on the top left. The icons on the right are for going up one directory, going to the home directory, and creating a new subdirectory (which makes little sense in this context). Any HDF Files you create should have a ".hdf" extension, but Rocketeer can open them even of they do not have it.

You can change to a subdirectory by double-clicking on its folder icon. You can also type a directory name in the white slot and hit carriage return to go to any directory.

You may select more than one HDF file using this dialog. You can also choose "Select Data Files" more than once to add more data files to the ones that you have already opened. To select a group of files, click on the first one and then Shift-click on the last one to select all files in between. You can select/deselect individual files using Control-click. Note that this is the same the way selection normally works on Windows and Macs.

An alternate method of selecting data files is available through the "Select Data Files In ..." menu item:



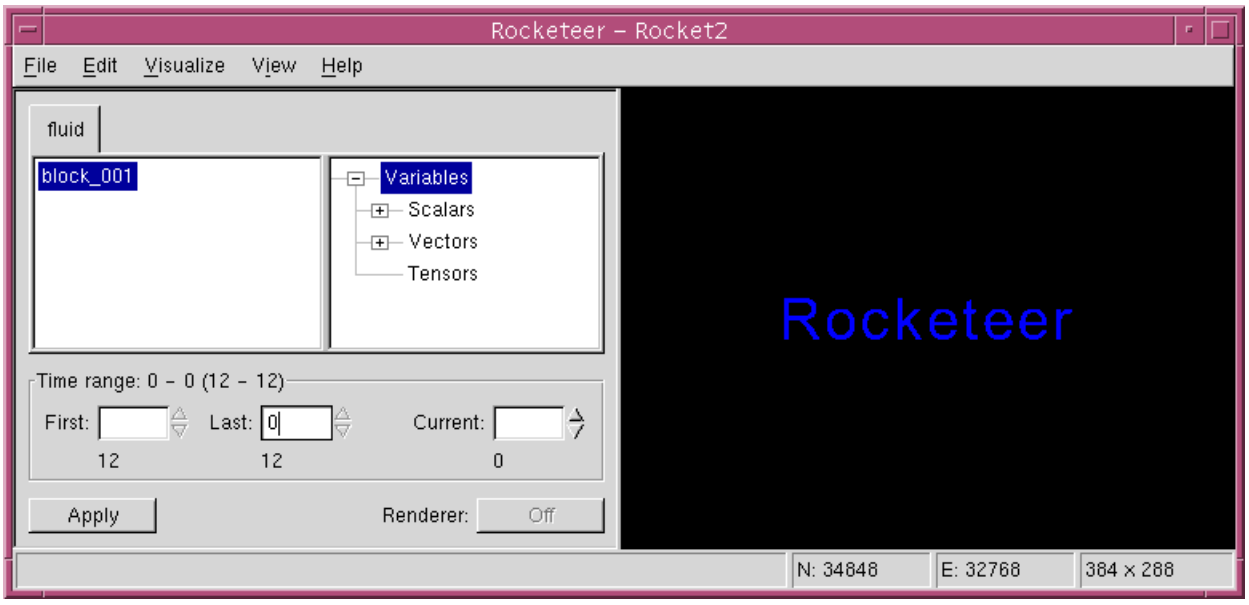
In this dialog box, the directory names are listed, but not the files they contain. Instead, files can be selected by typing a "regular expression" (containing UNIX style wildcard characters) in the "Filter" box. Above this box, it tells you how many matching file names there are in the selected directory, given the currently entered regular expression. You can substitute "HDF files (*.hdf)" with a string such as "s*.hdf" to select all HDF files that begin with the letter "s". This file selection method is particularly useful when your directories contain a large number of HDF files.

Each HDF file can contain multiple blocks, multiple time levels, and/or multiple materials. Information about the data is stored in the HDF files along with the data itself so that you do not need to give Rocketeer this information yourself through the GUI. We will go into detail about the file format later.

Select the two example HDF files that you just downloaded and click "OK" to open them. These HDF files contain data defined on a multiblock unstructured grid. The coordinates for the grid are included in this file.

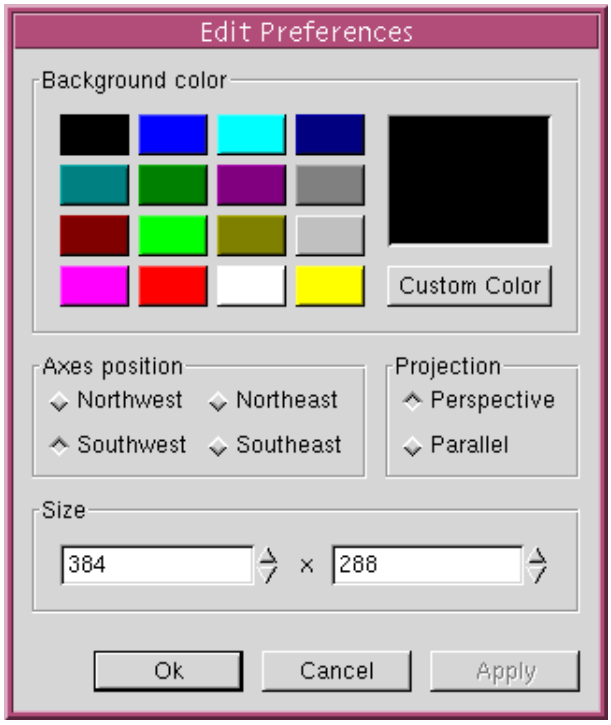
In the figure below, the "solid" tab indicates the material type attribute found in the HDF file. The two-paneled window attached to this tab lists the mesh blocks defined in the file (left hand panel) and the types of variables stored in the file (right hand panel). The "Time range" shows all of the time levels in the files you opened (here just level number 0), and the "First" and "Last" controls let you restrict the range of output times that you wish to visualize. The choice of output times can affect the dynamic ranges of the field variables and therefore the color scales used to represent the data. The "Current" control lets you select which time level to view. Rocketeer shows both the number or index of the file (starting

with 0 and ranging over all the files you have opened) as well as the time value read from the file (in this case, 8.4).



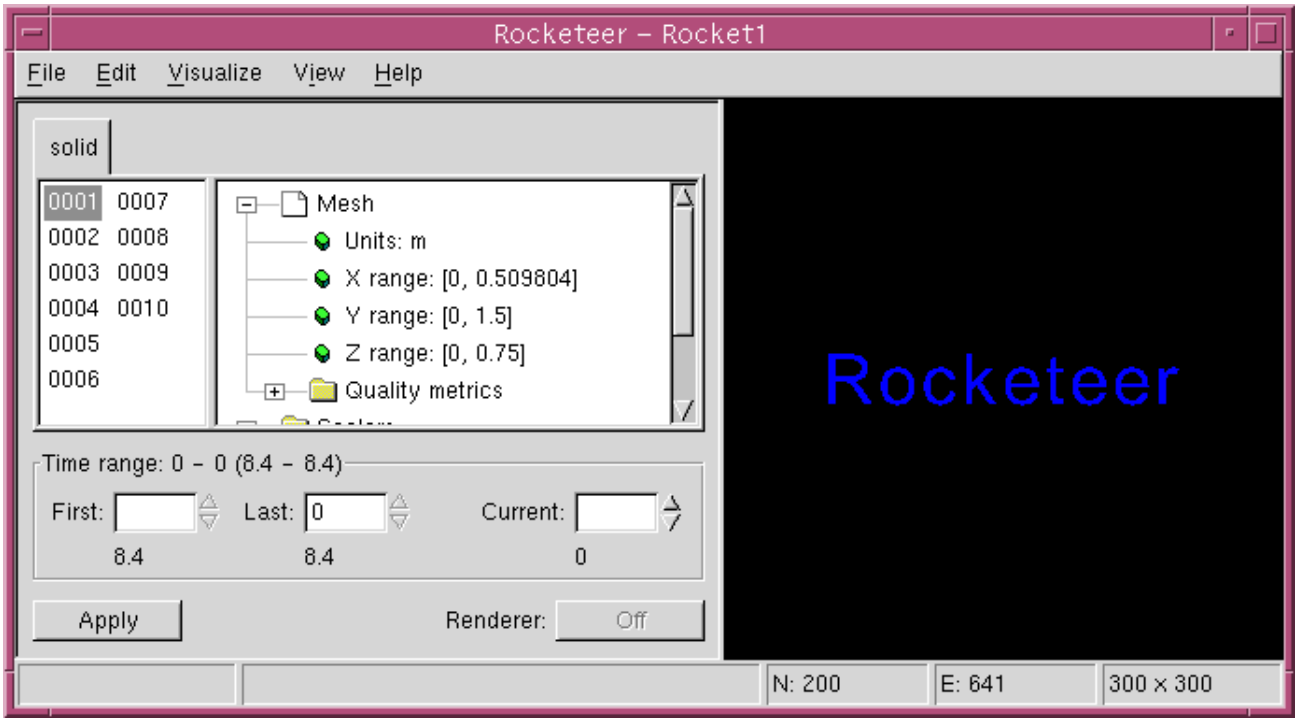
Select one or more blocks from the left hand panel and then click "Apply". (This step is unnecessary if there is only one block.) Values for the number of nodes (N) and number of elements (E) in the selected block(s) (unless you've selected and then deselected some blocks) appear in the boxes along the bottom of the Rocketeer window to the left of the image size information.

In the lower right corner of the main window, the image size in pixels is displayed. This information is useful for generating images of a particular size or aspect ratio. You can change the image size by dragging the lower right corner of the window, but a more precise way of specifying the image size is available through the "Edit/Preferences" dialog. For example, you can choose 300 × 300 pixels, and Rocketeer will save that as the default image size. With this dialog, you can also choose the background color, type of projection, and location of the orientation axes on the image. You may need to hit one pair of up/down arrows to get it to apply the changes, and you may need to do this more than once to get the desired size.

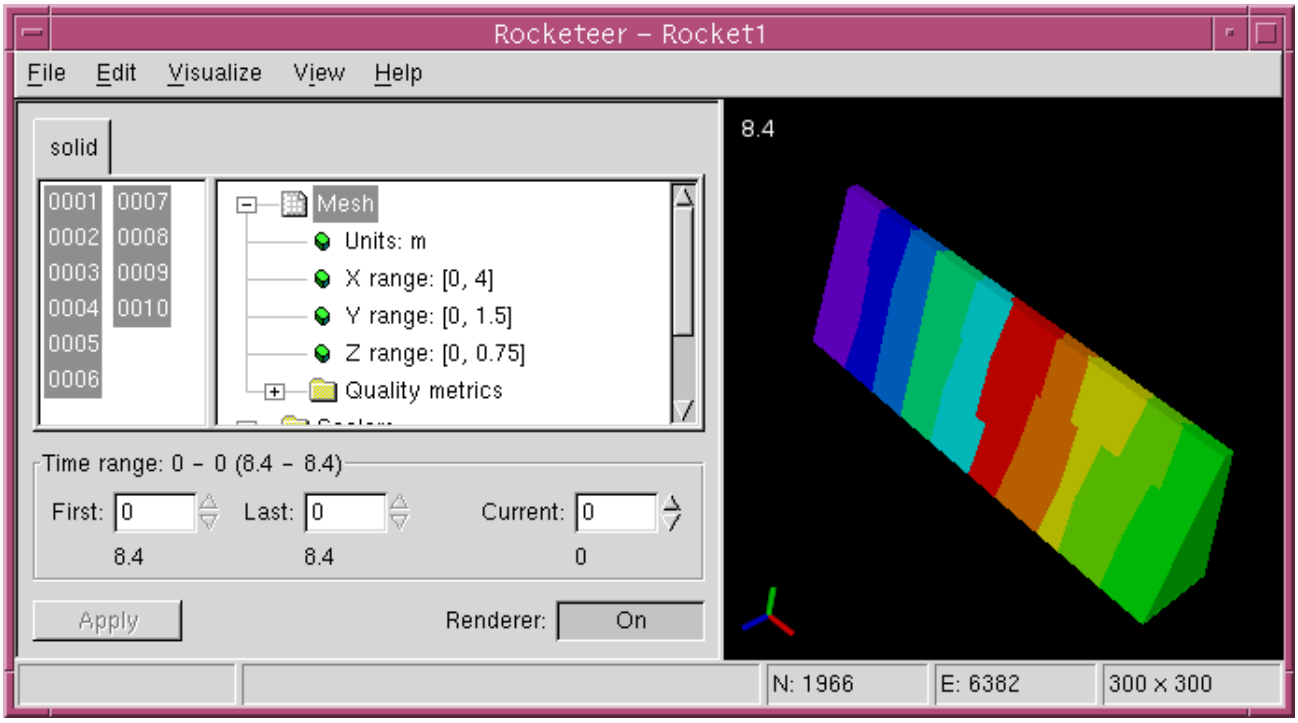


6.2 A Quick Look at the Mesh Blocks

Returning to the main window, you can click on the plus signs next to the "Mesh", "Scalars", "Vectors", and "Tensors" icons to see metadata for the data sets contained in the file. In the screen shot below, we've resized the 2 left panels by clicking and dragging the border between them. You can see the ranges of the x, y, and z coordinates of the selected block ("0001").

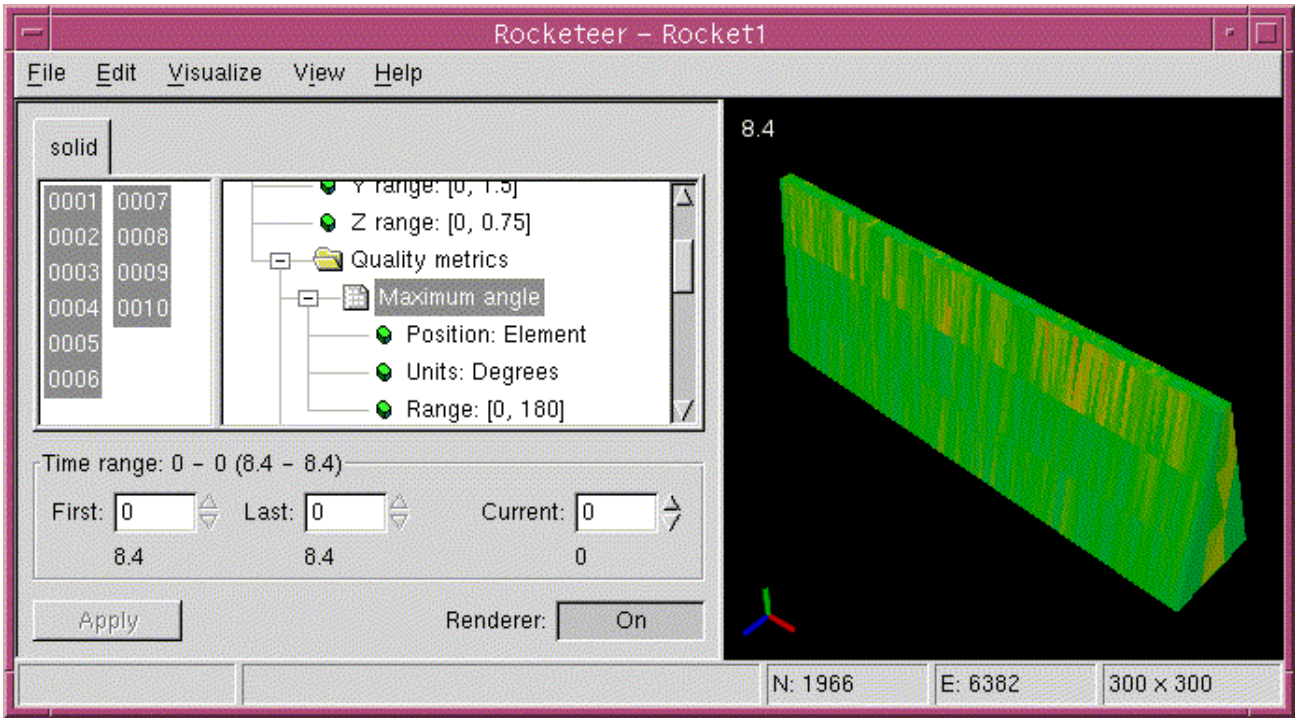


You can select more than one block in the same manner as you slected more than one HDF file to be opened. To display a surface plot of all 10 mesh blocks, each in a different color follow these steps: Select all 10 blocks and hit "Apply". The number of nodes and elements in all 10 blocks will now be displayed next to "N" and "E". Click on the "Mesh" icon to select the mesh as the variable on which graphics operations should act. Now go to the "Visualize" menu and select "Add Surface\Grid". Finally, click the "Renderer" button:

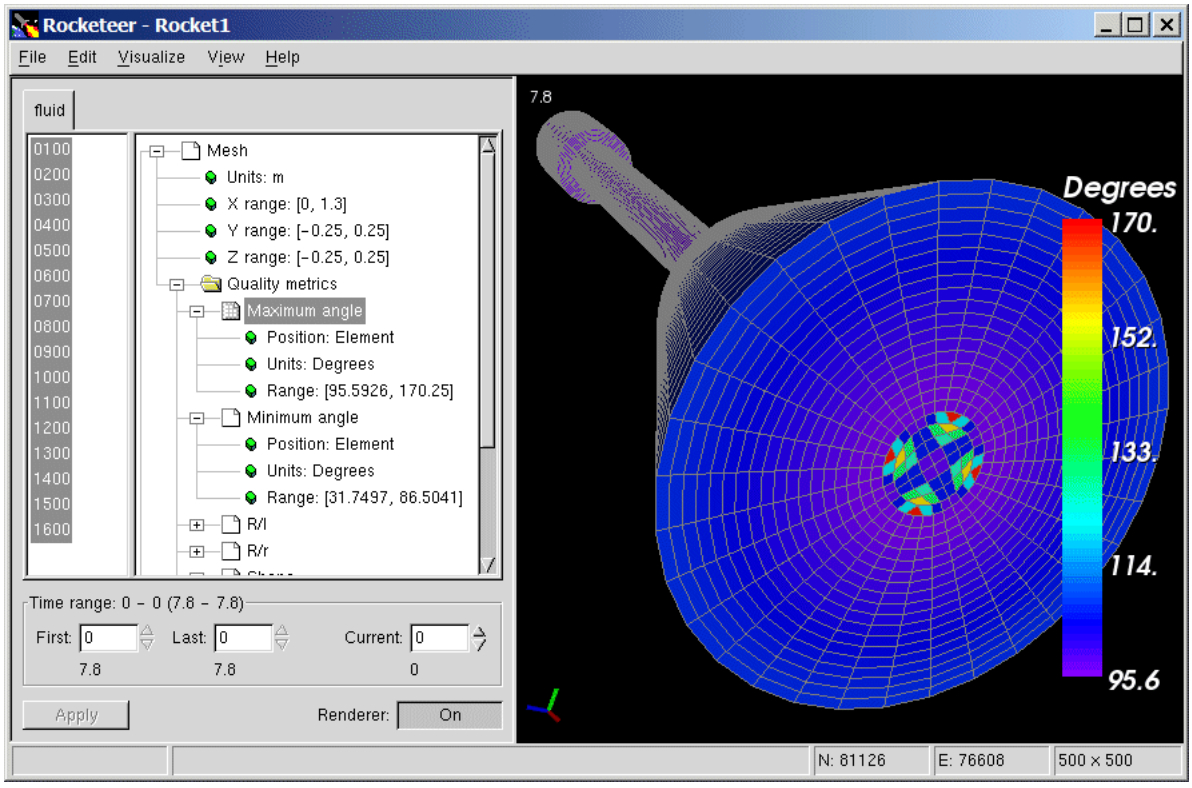


6.3 Mesh Quality Measures

Rocketeer version 1.3.6 can display various mesh quality measures. The built-in set of algebraic mesh quality metrics are those provided in a paper by Knupp. These metrics are implemented as plug-in libraries so that the user can add different metrics as desired without recompling Rocketeer. As an example, the figure below shows the maximum dihedral angle:



Below is a more recent example using the structured mesh lab scale rocket data set. The maximum dihedral angle is nearly 180 degrees in "corner cells" along the "core block" that runs along the axis of the rocket. These distorted cells are present because we do not want a singularity in the grid on the axis.

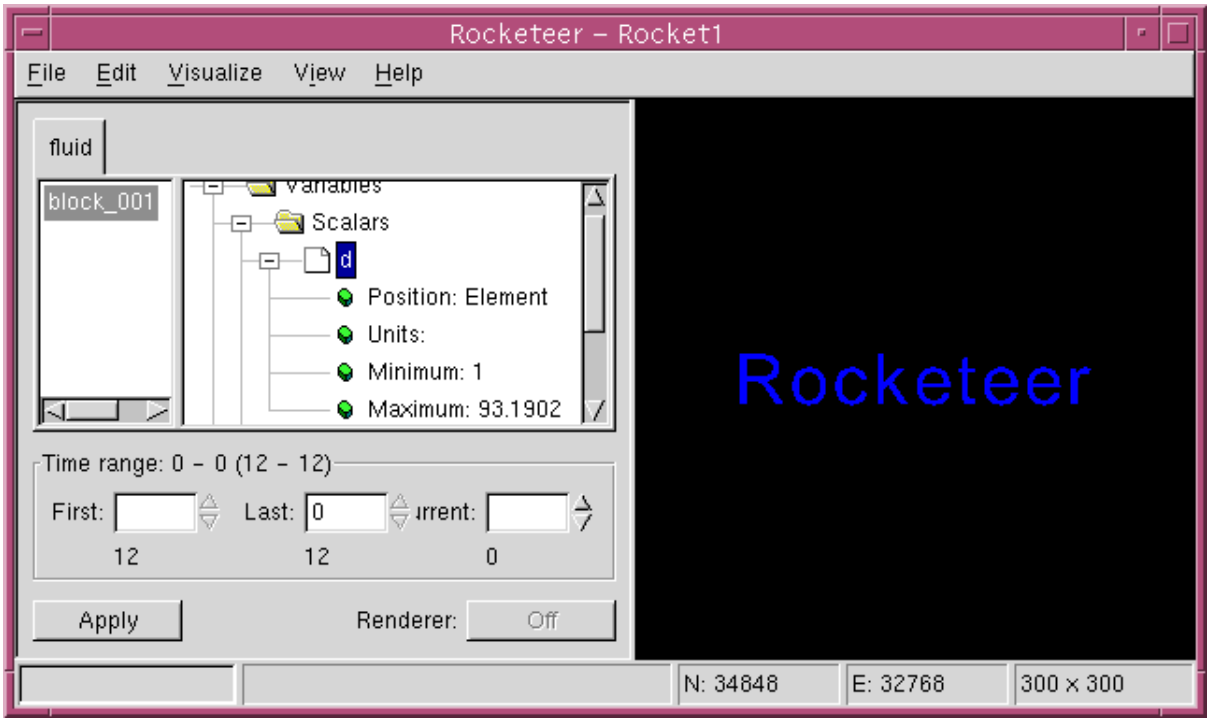


You can show the mesh itself in the same image as described below using the – Edt/Edit Surface \ Grid/Grid" menu item.

6.4 Surface and Grid Plots

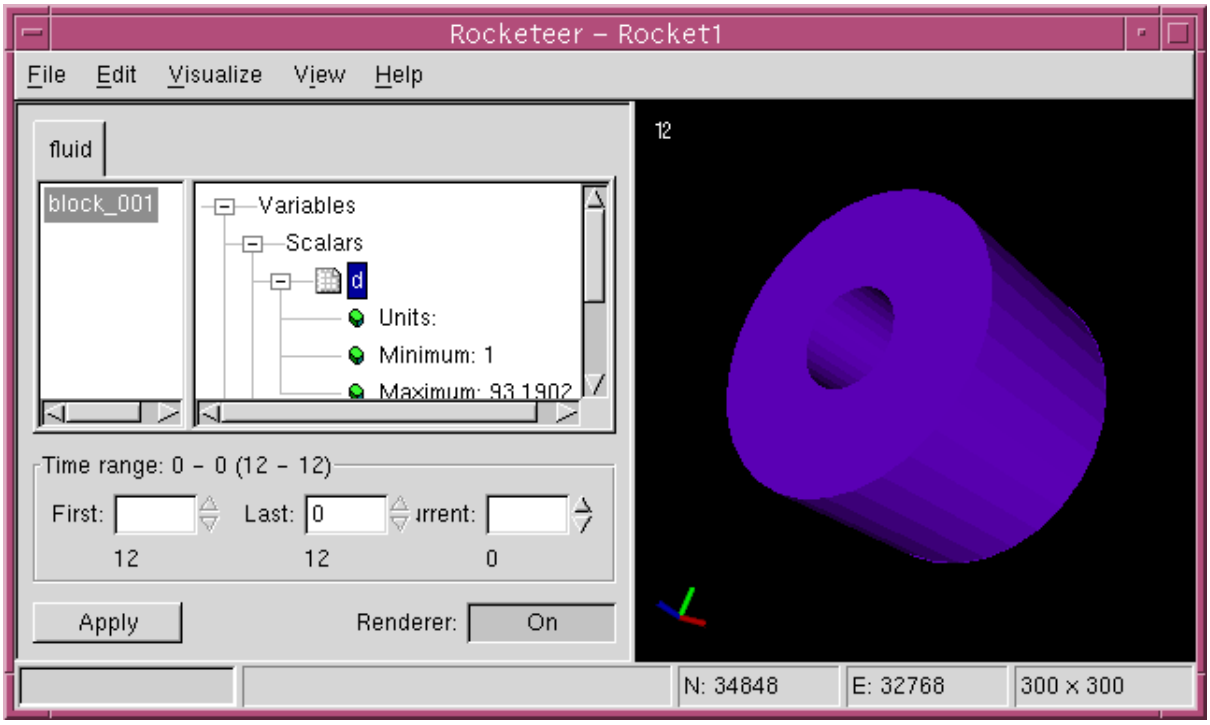
For the remainder of this User's Guide, the images were made using Rocketeer v1.3. There are small differences in the appearance of the GUI, such as how the range is displayed.

Download and open zmp_012.hdf for the next portion of this tutorial. Click on the Scalars icon and then on the variable called "d". You should see a window like the one below.The "position" attribute refers to whether it is a node centered or element centered qualtity. The physical units (if any were specified) of each data set and their ranges are also displayed. For vector variables, the range of the magnitude is displayed and can be plotted as a scalar. For tensors, the corresponding scalar is the trace. In the figure below, you can see the min/max and units of scalar variable "d" (the mass density).



There is also one vector field in this HDF file, the velocity v.

Select the scalar variable "d" by clicking on it. Then go to the "Visualize" menu and choose "Add Surface/Grid". Rocketeer determines the location of the surface of the grid. This step can be rather time consuming for a large, unstructured data set (~10 seconds for 847,000 tetrahedral elements), but structured mesh algorithms tend to be much faster. Now click on the Renderer On/Off button to view the image.



6.4.1 Aside: Positioning the Image

Initially, the x axis points to the right, the y axis points up, and the z axis points out of the page. To rotate the image, move the mouse pointer into the image panel, and then hold down the left mouse button and drag in the desired direction. You can display (or not display) the time and the colored axes as in the image above by means of the radio buttons under the "View" menu. The axes in the image window indicates the image orientation, with Red, Green, and Blue corresponding to the x, y, and z axes. You can choose the corner of the image window in which the axes appears under "Edit/Edit Preferences". The time will appear in a different corner from the one with the axes.

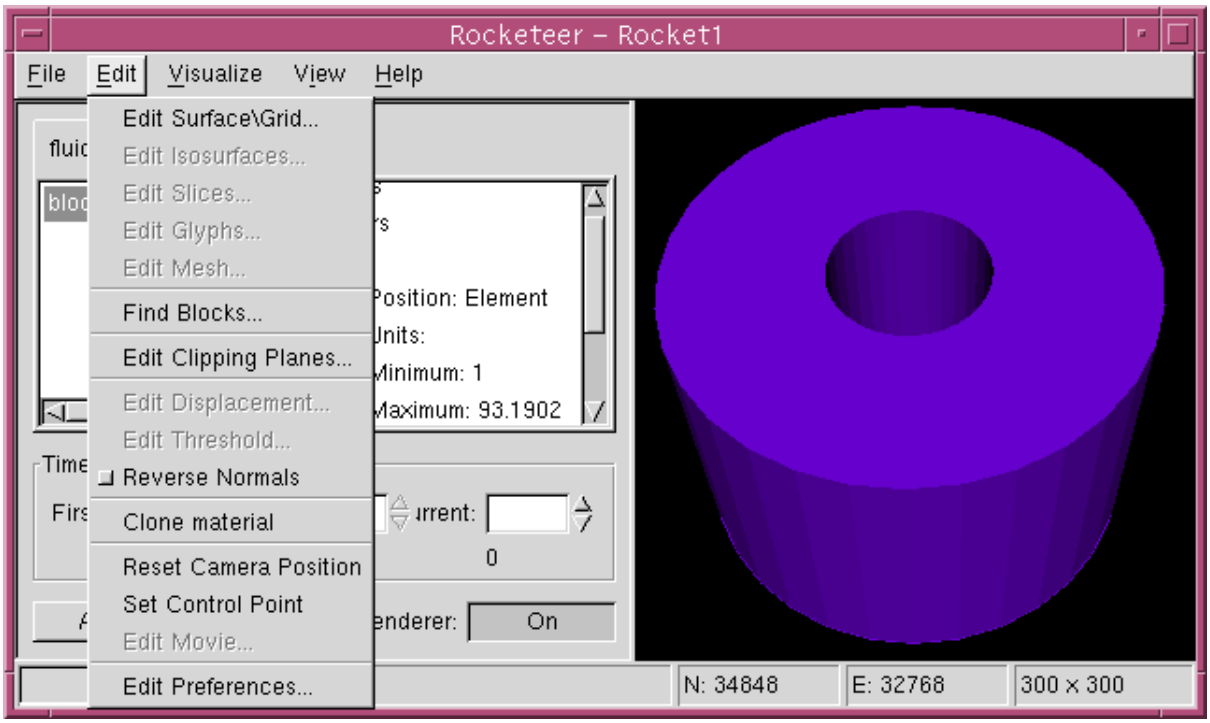
The full set of image positioning functions can be accessed as indicated in the table below:

Image Positioning Function	**Mouse Input**
Rotate the image about an axis in the plane of the screen	Left click and drag
Rotate the image about an axis perpendicular to the screen	Ctrl-left click and drag
Translate the image in the plane of the screen	Middle click and drag
Zoom in or out	Right click and drag

Orienting the image takes some practice. The image does not move with the mouse pointer when you drag across it. Instead, the amount that the image moves for a given mouse movement is proportional to the distance from the center of the image to the mouse pointer. For example, if you want to translate the image by a relatively large amount to the right, middle click inside the image window near the left edge and drag almost all the way to the right edge. It may take a while for a large image to be redrawn at its new location. It may

help to hold the mouse button down after dragging and wait for the image to be redrawn before moving the mouse again. Finally, if you accidentally drag the mouse pointer to a location outside the image window, you will have to middle click inside the image window to get it to stop moving as you move the mouse. This method applies for quitting the other positioning operations as well — make sure the pointer is inside the image window before you release the mouse button and any control keys.

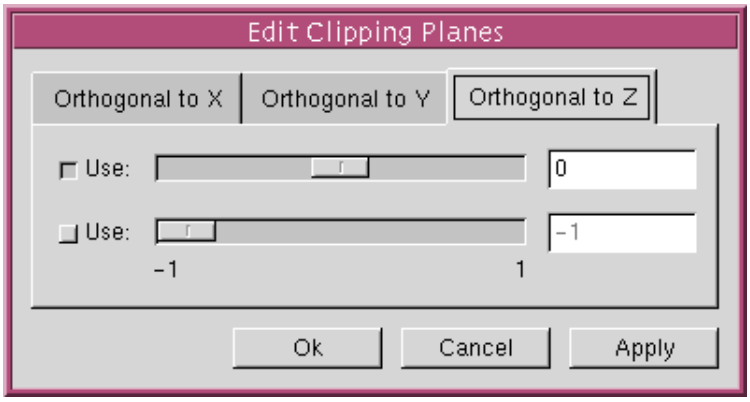
There is a "Reset Camera Position" function in the "Edit" menu that can be used to put the image back into the field of view if you have accidentally moved it out of range. This menu item is also useful for centering the axis of rotation on an image after you've deselected many blocks and hit "Apply".



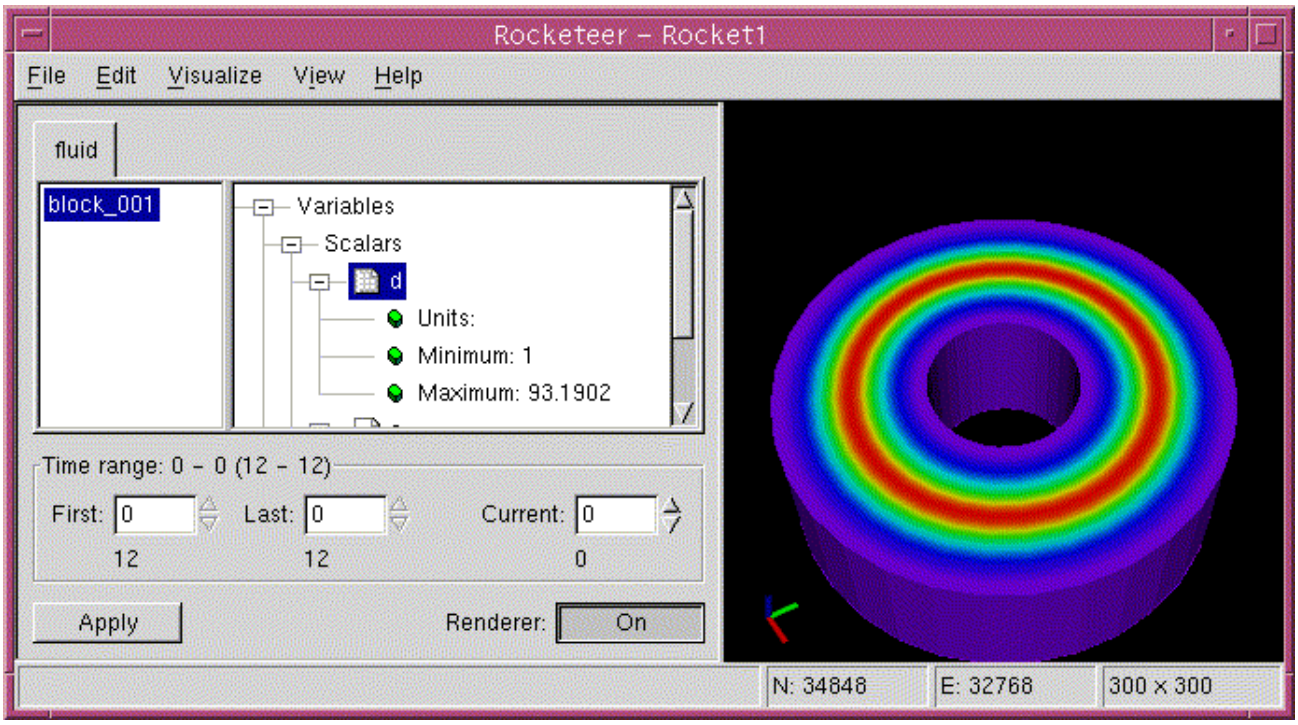
The "Save Camera Position" and "Load Camera Position" items on the "File" menu store and reload a previously saved camera position file (*.cam). This precise image positioning can be useful for comparing images from different data sets or for generating frames to add to an existing animation sequence, although **??** may be a better option for this purpose. The camera file is human readable, so one could manually edit it to position the camera as desired.

6.4.2 Aside: Clipping Planes

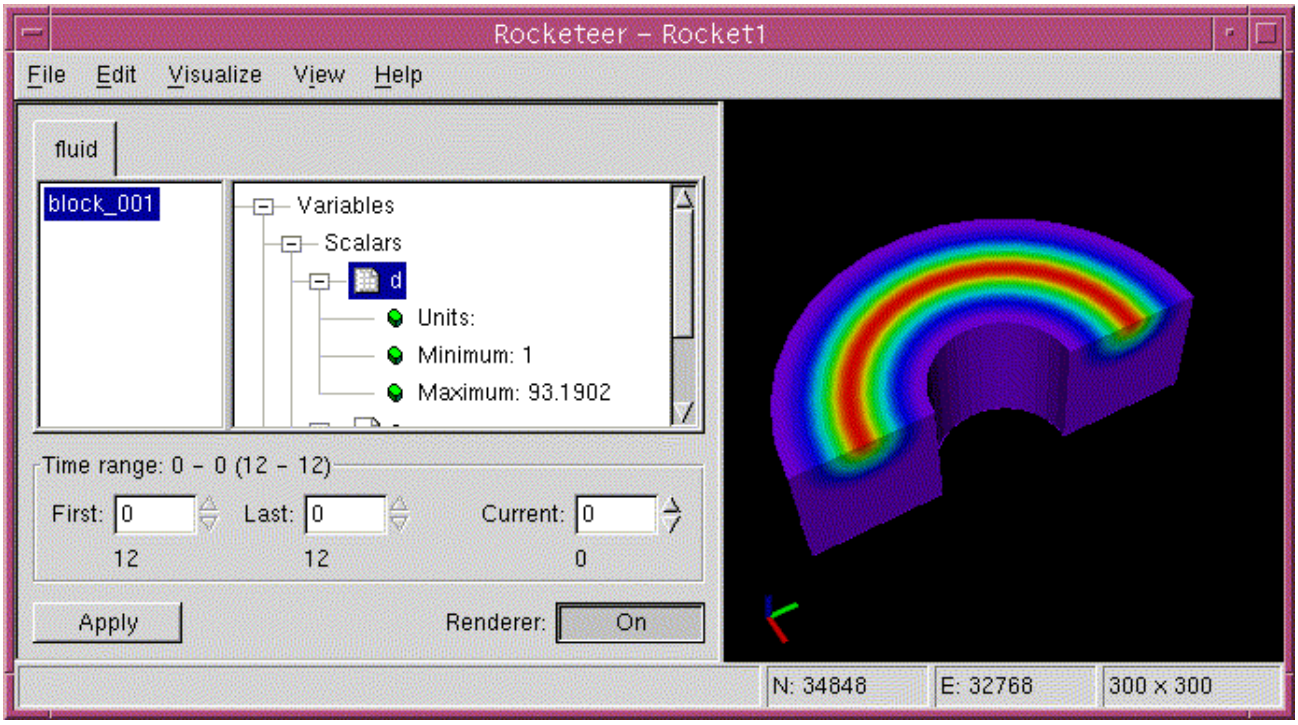
You can cut away part of the image to make the interior more visible. Rocketeer clips all visualized variables at planes normal to the x, y, or z axes only. Choose "Edit Clipping Planes" from the "Edit" menu to bring up the following dialog box.



Click on the "Orthogonal to Z" tab, and then on the "Use" radio button for the clipping plane at the largest value of z. Change the value of its location in the white box on the right from 1 to 0 and hit "Apply":

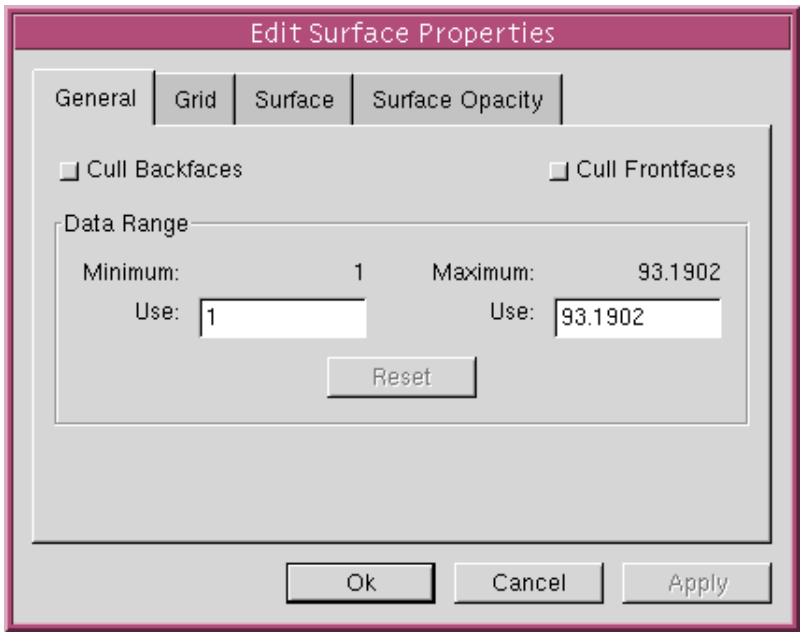


You can use more than one clipping plane at a time. In the image below, we also enabled a clipping plane at x = 0.



6.4.3 More Surface and Grid Plot Capabilities

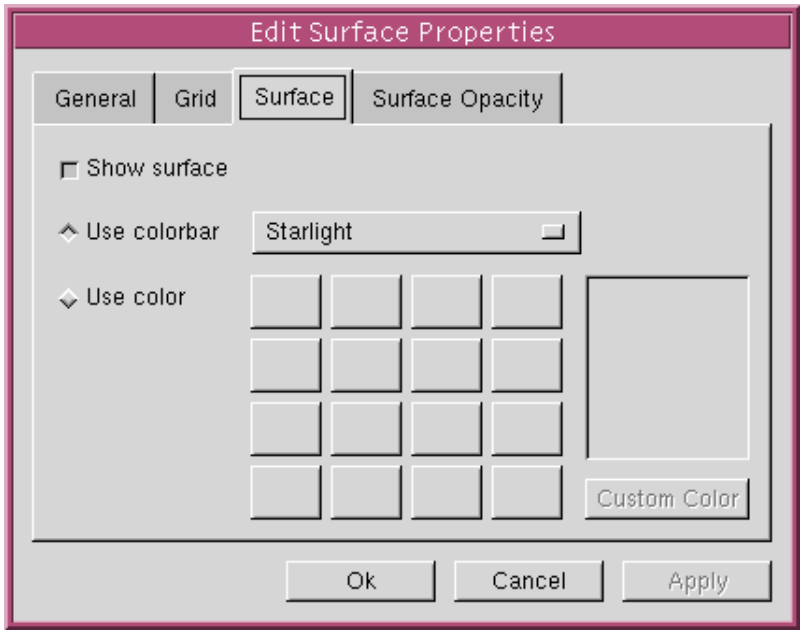
To explore more of Rocketeer's capabilities for Surface and Grid plots, pull down the "Edit" menu and select "Edit Surface/Grid". The following dialog box should appear:



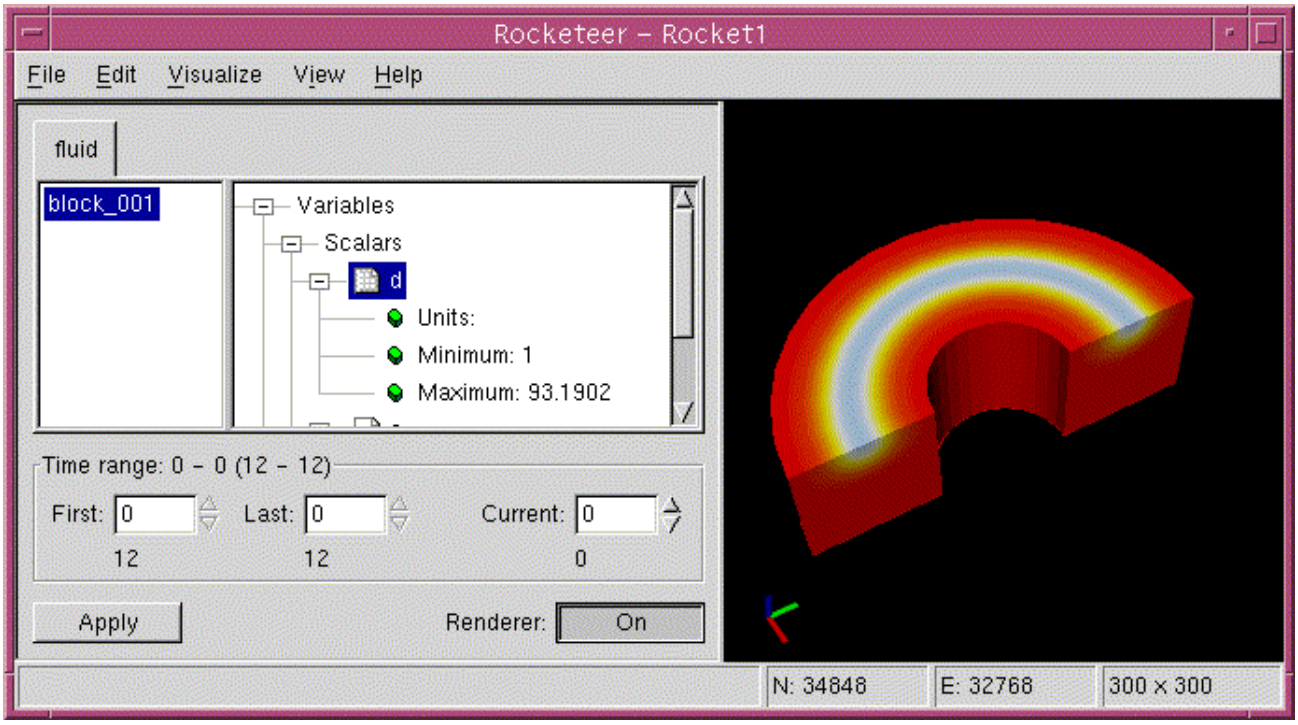
The "General" tab lets you change range of the data to be displayed on a surface plot. One example of when this would be useful is if you had a series of data sets containing a "bad"

value in one cell at some time level. The bad value could greatly extend the range of values needed to display all of the data, making it difficult to see the rest of the data. Another situation for setting the range would be when you want to extend a series of animation files, but do not want to store the full set of output dumps on disk at the same time.

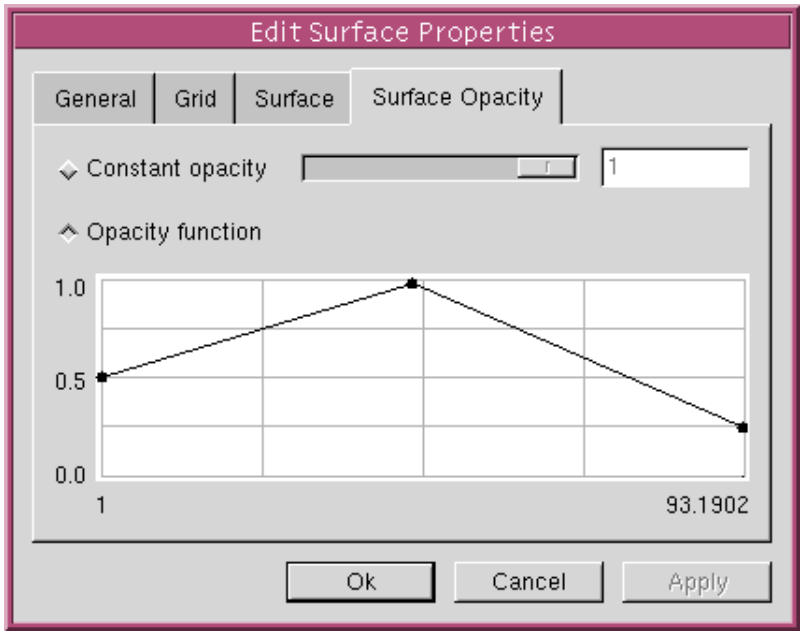
The radio buttons marked "Cull Back faces" and "Cull Front faces" control whether the polygons making up the surface whose normals point away or toward the camera are visible. This can be useful if your graphics system does not perform hidden surface removal properly, or if you want to see through the image to the back side. However, it is often more useful to make the surface translucent by changing its opacity (see below). If the image still doesn't look right, you might try clicking the "Reverse normals" button on the "Edit" menu to change surface inner normals to outer normals and vice versa.



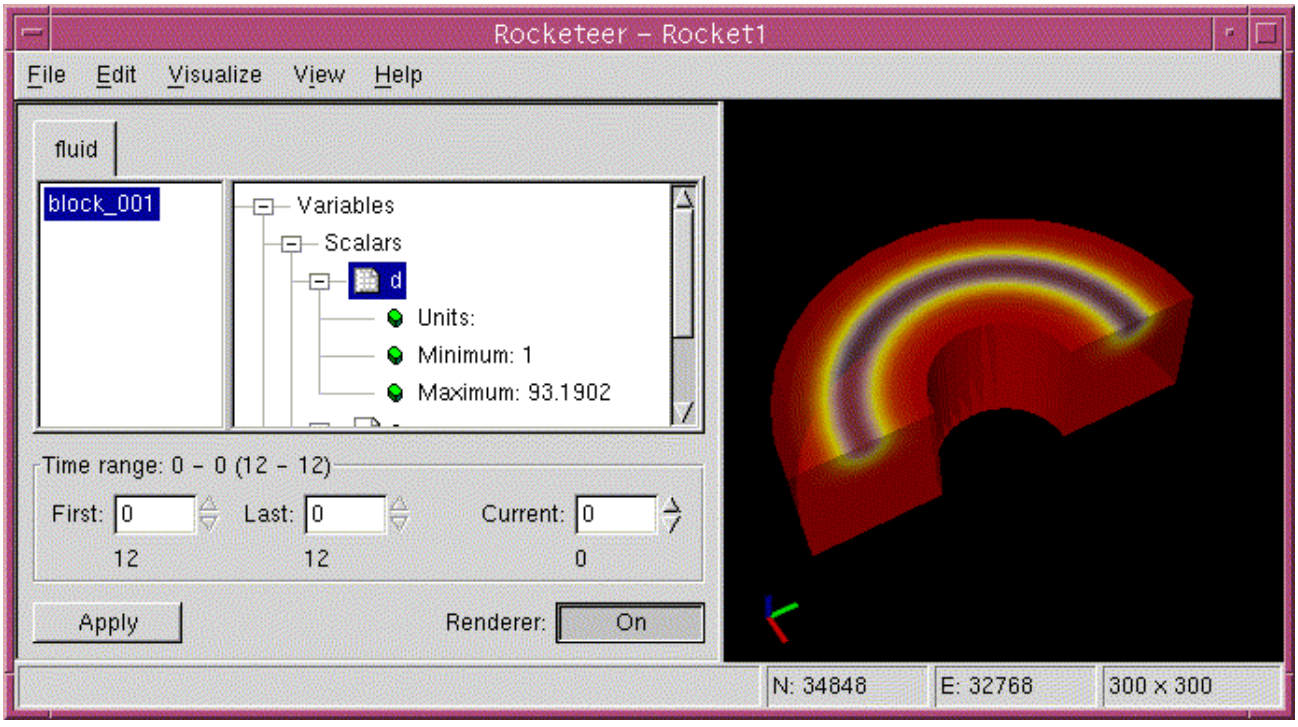
The "Surface" tab allows you to choose whether the surface is visible at all ("Show Surface" radio button). You can also choose to display a field variable on the surface using a color scale ("Use colorbar") or using just one color ("Use color"). A single color would be useful if you just want to show the position of the surface, rather than the variation of a variable on it. If you wish to use a color scale, you can select one from a list of built-in and user-generated color tables. Here we have selected "Starlight", which ranges from red to yellow to white to blue like the colors of increasingly hot stars/flames:



Select the "Surface Opacity" tab to make the surface more transparent. A constant opacity function (equal to 1 to make the surface completely opaque) is selected by default. You can change the value of this constant either by using the slider or by entering a number (e.g., 0.5) in the small white box to the right. To make the opacity a function of the variable you are plotting, click the radio button marked "Opacity function". Then, in the opacity function window, you can click and drag to add or move "control points" for the curve (line segments) that you wish to use. To remove a control point, just move it on top of a neighboring control point. On some systems, objects appear opaque regardless of the opacity function you set. It may look better if you render the image in software, which can be done using the "File/Make Movie" menu item (see below).



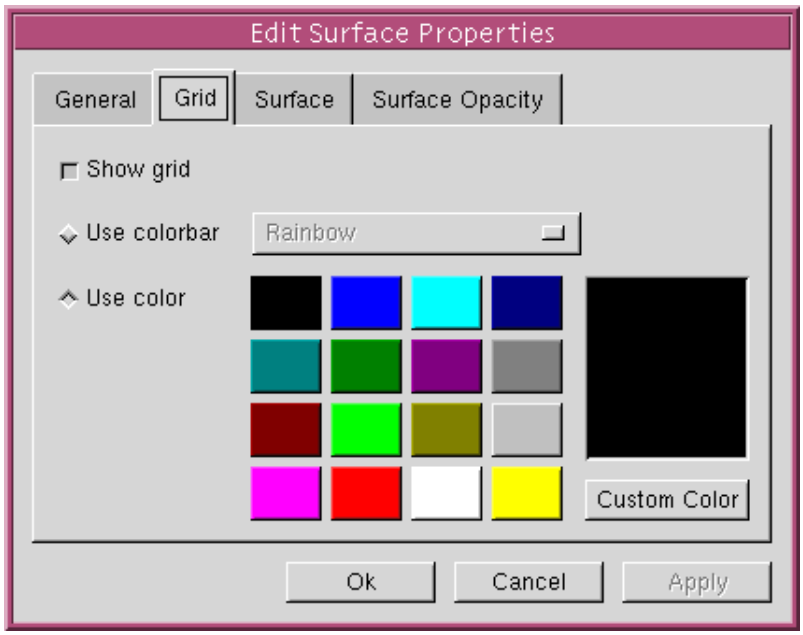
The downside of setting the opacity below 1.0 is that the colors are duller (and there may be some strange lighting effects caused by your graphics card's implementation of OpenGL or bugs in the VTk library routines). The advantage is that you should be able to see through the surface.



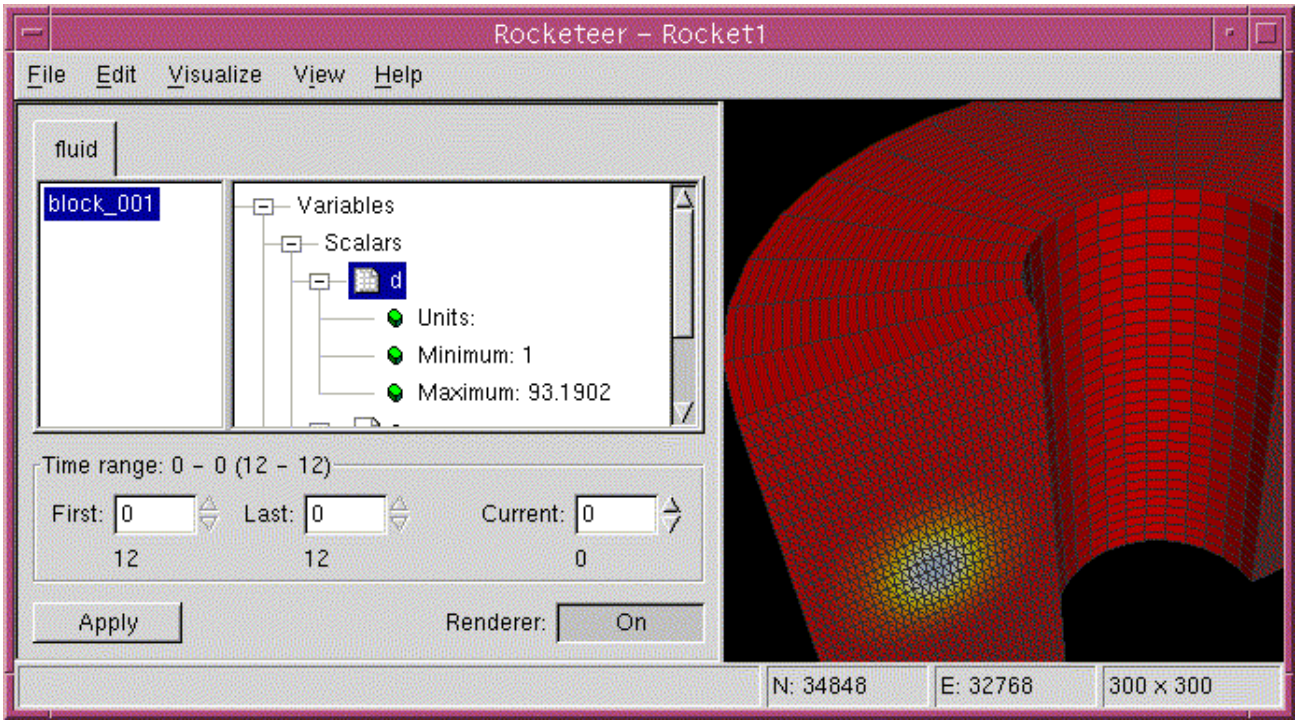
(More examples of translucent surfaces appear below.)

Select the "Grid" tab to specify whether the grid should be visible and whether it should be

all one color or vary with the field variable you are plotting. Here we have clicked the radio button marked "Show grid" and selected a single color (black):



In the image below only the x clipping plane is activated. The grid looks OK on the real surfaces of the domain, but on the clipped surface, the grid cells are crossed by a diagonal line ("triangulated & rdquot). This is how VTk shows the grid on a clipped surface. Even if the grid cells were tetrahedra, extraneous line segments would be visible on a clipped surface.



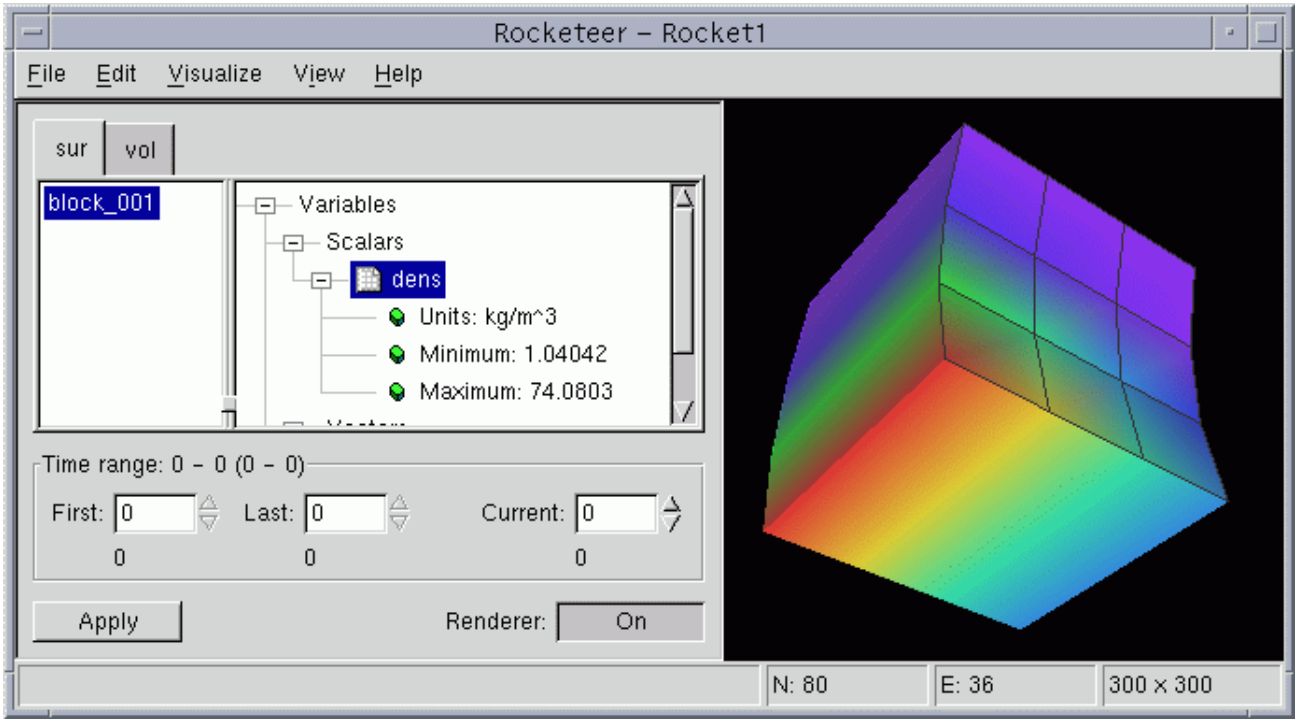
The ability to show the grid can be useful for determining whether the grid has the appropriate spatial resolution, the degree of skewness of the grid, etc. Rocketeer can also visualize the 3-D volumetric mesh (.

6.4.4 Surface Meshes

You can supply the surface mesh (triangles and quads) of an object in a separate data set so that it can be displayed more quickly. When working from a full volumetric mesh, Rocketeer takes more time to determine which nodes are on the surface of the domain.

Download the HDF file unsurf.hdf and, if you wish, the Fortran code that generates it using the This HDF file contains both data defined on an unstructured quad surface mesh and data defined on an unstructured hex volumetric mesh. These data sets are treated as being distinct because they have been assigned different "material types".

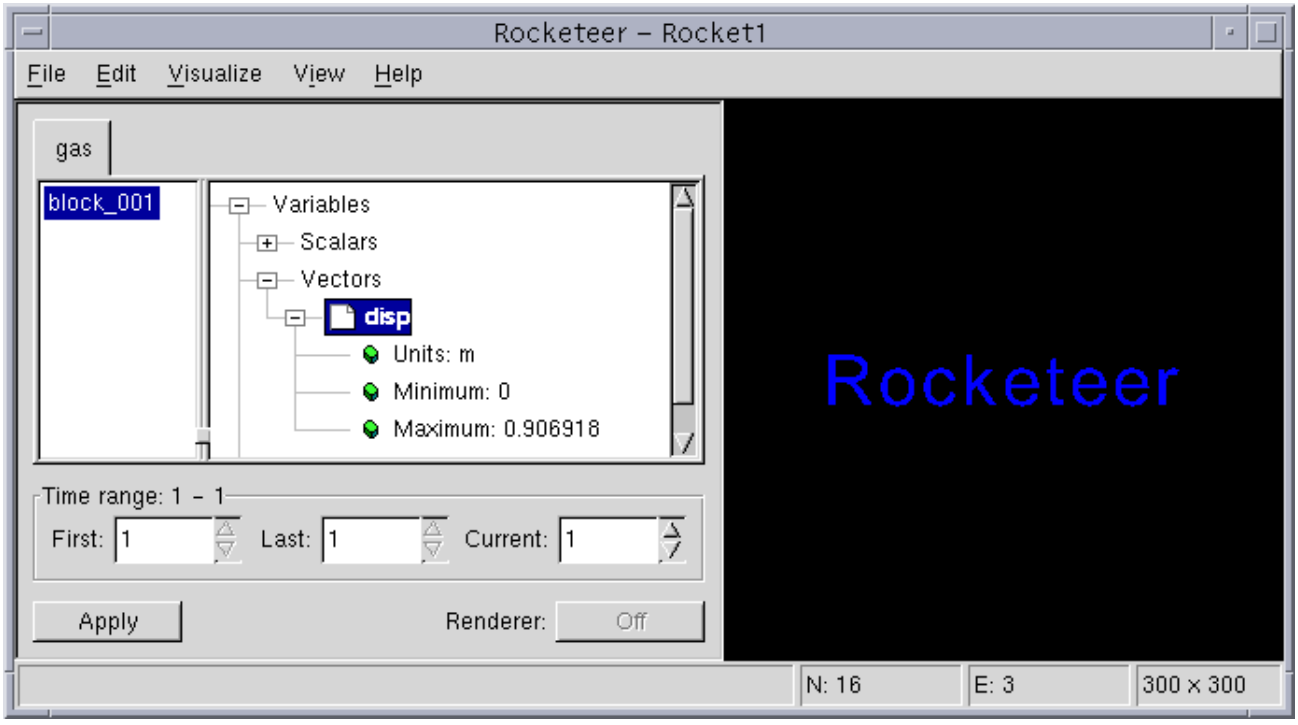
For each of the two material types, select the density and choose "Visualize/Add Surface\Grid". For the "surf" material type only, Edit the Surface\Grid plot to show the grid lines in black. The mesh is visible only on the surface described by the surface mesh – as we chose to define it in this file, it does not cover the entire surface of the volumetric mesh.



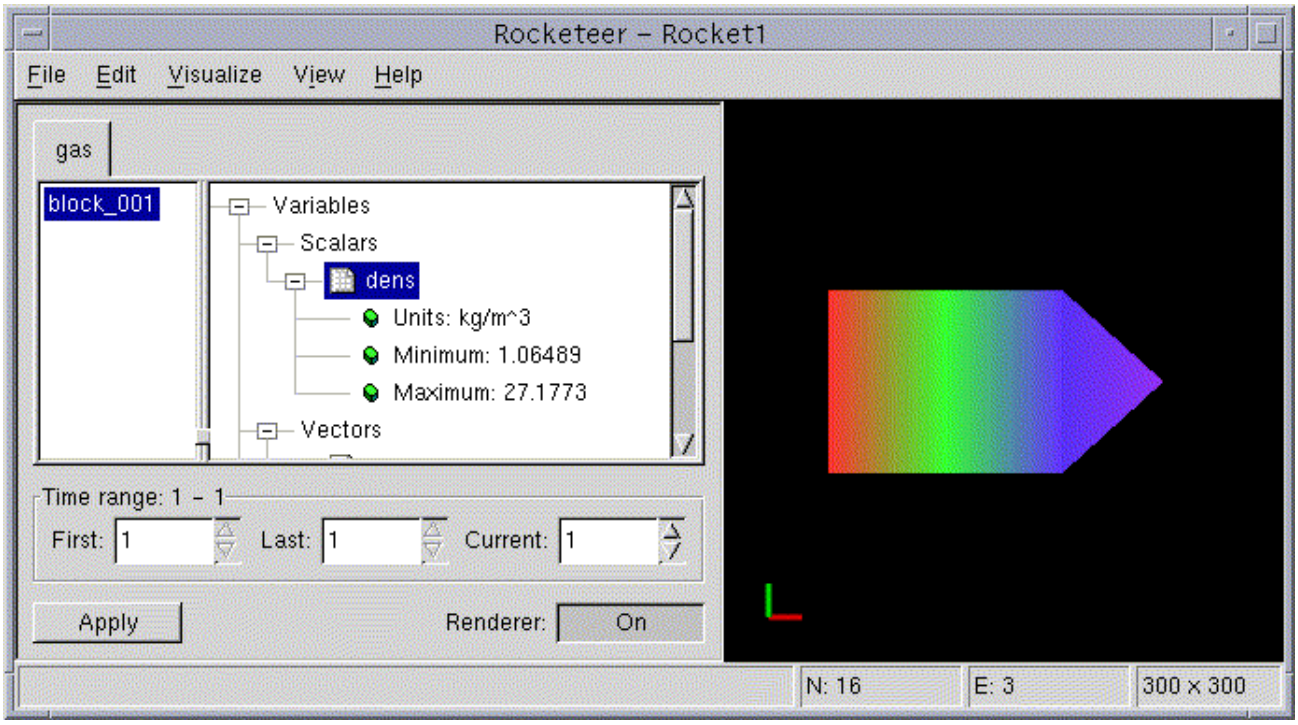
6.4.5 Aside: Displacements and deformed coordinates

Download the HDF file unstr1.hdf and open it in a new Rocketeer window. This file was generated by another Fortran code that uses the The undeformed coordinates and connectivity for this file will be read from unstr0.hdf, so download it to the directory with unstr1.hdf.

In the Variables panel, click on the Vectors "+" box and then on the "disp" (displacement). Choose "Use as displacement" from the "Visualize" menu.

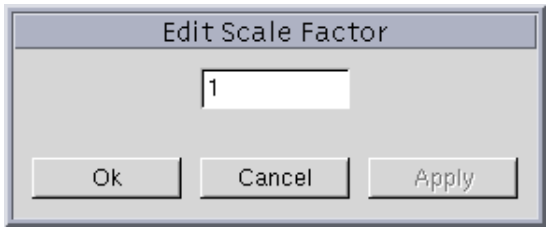


I've intentionally made the displacement large so that the difference between the deformed and undeformed configurations would be easy to see here. Now select the density under "Scalars" in the Variables panel and "Add surface/grid" from the "Visualize" menu. Finally, click the Renderer button:

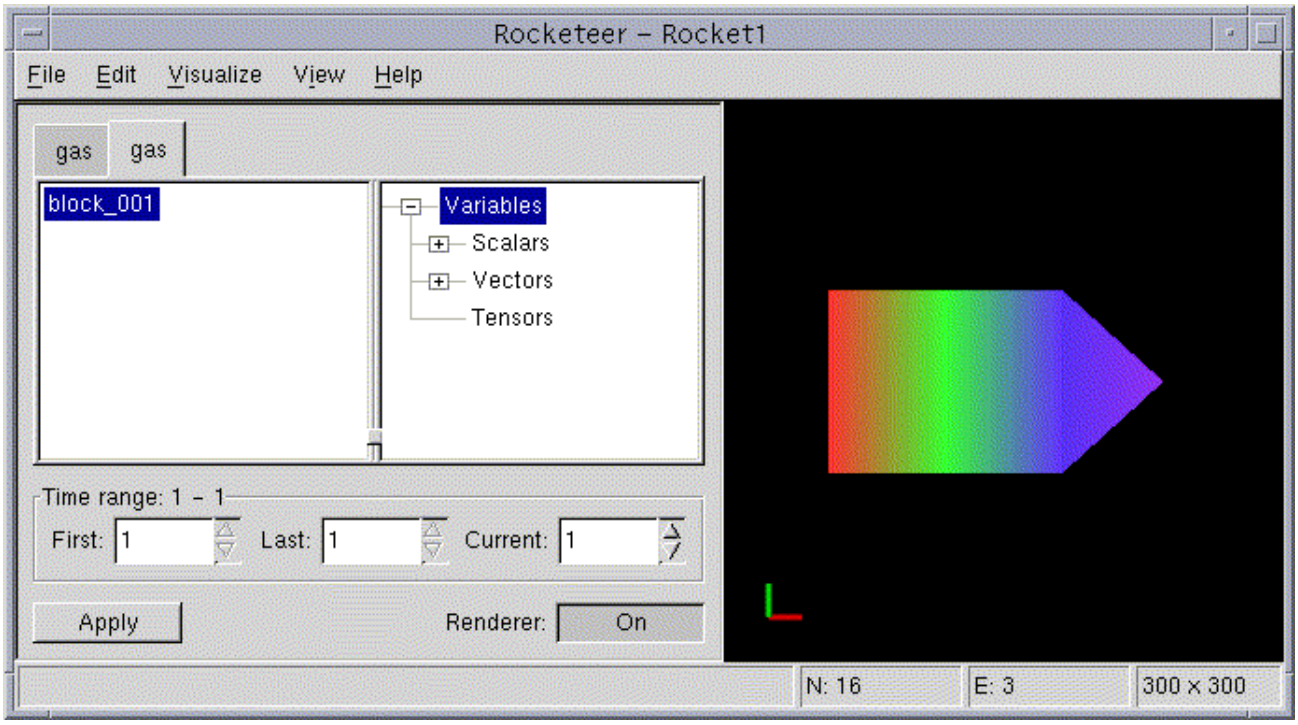


Note the difference in shape between this image, plotted on the deformed coordinates, and that of the same variable plotted on the undeformed coordinates ("Visualize/Ignore Displacement").

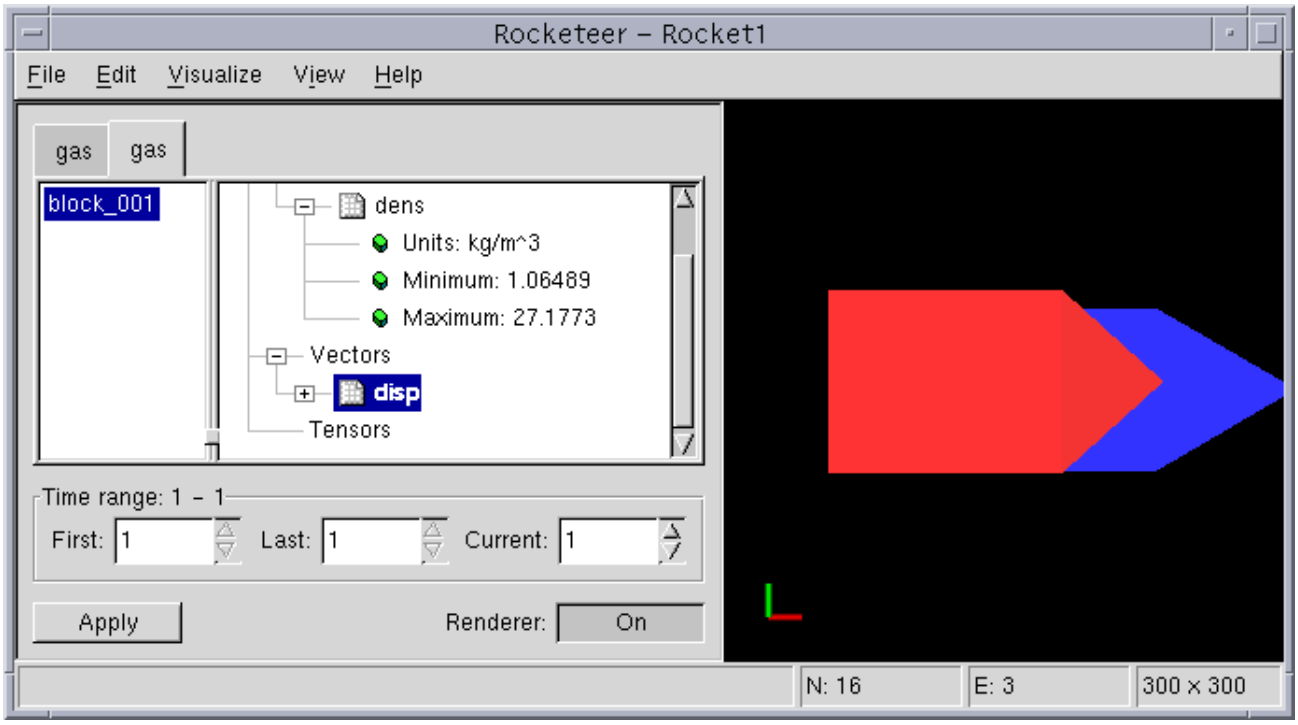
In many instances in structural mechanics, the displacement is too small to see. To make the displacement more visible, the magnitudes of the displacement vectors can be scaled using the dialog box under "Edit/Edit displacement":



Finally, it may be instructive to show both the deformed and undeformed object on the same image. To do this, select "Edit/Clone material". This creates a second material tab named "gas" which you can use to plot variables on another coordinate system.



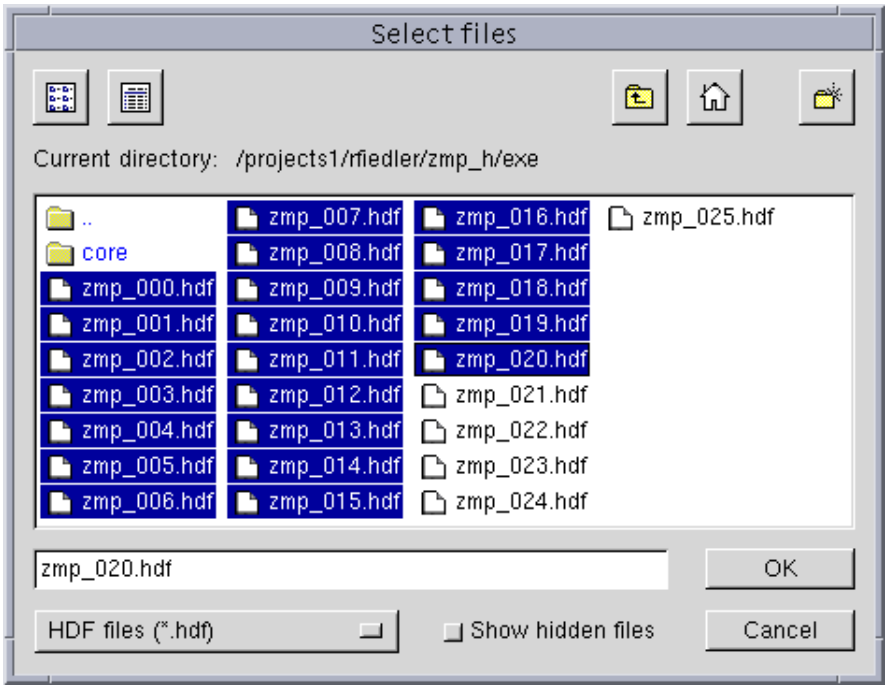
First, click on the leftmost "gas" material tab and then choose "Edit/Surface/grid" to make the deformed object all one color (blue). Then select "Visualize/Ignore displacement" to show the undeformed object. Now click on the rightmost "gas" tab, select the displacement in the Variables panel, and choose "Visualize/Use as displacement". Next, select the density in the Variables panel and pick "Visualize/Add surface/grid". Edit the surface using "Edit/Edit surface/grid" ("Surface" tab) to make the deformed surface all one color (red):



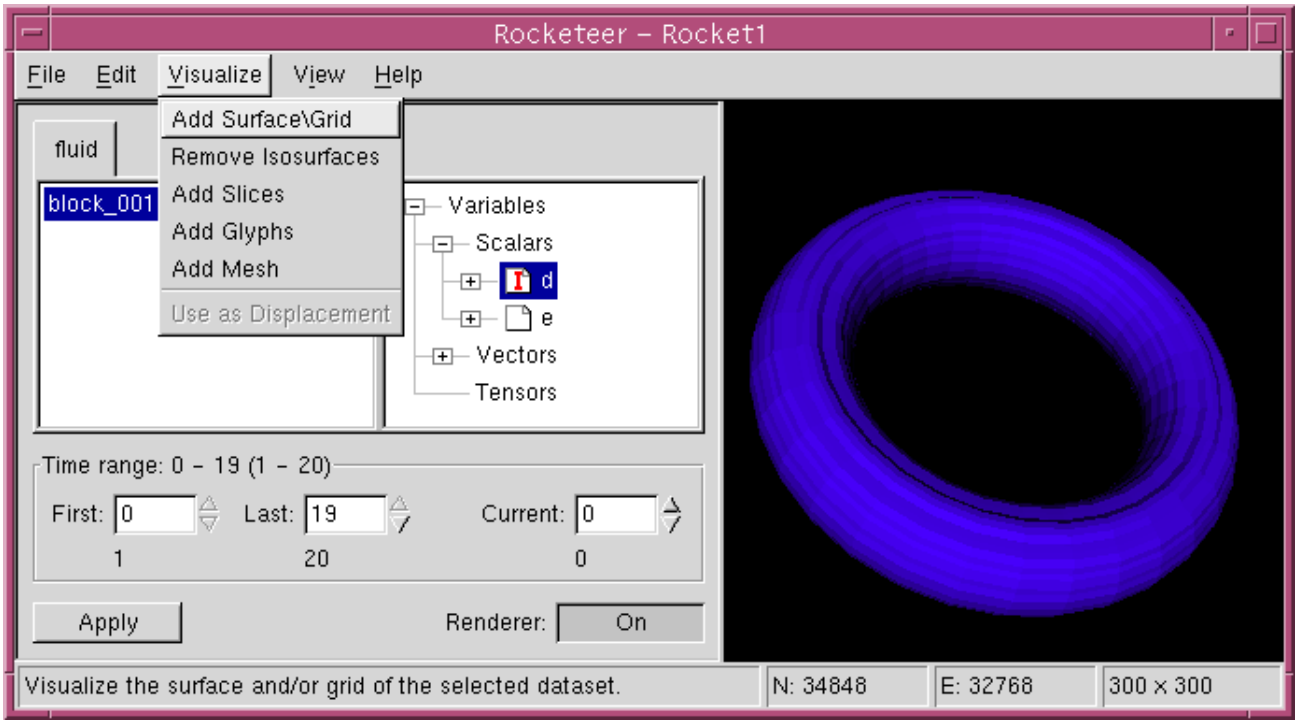
The above image clearly shows the contraction of the object along the x axis and its expansion along the y and z axes. You can make the surfaces translucent and/or show the grids in order to see the deformation of more complicated objects better.

6.5 Isosurface Plots

Download and open one of 20 snapshots from the ZEUS-MP expanding hot ring calculation. We will show how to produce an animation from this series later ins Section **??**.



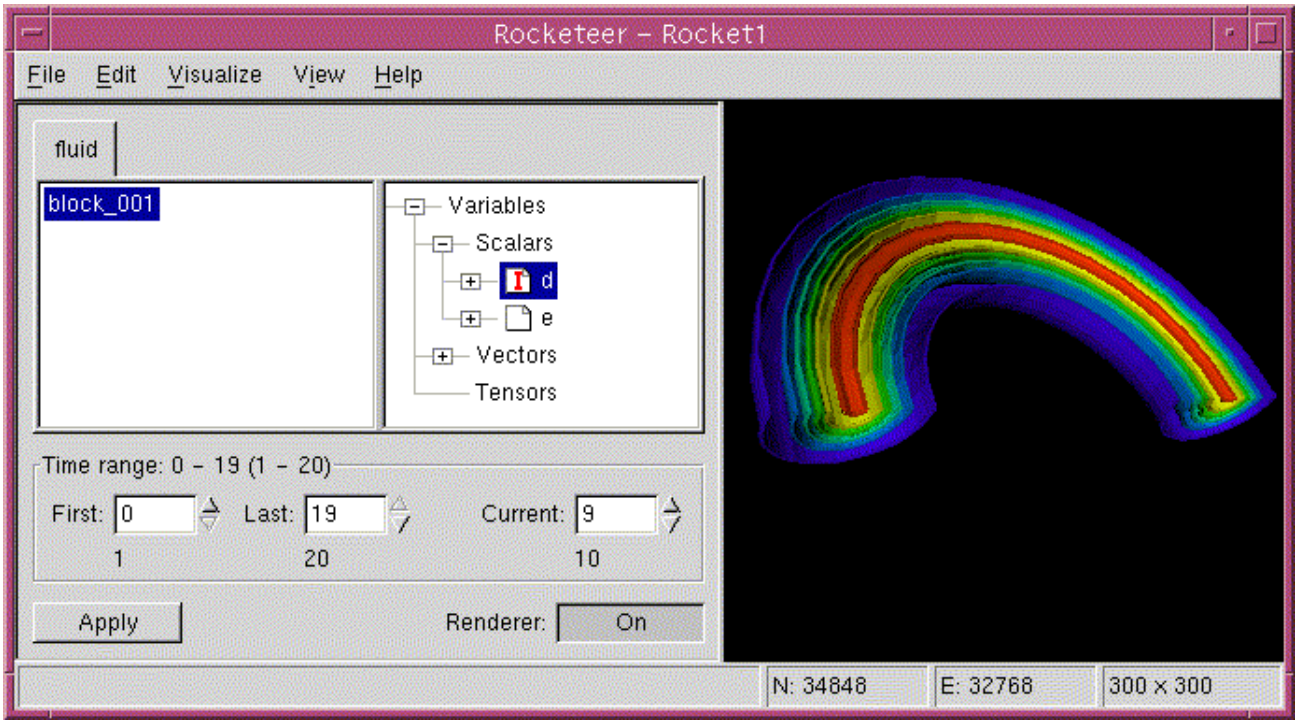
Select the scalar variable "d" and choose "Visualize/Add Isosurfaces". Turn on the Renderer.:



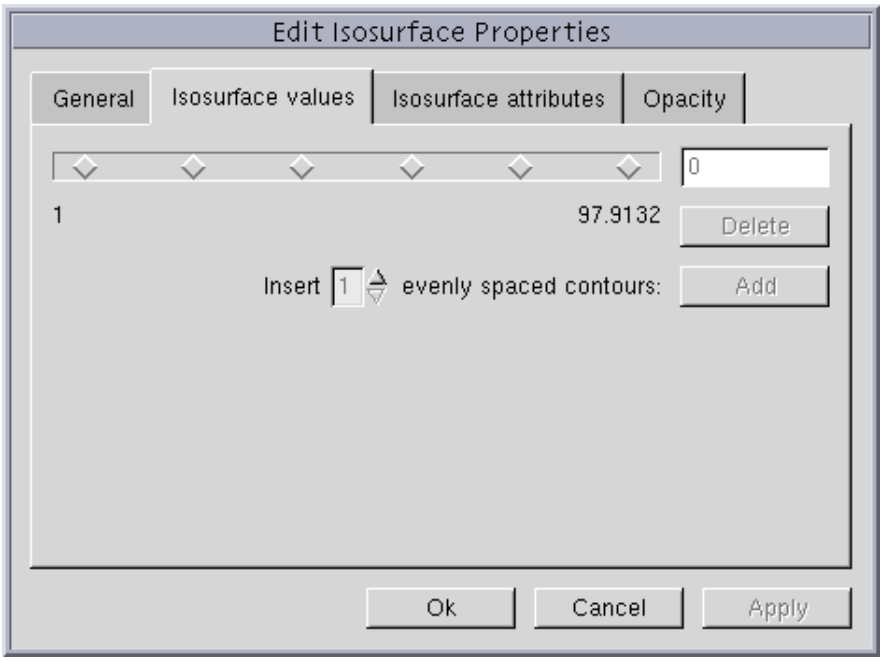
By default, a set of 7 evenly spaced opaque isosurfaces are displayed. If your graphics card renders the image correctly, you will see only the outer, blue isosurface. Edit the clipping planes as we have done above to cut the image in half at $y = 0$ and at $z = 0$ so that you can

see inside. To obtain the figure below, I have clicked on the up arrow next to "Current" to change the displayed time level to frame number 10.

Notes on time levels: If you click the down arrow from level 0 (first level), it will jump up to level 20 (last level). Typing a number in the "Current" box also works. Both an internal time index (which counts files opened, beginning at 0) and the physical time value stored in the HDF file are displayed for all times in this dialog box. You can type in an index and it shows you corresponding the physical time.

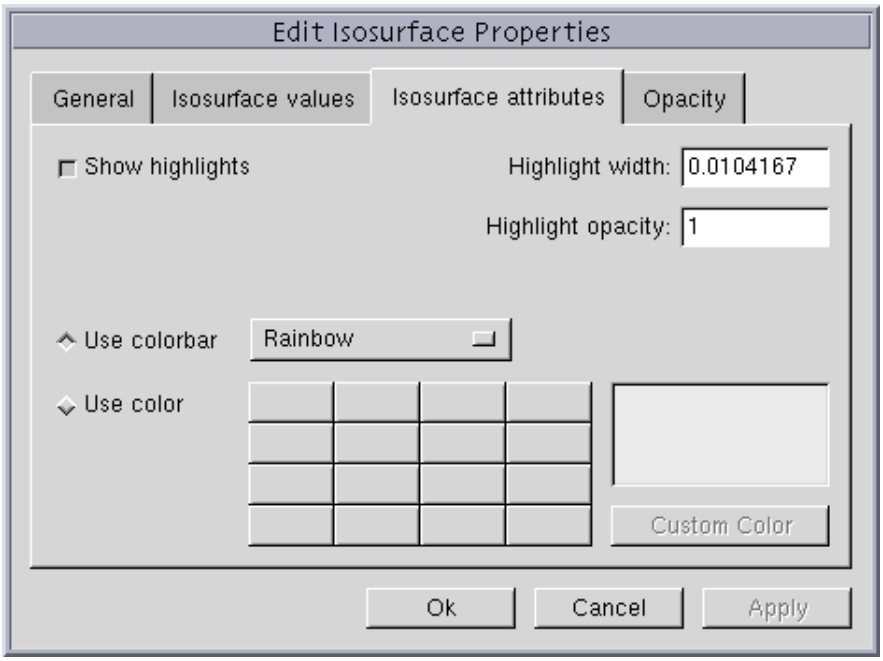


If you want to change the appearance or values of the isosurfaces, choose "Edit Isosurfaces" from the "Edit" menu. The options on the "General" and "Opacity" tabs are the same as they are on the "Edit Surface/Grid" dialog box (see above). The "Isosurface values" tab allows you to specify which isosurface values to use:



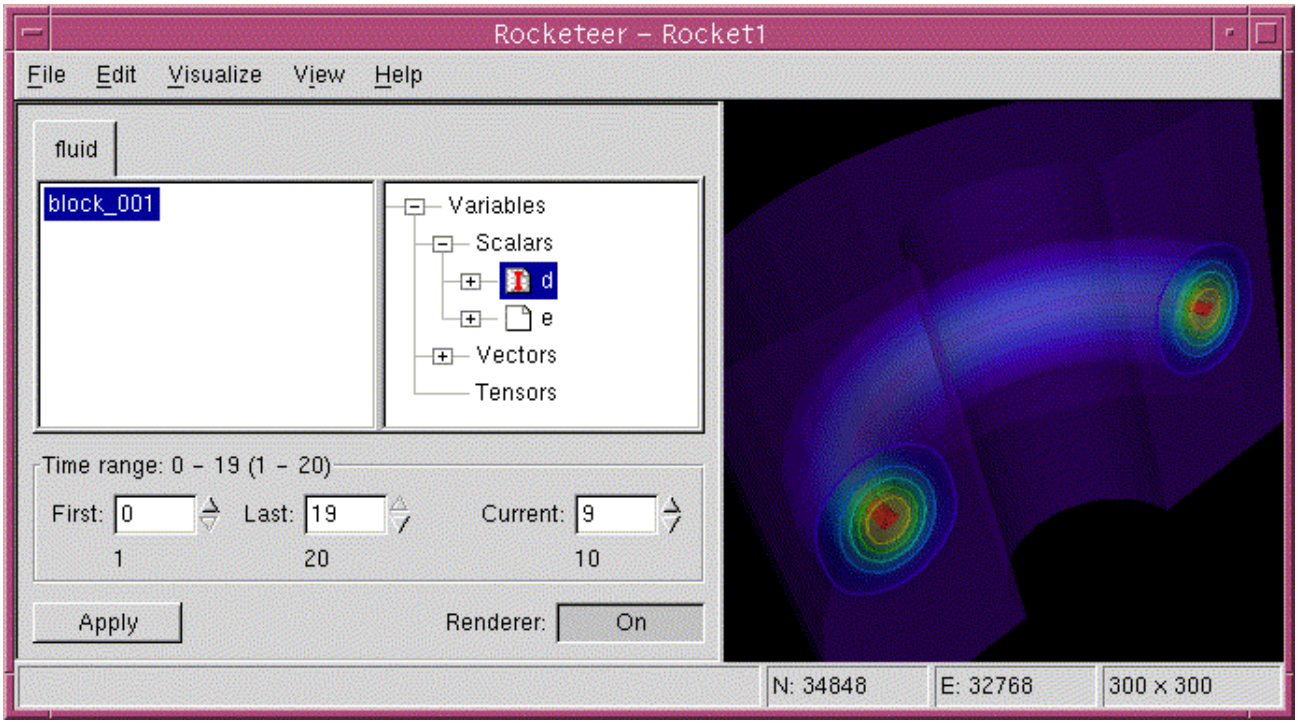
The bar with the diamonds lets you choose isosurfaces manually. You can change each isosurface level by clicking and dragging the corresponding diamond. For higher precision, you can click on a diamond to select and display the isosurface value in the white box, and then change it by typing in the desired value. You can delete individual isosurfaces by selecting them and clicking the "Delete" button. Finally, you can add a specified number of evenly spaced isosurfaces between two diamonds by selecting both of them, setting the desired number to add, and then clicking the "Add" button. To select two diamonds between which to add isosurfaces, click on the first one (it becomes red) and then shift-click on the second one (it becomes black).

The "Isosurface attributes" tab controls isosurface "Highlights", which are actually contours drawn on the edges of isosurfaces where they are cut by the clipping planes. We use these highlights to help make it easier to see where the isosurfaces end or have been cut. You can choose whether or not to use highlights at all via the "Show highlights" radio bottom, and you can set their width and opacity to your taste by changing values in the corresponding boxes.



This tab also allows you to select a color scale for the isosurfaces, or to make them all one color. A set of isosurfaces that are all the same color is probably not useful, but you can use "Edit/Clone Material" to create multiple copies of the variable you want to display, and then draw one isosurface for each clone in the color of your choice. A ight be better for this purpose.

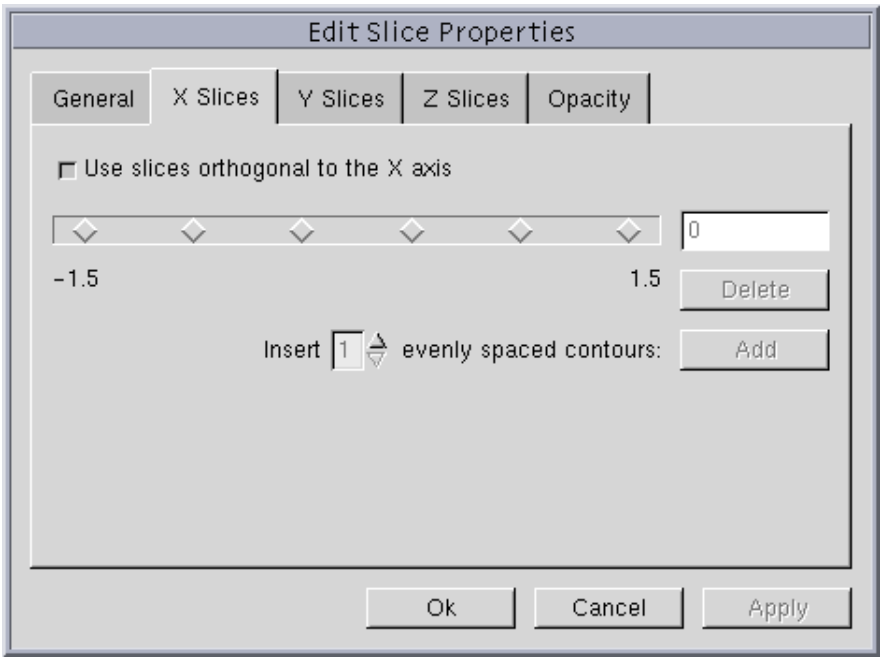
It is sometimes useful to combine a nearly transparent Surface plot with a nearly opaque Isosurface plot. This combination indicates the extent of the computational volume while allowing you to see the isosurfaces contained within, especially when you rotate the object back and forth and your graphics hardware displays everything correctly.



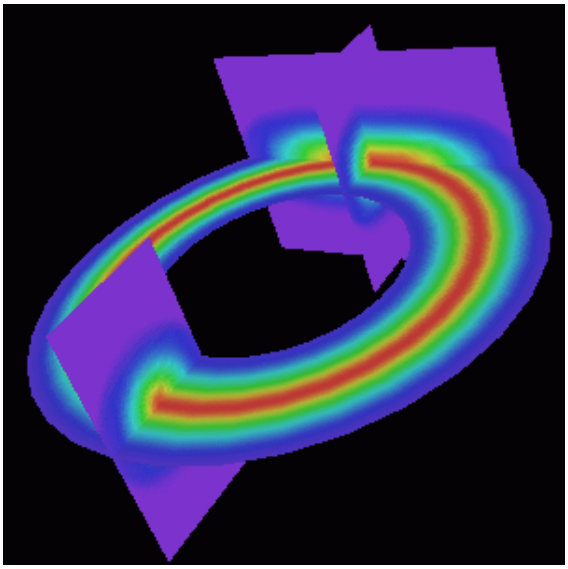
6.6 Slices

Using the same data set zmp_020.hdf as above, remove the isosurfaces and add slices ("Visualize" menu). By default, 7 evenly spaced slices across the z axis are displayed. You can control the slices using the "Edit/Edit Slices" dialog box. The "General" tablets you set the data range and the color scale.

On each "Slices" tab, there is a radio button to turn that set on or off. You can click and drag diamonds to move the corresponding slices or click on a diamond and type in the desired slice location. Right-clicking on a diamond removes the slice. You can also add a set of evenly spaced slices between two selected ones by specifying their number and clicking on "Add". Finally, you can change the opacity of the slices using the "Opacity" tab.



In the image below, one slice through z = 0, one slice through y = 0, and one slice through x = -1.5 are displayed. Clipping planes have been turned off here, but in some situations they can help you to see more of the slices.

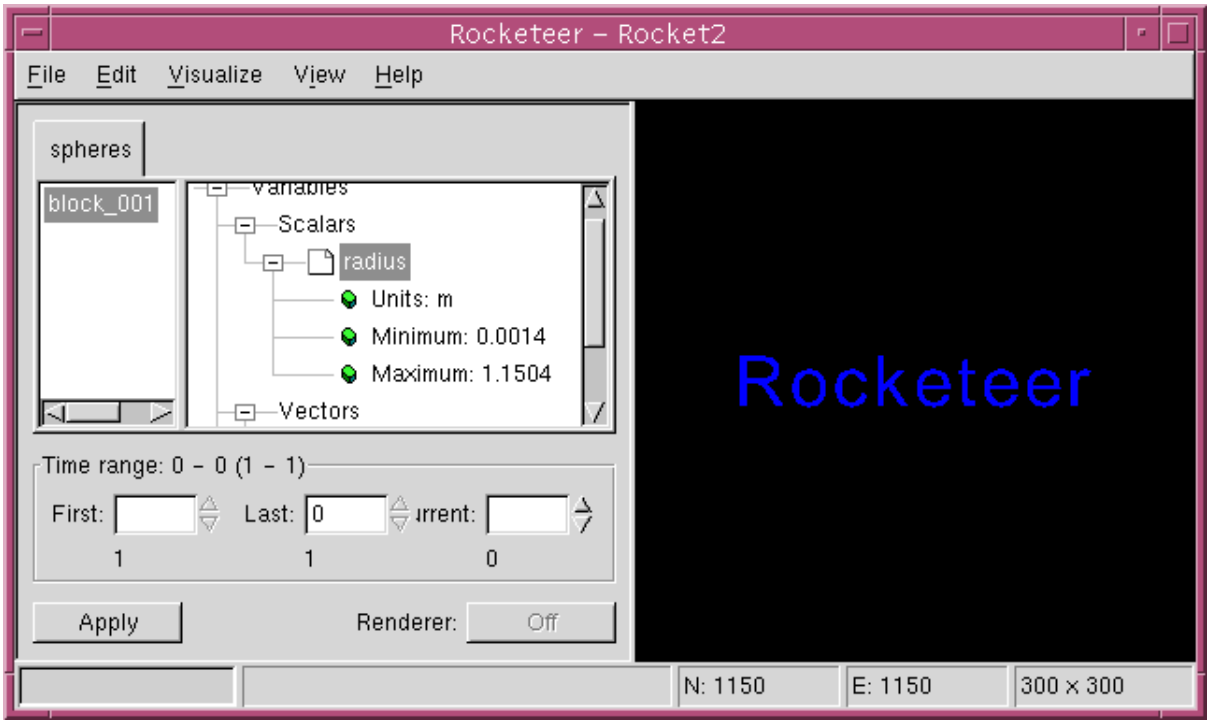


6.7 Glyphs

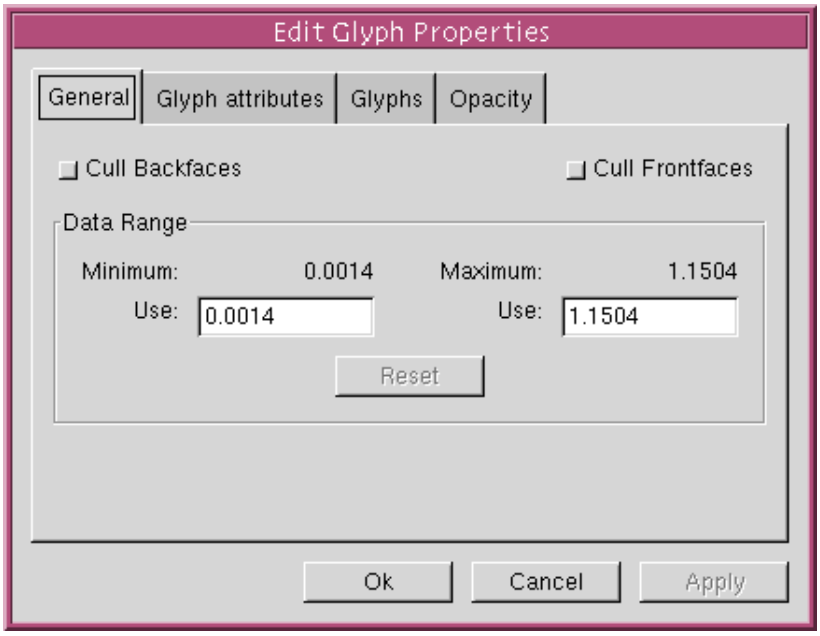
Rocketeer can display "point data", that is, data values given on a set of unconnected points, as well as on a grid. Various glyphs, including spheres for scalars and oriented cones for vectors, can be drawn at each point.

6.7.1 Scalars

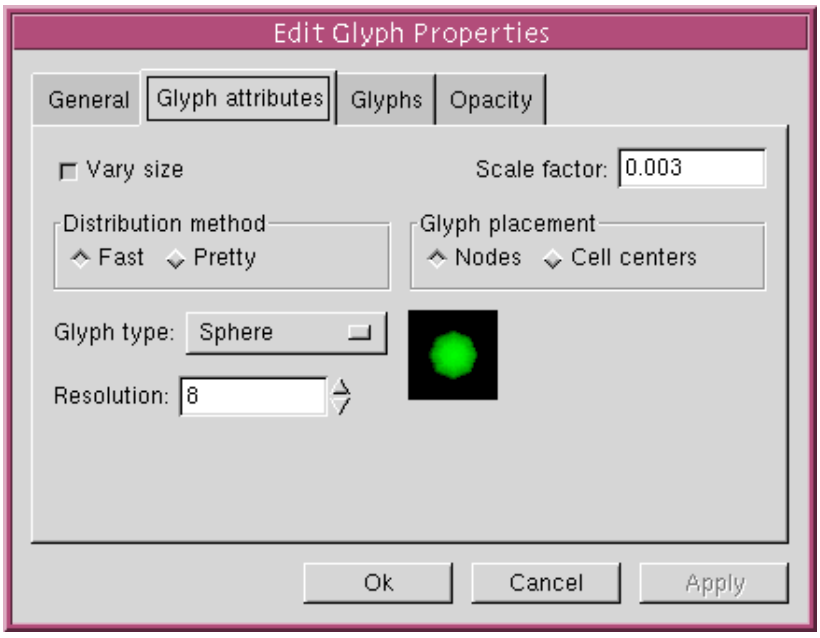
The scalar data values can be the radius of each sphere, or the data can be some other variable to be represented by the color of the spheres. To see a sphere example, download and open the HDF file spher1.hdf. You may also wish to download the Fortran source code and raw input data used to generate this HDF file.



There are 1150 spheres of three different radii in the file. Choose "Add Glyphs" from the "Visualize" menu, and then "Edit Glyphs" from the "Edit" menu (the max and min are incorrect here because this image was made from a different data set):



The General tab lets you set the range of radii to display. Limiting the range affects the color scale for the glyphs, but not which glyphs are displayed.



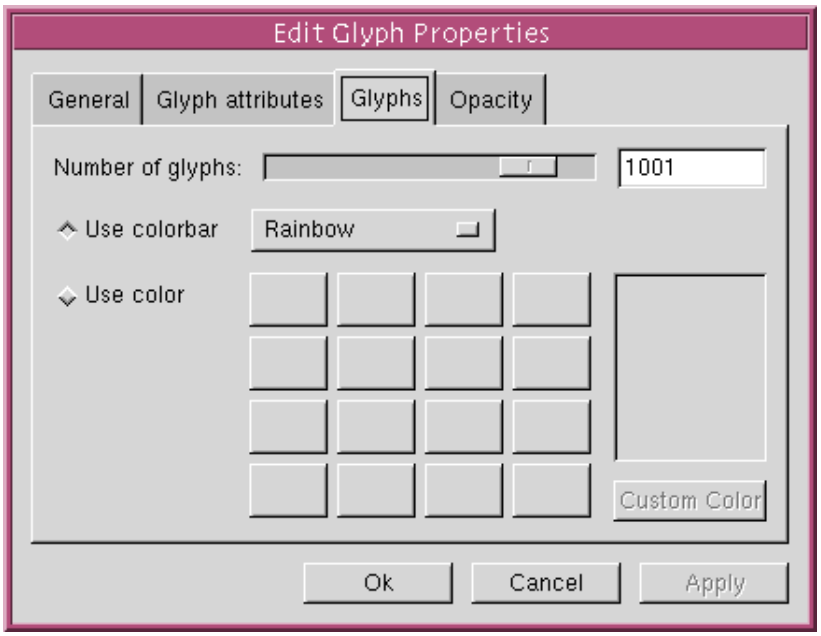
On the Glyph attributes tab you can select the size and type of glyphs to display. By default, for scalar variables the glyphs are spheres whose radii vary with the variable being plotted. You can change the glyph type by clicking the glyph type bar and selecting from the pop-up menu. The resolution determines the number of polygons that make up the glyph.

You can make all of the spheres the same size by unchecking "Vary size". This would be useful if the point data represents something other than the radius of the spheres, for

example, aluminum fraction. In this case, the color of the spheres indicates the aluminum fraction (the rest could be aluminum oxide). The scale factor normalizes the sizes of the glyphs. For some data sets, you may have to increase this factor to see any glyphs.

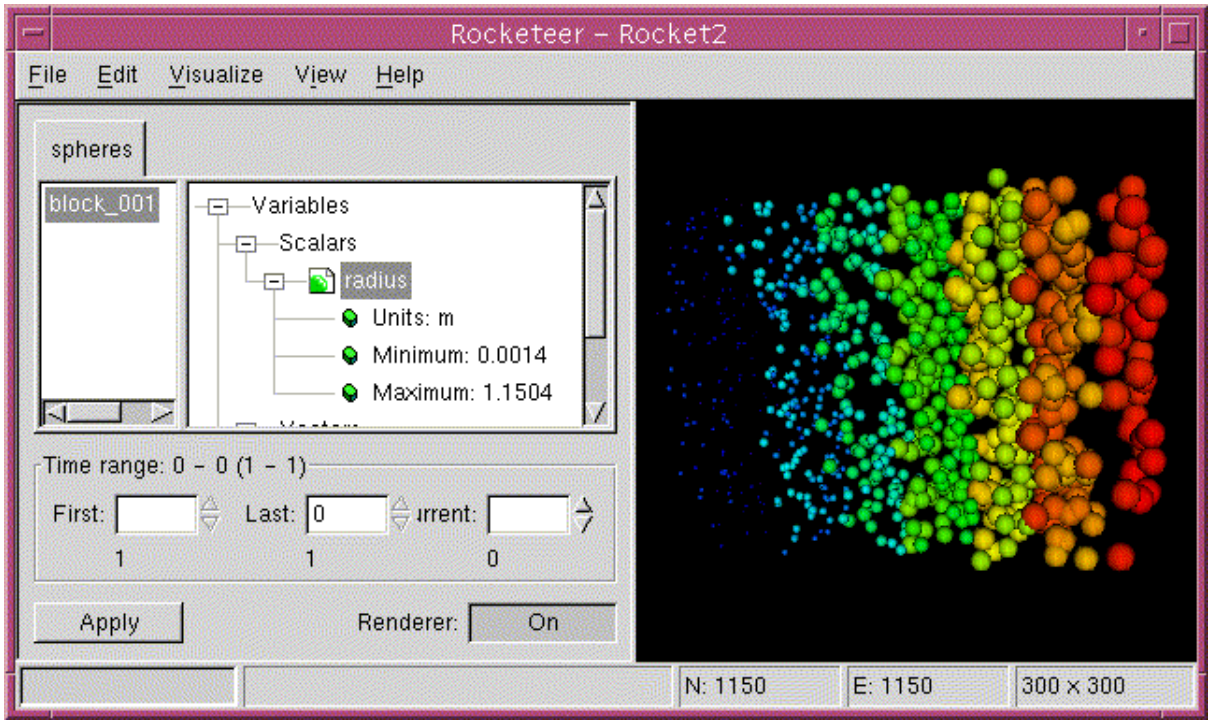
The individual spheres that appear in the image are chosen using one of two distribution methods. If the "Pretty" radio button is depressed, the data points are sorted into cubic bins in space and the glyph closest to the center of each bin is displayed. If instead the "Fast" radio button is selected, a random set of points is selected from the data set without regard to where they are located in space. This takes less time than the "Pretty" option, but make not look as nice. If you want to display all of the glyphs, be sure to select the "Fast" option. If you are making an animation, the "Fast" option may look better than "Pretty" if the points are moving because "Fast" selects the same set of points in each frame.

The placement of the glyphs (with respect to a mesh) can be at the nodes (points) associated with the glyphs, or at the centers of the elements formed by neighboring points.



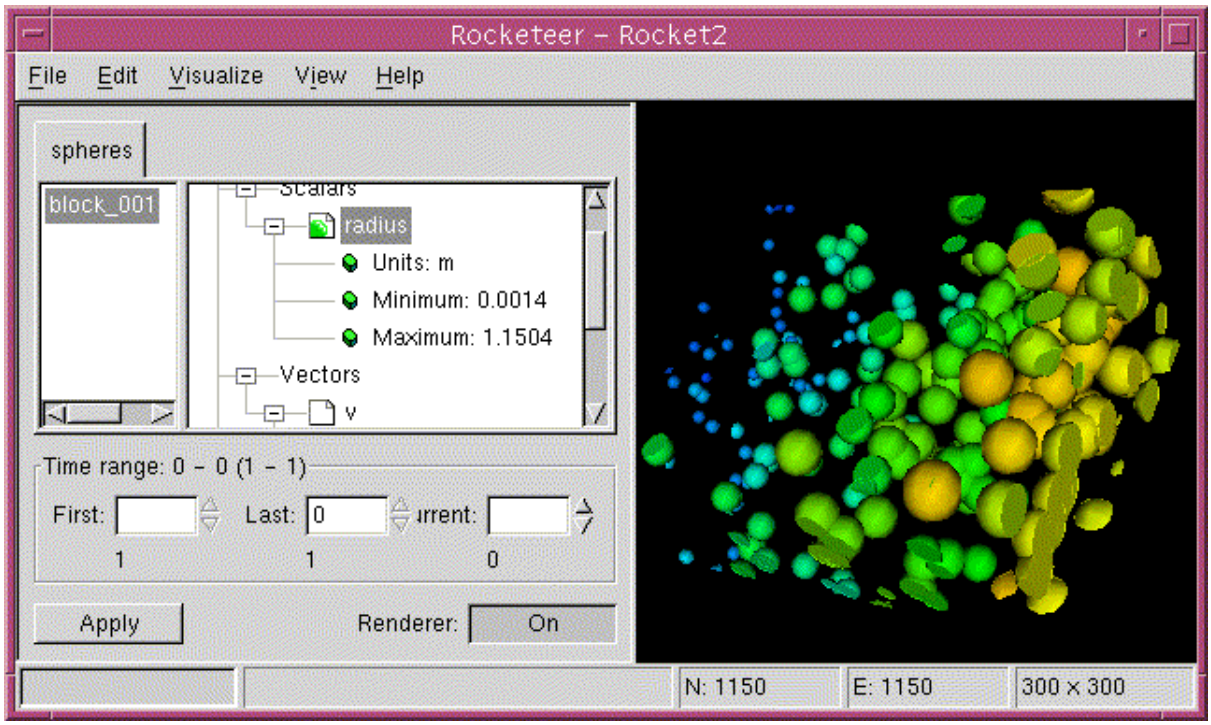
On the "Glyphs" tab, the number of spheres that appear can be adjusted using the slider. The best number of glyphs to use may require some experimentation. The spacing of glyphs under both the "fast" and "pretty" distribution methods can look much better after a relatively small change in the number of glyphs to be displayed.

The glyphs can be all one color (check "Use color") or a color scale can be used. In the image below, the radii vary and spheres of different sizes are displayed using the Rainbow color scale. Note that the scale factor was adjusted from its initial value to produce this image.



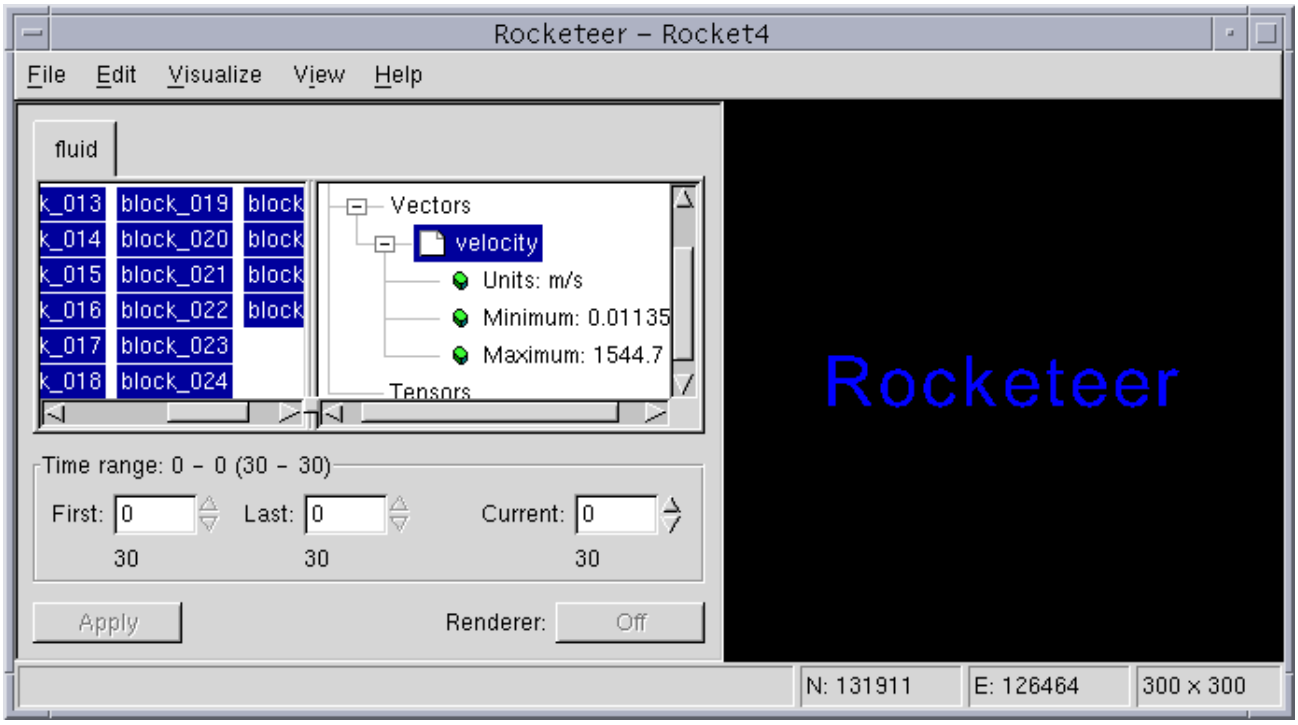
Finally, you can change the opacity of the glyphs on the "opacity" tab.

You can use clipping planes with glyphs. The image below includes 6 active clipping planes. The spheres appear solid because we draw disks where they are clipped:

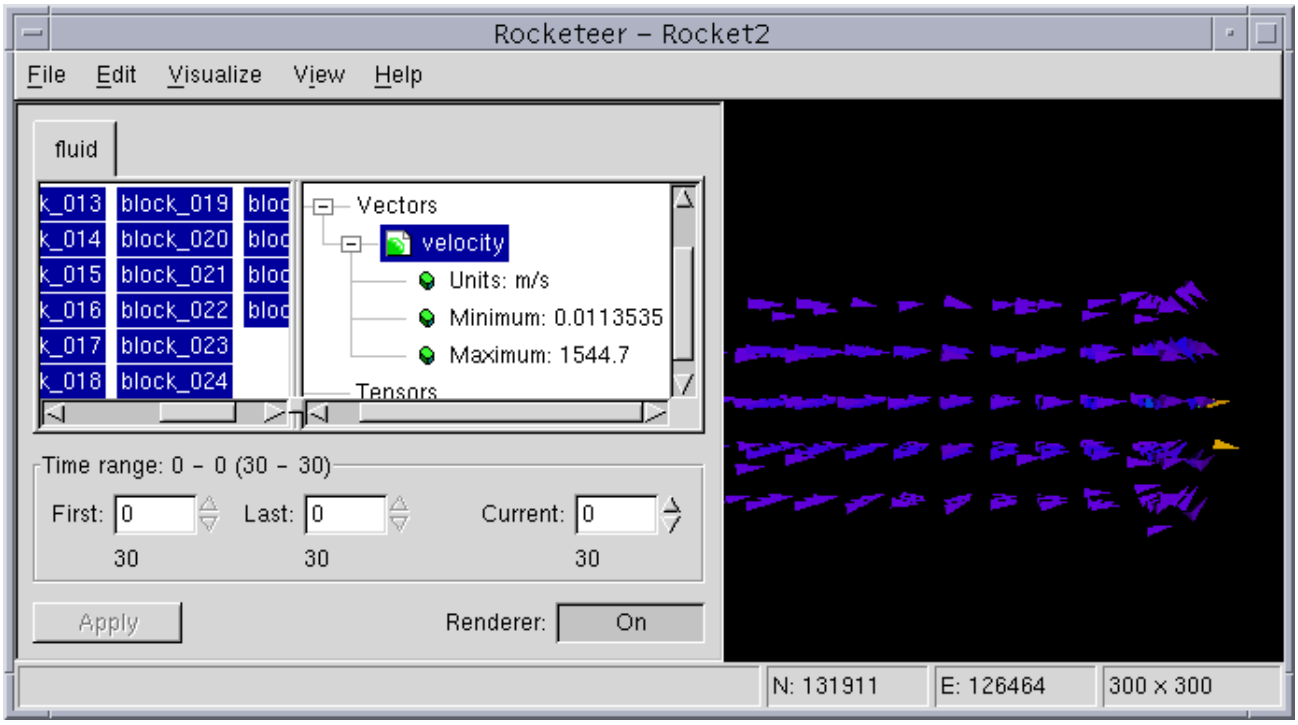


6.7.2 Vectors

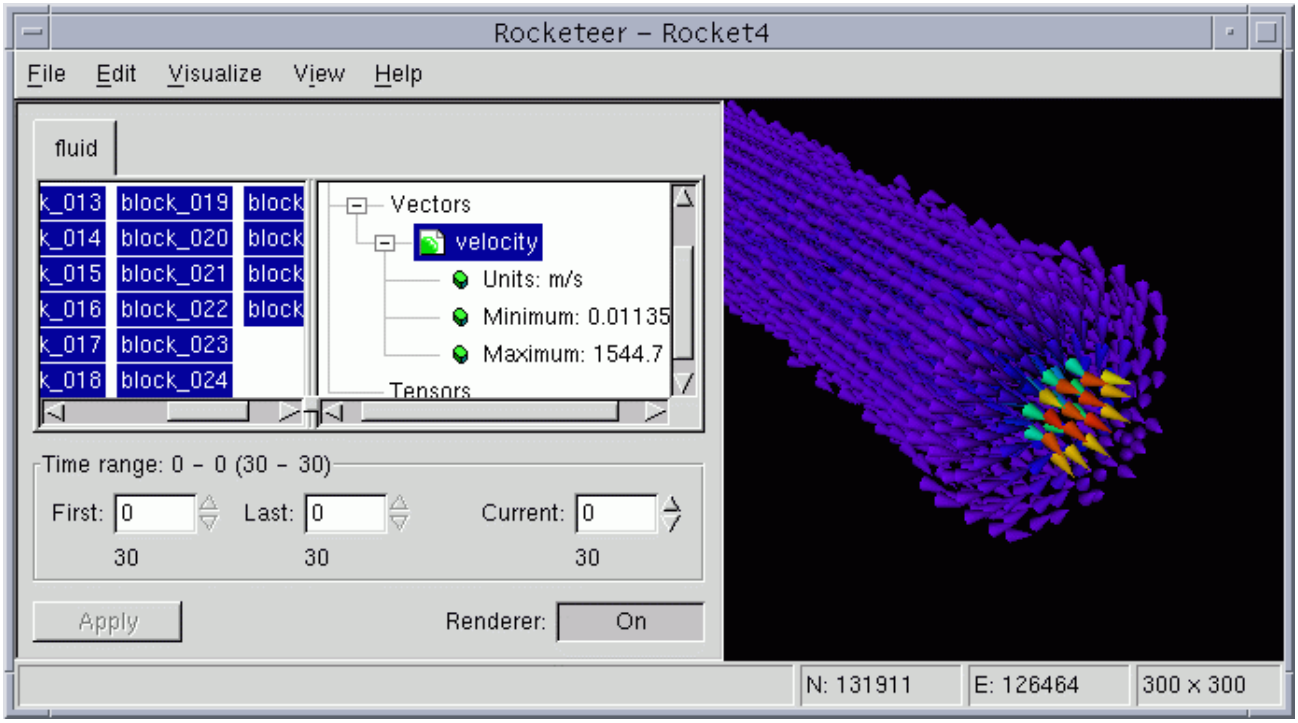
To see an example of how vectors can be displayed using glyphs, download fcyl13_030.hdf. This file contains 28 blocks from a rocket simulation, so to see the entire rocket you must select all of them and click "Apply".



Select the velocity in the Variables panel and select "Visualize/Add Glyphs". Choose "Edit/Edit Glyphs/Glyph attributes" and adjust the scale factor to 0.01:

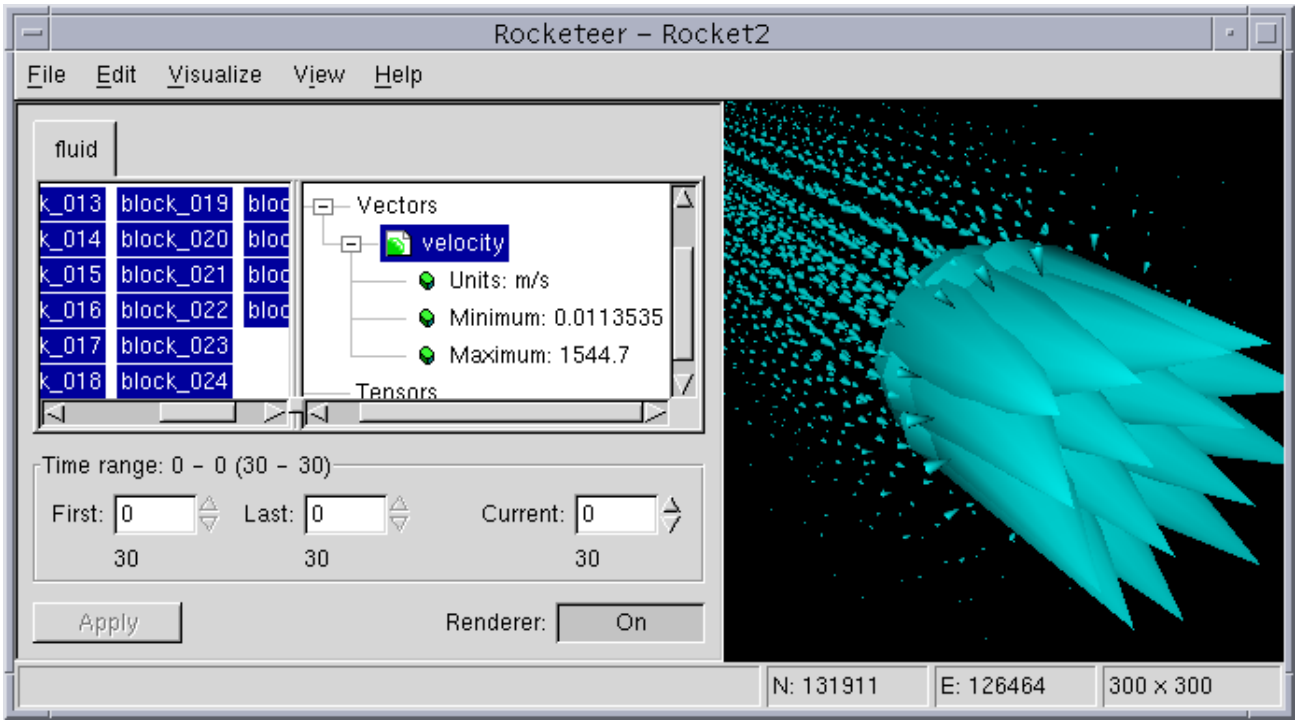


By default the glyphs are cones (3 sided), which for this data set are yellow (larger magnitude) only near the nozzle of the rocket. Again select "Edit Glyphs/Glyph Attributes" and change the resolution to 10 so that the cones look like smooth 3-D objects. Now go to the "Glyphs" tab and move the slider to about 21179 to show more cones.



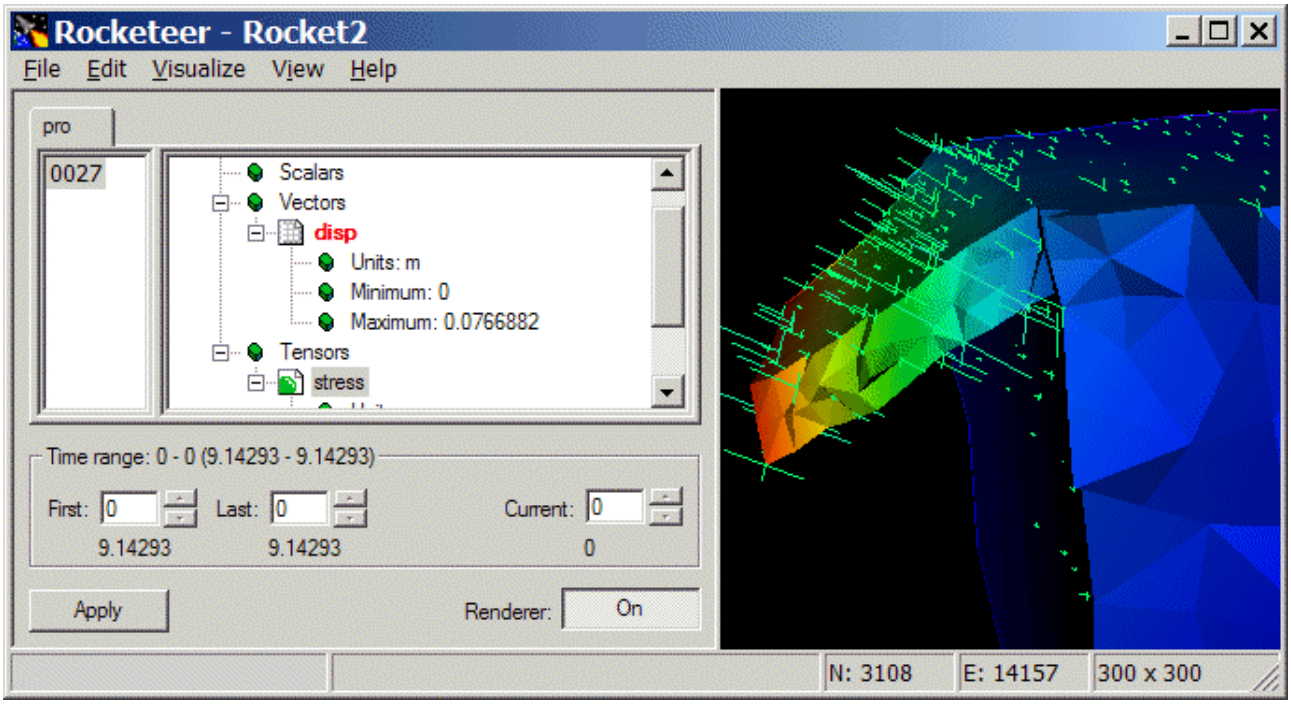
Here you can begin to see how the flow turns to escape through the nozzle. The image at the beginning of this User's Guide was made from this same data file plus the corresponding data file that contains the stress in the solid propellant. There you can see the flow near the head end of the rocket (there is a substantial gap between the top of the case and the propellant). The flow enters at the propellant surface and turns to flow up and down the axis of the rocket in the "chamber filling" phase of ignition.

On the "Edit/Edit Glyphs/Glyph Attributes" tab you can click the "Vary size" button to make the sizes of the cones correspond to the magnitudes of the vectors. Select "Cell centers" for the Glyph placement, since the velocity values are defined on cell centers in this data file. On the "Glyphs" tab you can choose to make the cones all one color. Choose as the scale factor 0.00004 and you have a more traditional (but less informative) display of the velocity field:



6.7.3 Tensors

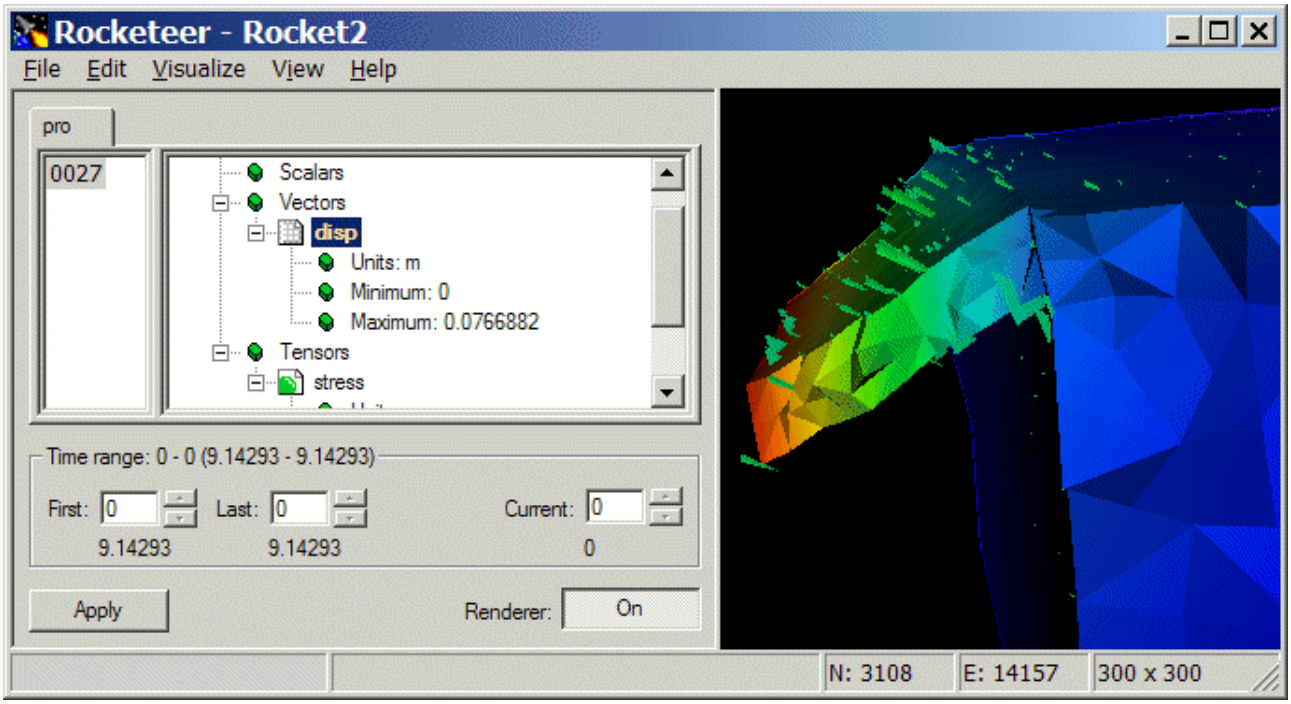
Download T_frac_09.142930_006.hdf, which contains the displacement vector and stress tensor for 1 partition in a simulation of a flexible inhibitor near a joint slot in the Space Shuttle booster. Select the tensor, choose "Visualize/Add Glyphs". Now choose "Edit/Edit Glyphs" and reduce the scale factor by a factor of 100 on the "Glyphs" tab. Add a surface plot of the displacement magnitude, and also choose "Visualize/Use as Displacement" to display the inhibitor in the deformed coordinate system.



Unlike the other examples in this User's Guide, the figure above was saved from the MS Windows version of Rocketeer, so the GUI has a somewhat different style.

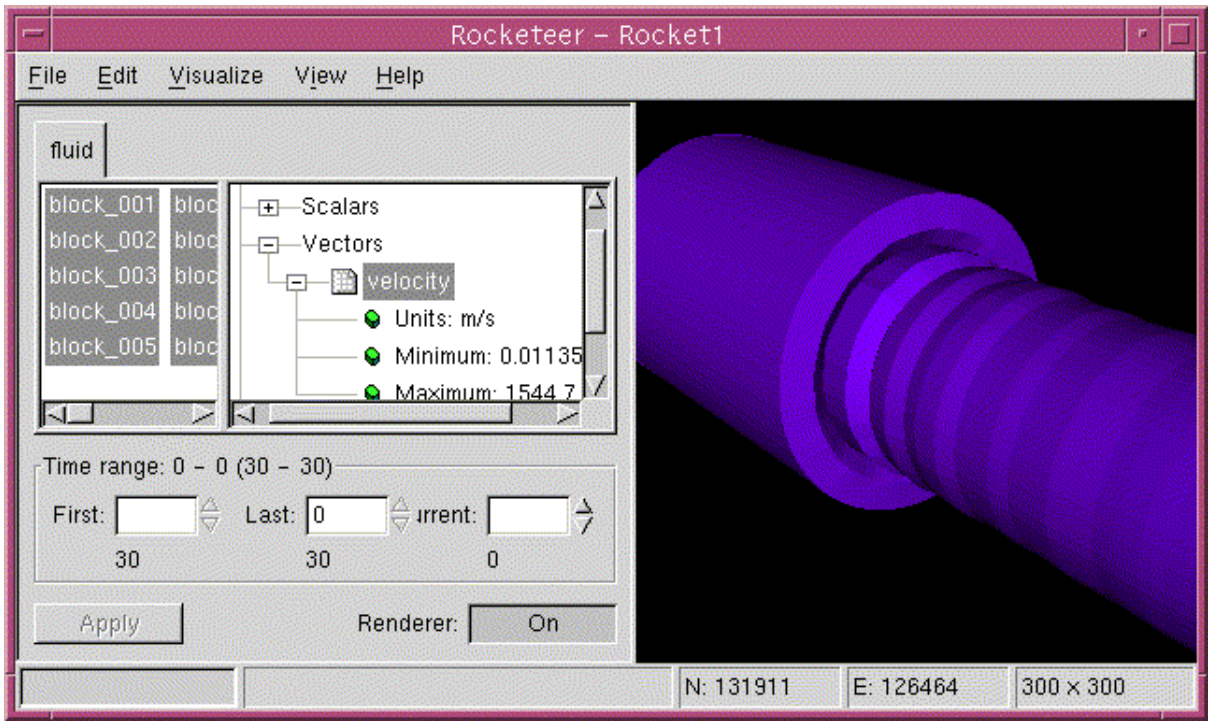
The stress tensor is represented by finding the 3 eigenvalues and eigenvectors, and drawing line segments oriented along the 3 eigenvectors. The line segments are scaled by the corresponding eigenvalues and colored by the trace of the tensor. In the image above, the red portion is at the free end of the inhibitor, which has been bent downward by a strong flow of gas headed toward the nozzle.

The "sphere" or the "cone" glyph may also be used for a tensor variable instead of the "axes" glyph. For spheres, it draws ellipsoids whose major and minor axes are oriented along the eigenvectors and scaled by the eigenvalues. The direction the "cone" glyphs point is along the eigenvector with the largest eigenvalue; the 3 eigenvalues determine the shape of the cones:

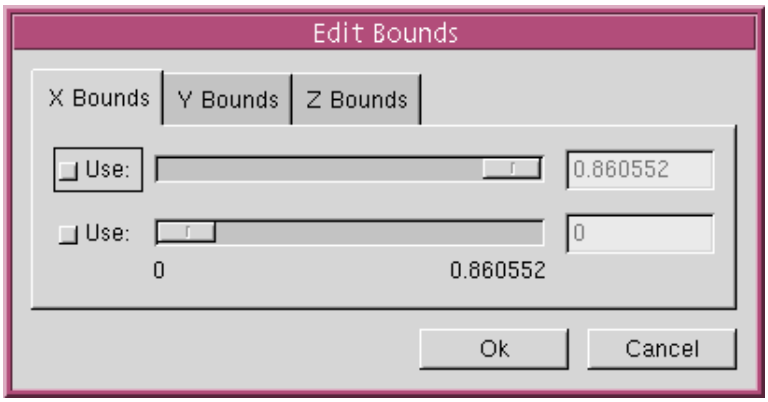


6.8 Selecting Blocks by Bounding Box

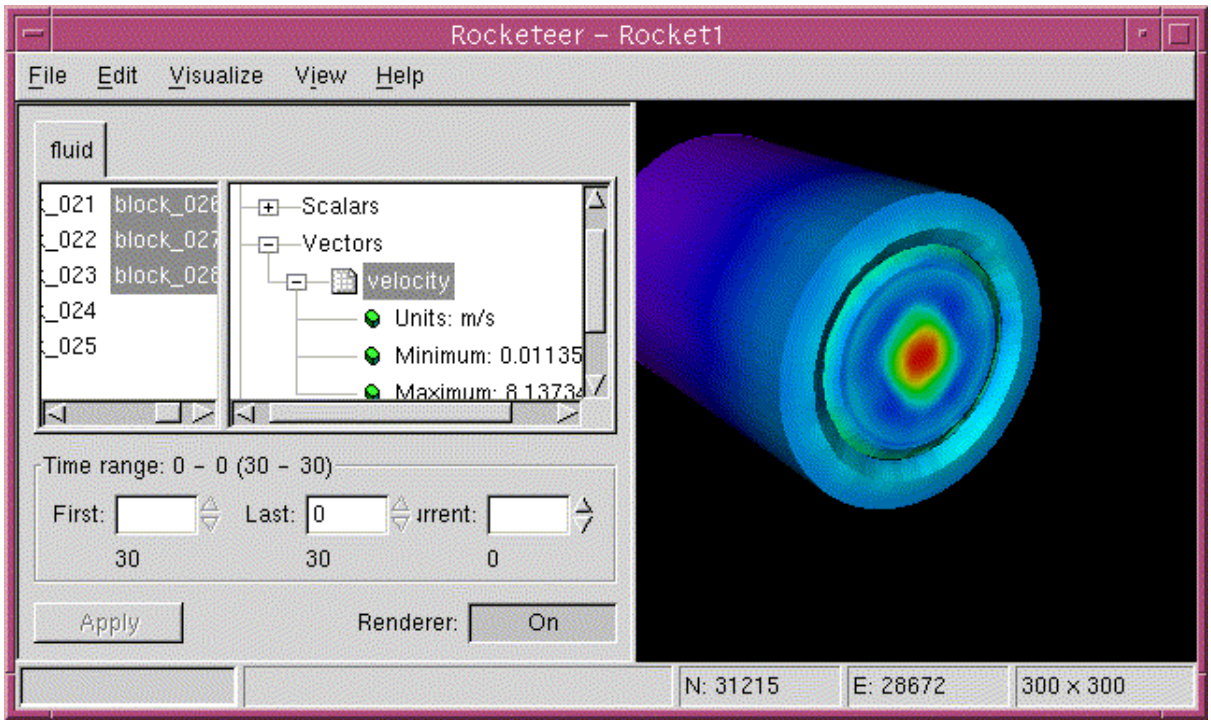
You can select blocks for visualization by specifying the x, y, and/or z coordinate ranges of a bounding box. The image below shows the magnitude of the velocity as a surface plot for all 28 blocks in the rocket data from file fcyl13_030.hdf:



After choosing "Edit/Select Blocks ...", a dialog box appears that is similar to the one for editing the clipping planes.



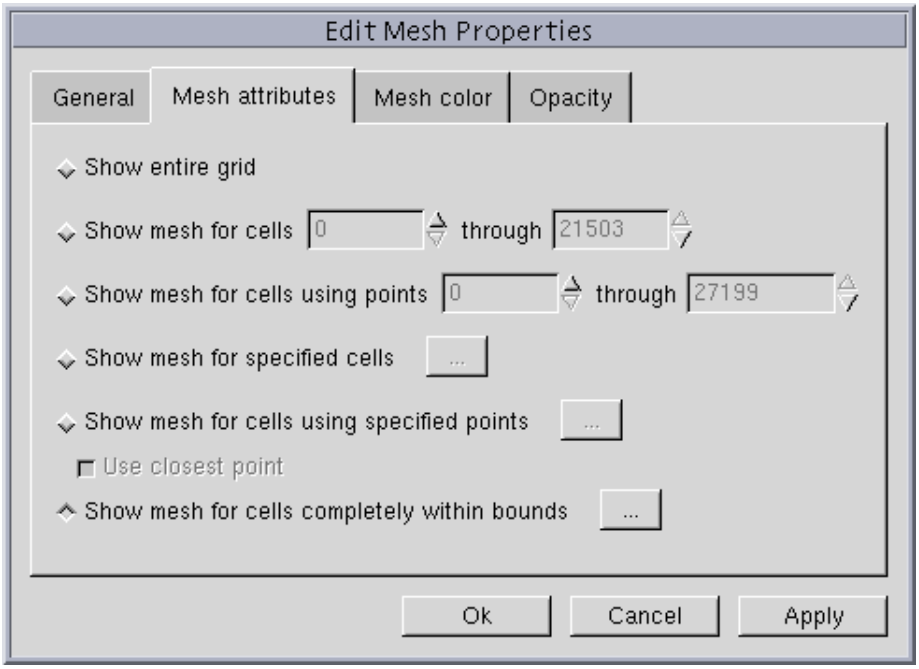
In this dialog box, you can click the radio buttons to activate upper and/or lower x, y, and/or z bounds, and use the sliders or type in numbers to specify the bounding box. When you click "OK", only the blocks that touch the specified bounding box will be selected. Click "Apply" to visualize only the selected blocks. In the image below, we set the upper bound on X to 0.1 (meters) to see just the head end, which is about 0.15 m long. Only blocks 26, 27, and 28 now appear.



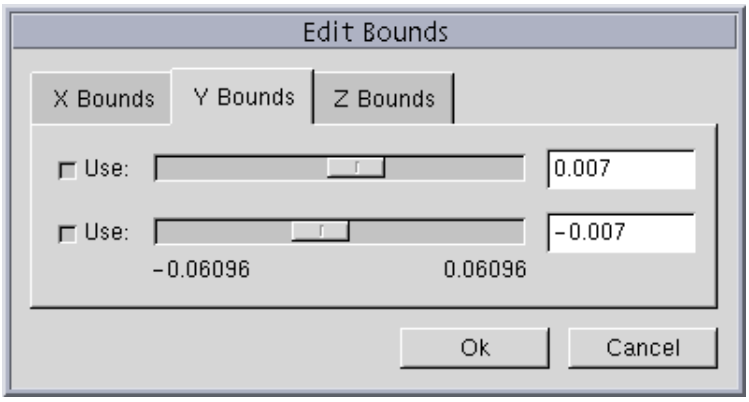
6.9 Volume Meshes

Open the file rsol_027.hdf, select all 16 blocks and click on "Apply". Now select the displacement vector and choose "Visualize/Use as Displacement". Select the pressure and "Visualize/Add Surface". Use the "Edit/Edit Surface/Opacity" tab to specify an opacity of about 0.4. Turn on the renderer.

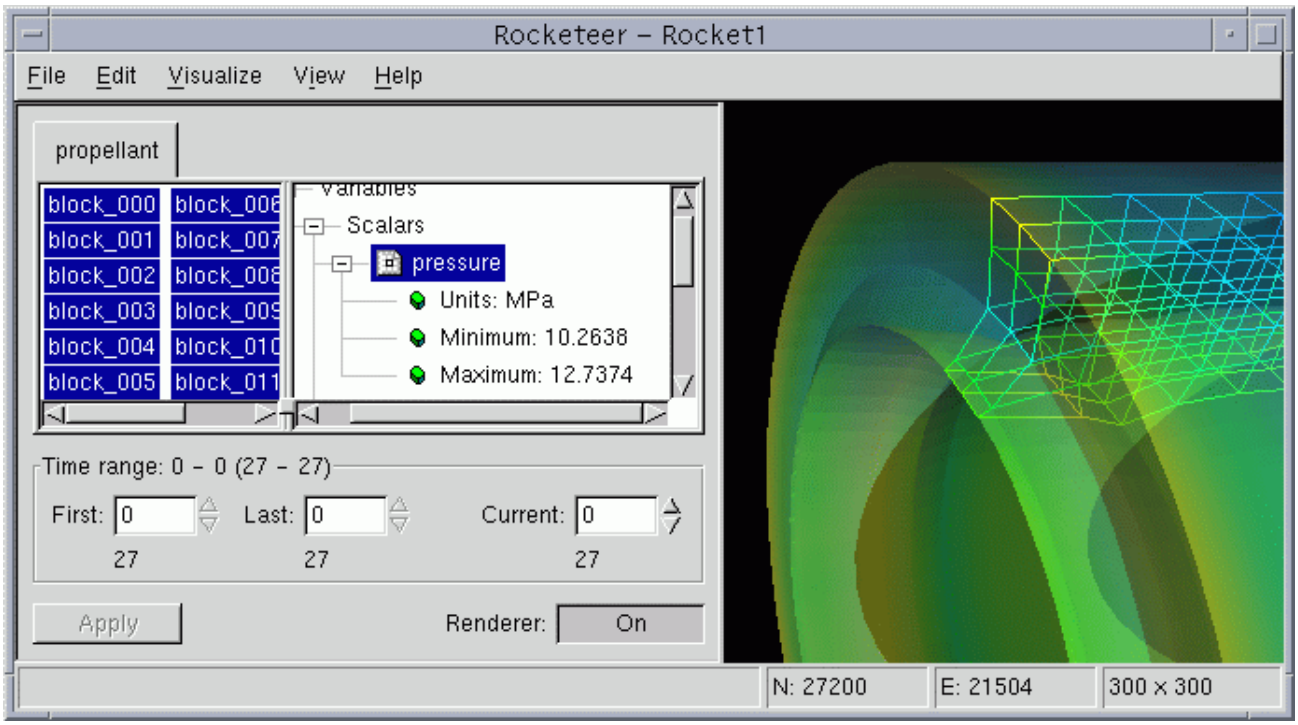
To add the 3-D interior mesh choose "Visualize/Add Mesh", and then "Edit/Edit Mesh". By default, the entire mesh is displayed. To see a portion of the mesh more clearly, select the "Mesh Attributes" tab:



This dialog gives numerous options for specifying which part of the 3-D mesh should be visible. Click on the corresponding radio button, and you can enter the index of one or more nodes or elements, or specify the coordinates of one or more points, and the mesh in the immediate vicinity will be displayed. You can also click the button for "Show mesh for cells completely within bounds" and then on the corresponding rectangular button marked "..." to get the "Edit Bounds" dialog box:



For this data set, click both "Use" radio buttons on the "Y Bounds" tab and set the bounds from -0.007 to +0.007. Two layers of elements lie in this range:

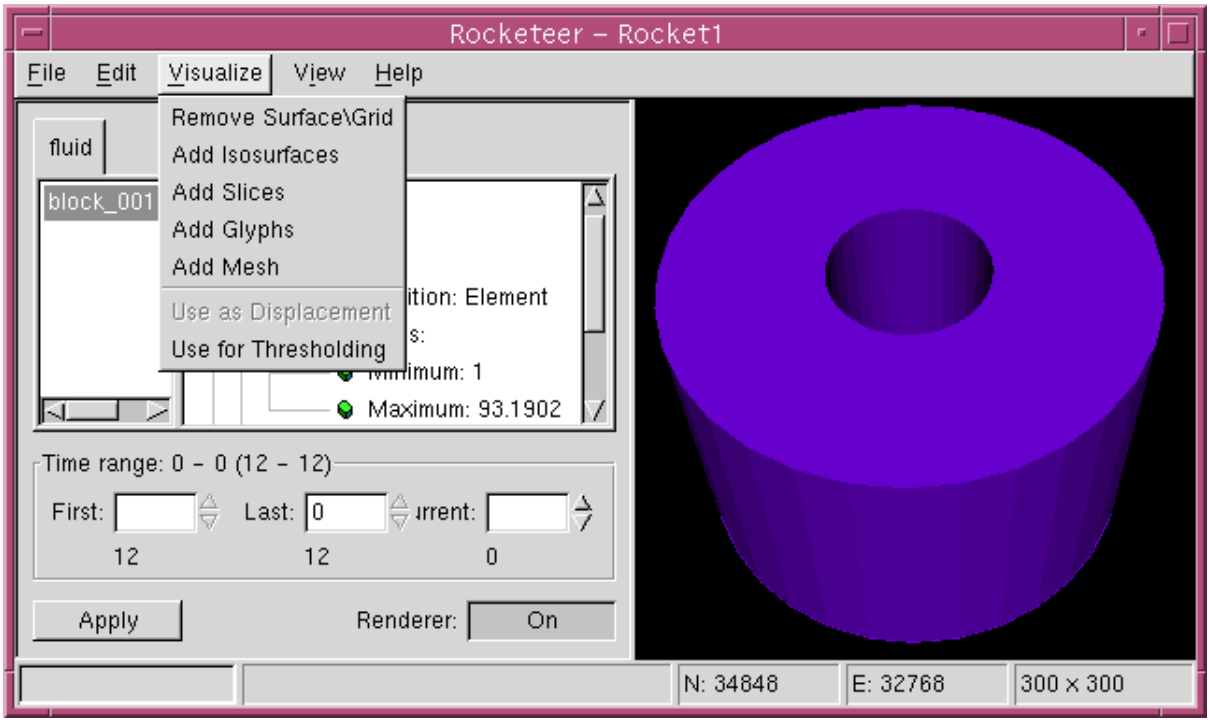


This mesh is very skewed near the inner radius at the head end of the cylinder.

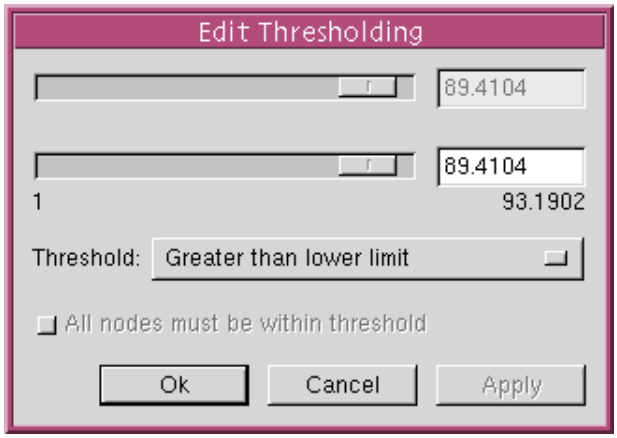
You can use any variable to color the mesh, and you can specify the color scale or a single color on the "Mesh Color" tab. You can control the range of the variable on the "General" tab, and the opacity of the mesh lines on the "Opacity" tab.

6.10 Displaying Elements Satisfying a Threshold

Rocketeer can display only those elements which meet user specified criteria based on the values of a scalar variable. Returning to the HDF file zmp_012.hdf, select the scalar variable "d", and choose "Use for Thresholding" from the "Visualize" menu:



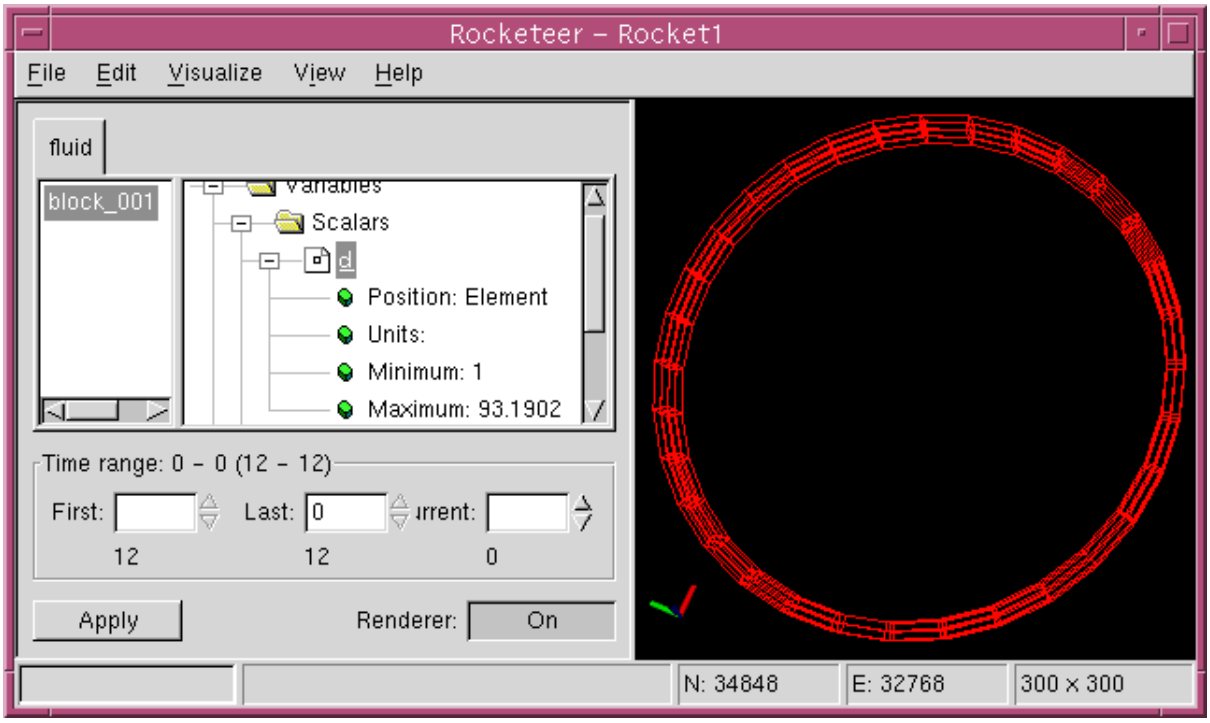
The "Edit Thresholding" dialog box that pops up allows you to specify which elements to display.



In this example, we moved the lower slide bar to have it display only the elements for which "d" $>$ 89.4104. You can also pick a lower limit, or choose a range of values and display elements for which "d" is inside or outside that range.

Thresholding works with scalars defined at either element centers or node centers. For node-centered scalars, you have a choice of requiring just one nodal value to satisfy the threshold, or of requiring all nodes in the element to satisfy it (radio button on).

Here, suppose we would like to see the mesh for the largest values of "d". Using the "Visualize" menu, remove the surface plot and select "Add Mesh":

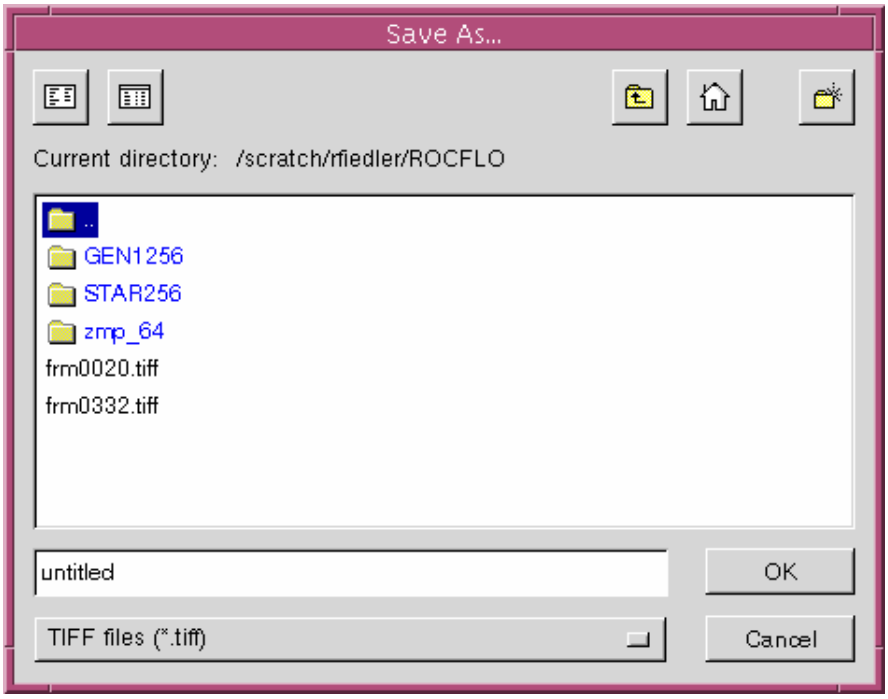


In the figure above, the elements where "d" is largest comprise a ring in the center of the domain. The mesh is colored by the value of "d", as usual.

You can change the threshold later by selecting "Edit/Edit Threshold...", which brings up the "Edit Thresholding" dialog box shown above.

6.11 Saving Images

You can save the image in several formats including TIFF, JPEG, and bitmap. Use "Save Image As" in the File menu:



You can create a new directory by clicking on the new folder icon in the upper right corner of this dialog box.

You may wish to save images using "Make Movie" (see below) instead of "Save Image As" because "Make Movie" by default uses software rendering, which can eliminate defects in the image that may be introduced by your graphics hardware and/or drivers.

6.12 Saving and Recovering Sessions

You can save the current state of a Rocketeer window by selecting "Save" or "Save As" on the "File" menu. The default action is to create a file in the current directory named Rocket n.roc, where n is the Rocketeer window number. You can choose the name and location in subsequent dialog boxes.

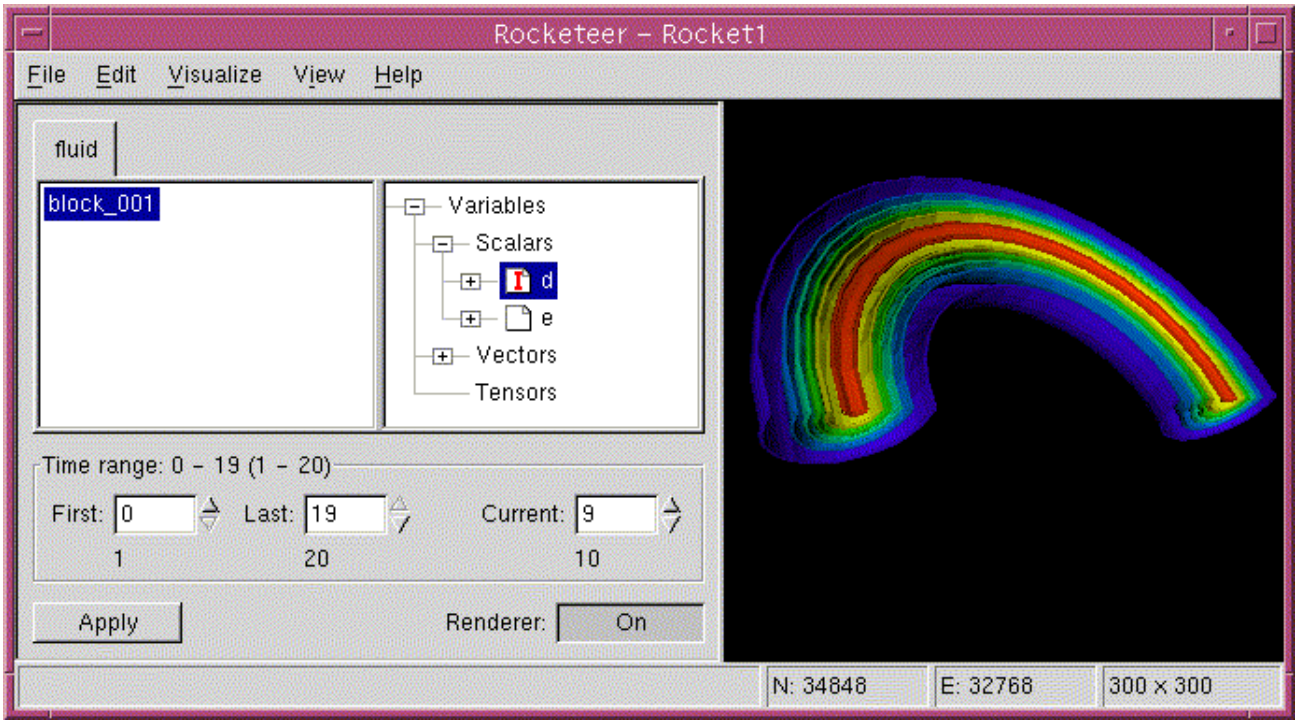
The `*.roc` files are text files describing the graphical operations performed, the camera position, and the list of opened HDF files. These can be edited by hand if desired, e.g., to change the HDF file names.

All three tools in the Rocketeer suite can process `*.roc` files. For the interactive tools, `*.roc` files provide a means of recovering a session or of creating a window like an existing one but with different data. For the batch mode tool, they serve as an input file which specifies the graphical operations to be performed on a set of HDF files (typically a series of snapshots from a simulation). More details on using the batch mode tool are given on the Voyager web page.

6.13 Animating a Series of Files

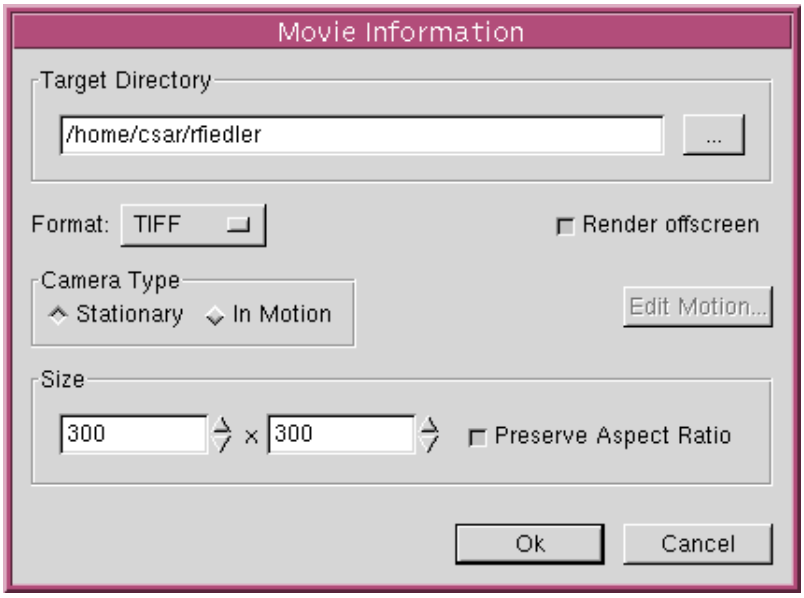
If you have a series of output dumps from a simulation, whether they are stored in separate files or in a single file, Rocketeer can process all of the dumps automatically to make frames for an animation by applying the same set of graphics operations using the same color scale and camera position.

Download the series of files from the ZEUS-MP expanding ring calculation. We used the first file above while discussing Choose "Select Data Files" from the "File" menu and select all 21 zmp_nnn.hdf files, click on the "d" variable and select "Visualize/Add isosurfaces", and then add an x = 0 and a z= 0 cutting plane. Here is the image for the data in zmp_010.hdf:



At this point you may want to change the current frame to look at more of the images that will comprise your animation. Hit the up or down arrows to change frames. You may also want to save the camera position ("Edit" menu). This will create a file (called "rocket.cam" by default) that can be loaded in another Rocketeer session ("Edit/Load camera position") to restore the orientation of your image.

To generate frames for animation, choose "Make Movie" from the "File" menu. You will see a dialog box which allows you to choose a folder, data format, and size (in pixels) for the frames it will save:



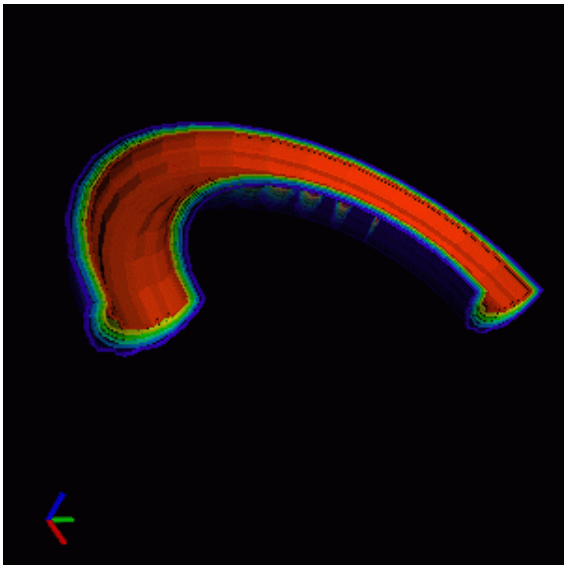
By default, the frames will be rendered off screen. This means it will not use your graphics hardware, so rendering the images will be slower than if they were rendered on screen. However, when processing a series of HDF files, most of the time is spent reading the files. Therefore, there is not usually a large penalty for rendering everything in software. On systems other than MS Windows, Rockeeteer will crash unless you render the images on screen (uncheck the radio button, unless it is grayed-out already). You may have to click the "In Motion" button and then the "Stationary" button to get the "OK" button to become active. When images are rendered on-screen, you will see each image as it is generated. Rocketeer will save whatever is displayed in the image window, so be sure it remains unobstructed during movie making. This means you should not try to do other work on your desktop, and you should turn off any screen savers before generating a lengthy series of frames.

Once you have generated the frames, you can animate them using an application such as JASC Software's Animation Shop 3 (Included with Paint Shop Pro 7) for MS Windows or ImageMagick for most platforms. Animation Shop 3 produces much smaller movie files than ImageMagick (because the lzw compression is disabled unless you recompile it yourself and enable compression), and the clarity is nearly as good. QuickTime Pro is also good for producing Aminations on Windows or the MacIntosh.

The frames are named "frm0000.tiff, frm0001.tiff, etc. To convert them into a GIF animation file using ImageMagick, type:

```
<path>convert -delay <n> -compression lzw -loop -1 frm*.tiff animation
```

where `<path>` is the path to the ImageMagick routines on your system and <n> is the number of hundredths of a second delay between each frame (n = 5 gives 20 frames per second, which may be too fast; try 10). The "-compression lzw" is optional, and the "-loop -1" causes the animation to play over and over. See the "man" page on "convert" for more information.



Click on the image above to see the animation. This file runs at 10 frames per second.

For clearer movies, generate frames 2 to 4 times larger in each direction than the desired size of the movie. For example, generate images that are 800 × 800 to 1600 × 1600 to make movies that are 400 × 400. To shrink the frames as they are made into a movie with ImageMagick, simply use the geometry option. The command:

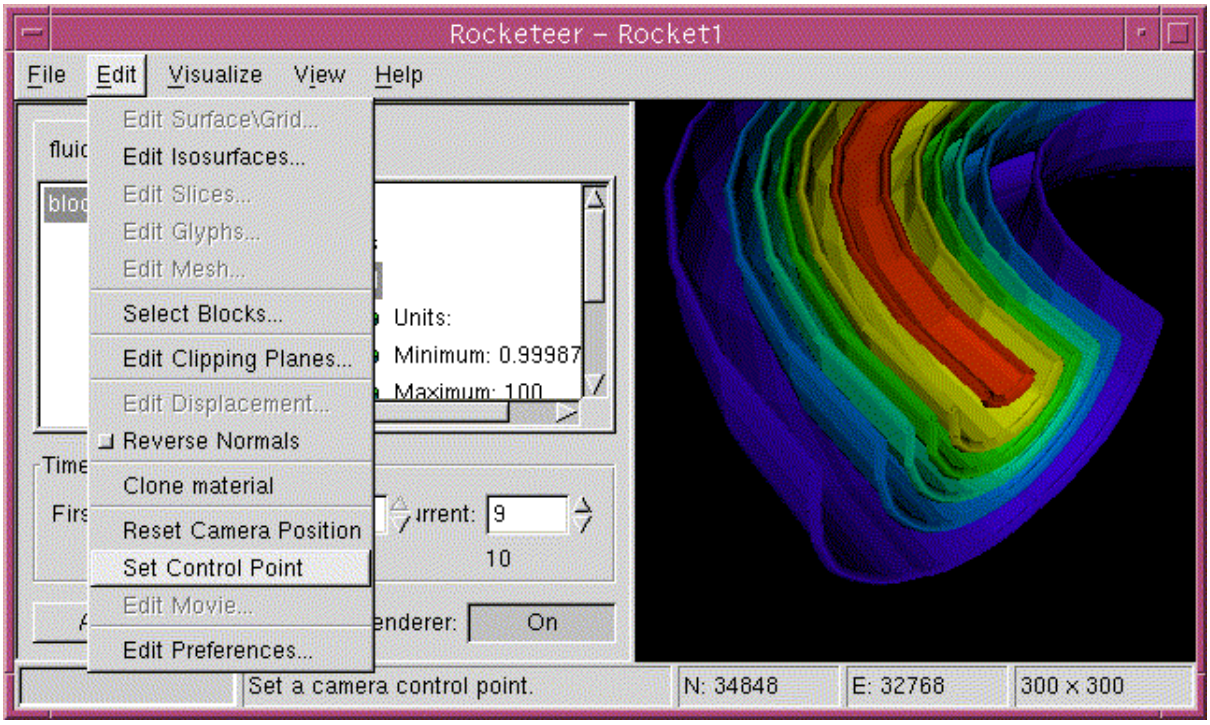
```
<path>convert -delay <n> -compression lzw -loop -1 -geom 50%x50% frm*.tiff animation
```

reduces the images by a factor of 2 in each direction.

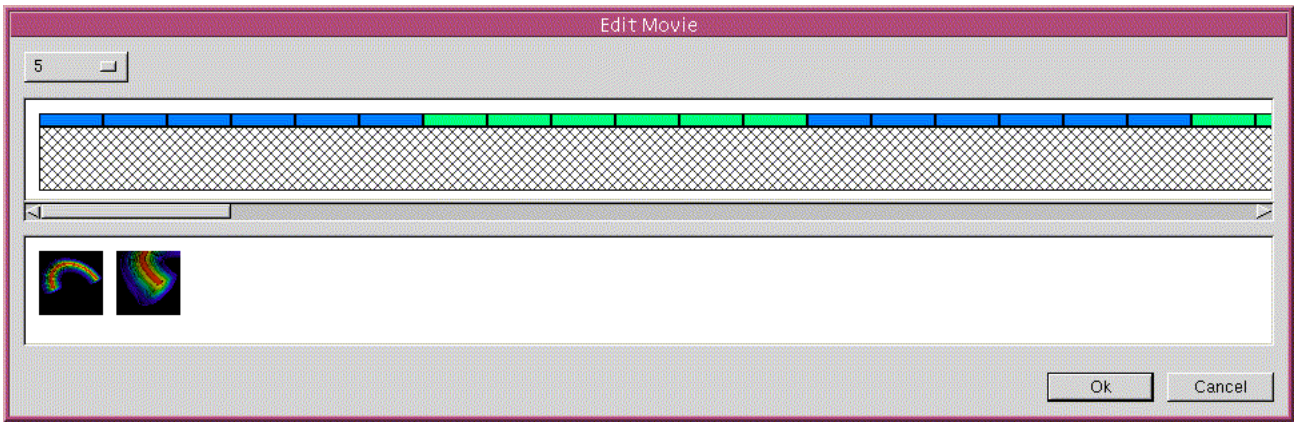
6.14 Camera Motion During an Animation

You can specify a controlled motion for the camera position and viewpoint to create animations with zooming and panning (not working in the current MS Windows release). This is accomplished by setting two or more "control points", which are different views of an image. For movies with a moving camera, Rocketeer generates multiple frames for each output time, interpolating the camera position between control points as it advances from one frame to the next.

We will now create an animation with a moving camera. Before changing the position of the camera from its location in the previous section, choose "Edit/Set Control Point" to select that view as the first control point. Now zoom in on the left arm of the ring, and set a second control point:



Once at least two control points are set, the "Edit Movie" item on the "File" menu may be selected to bring up the "Edit Movie" dialog box, which enables you to specify which output times are to have which camera positions (This dialog is broken in the current MS Windows version):



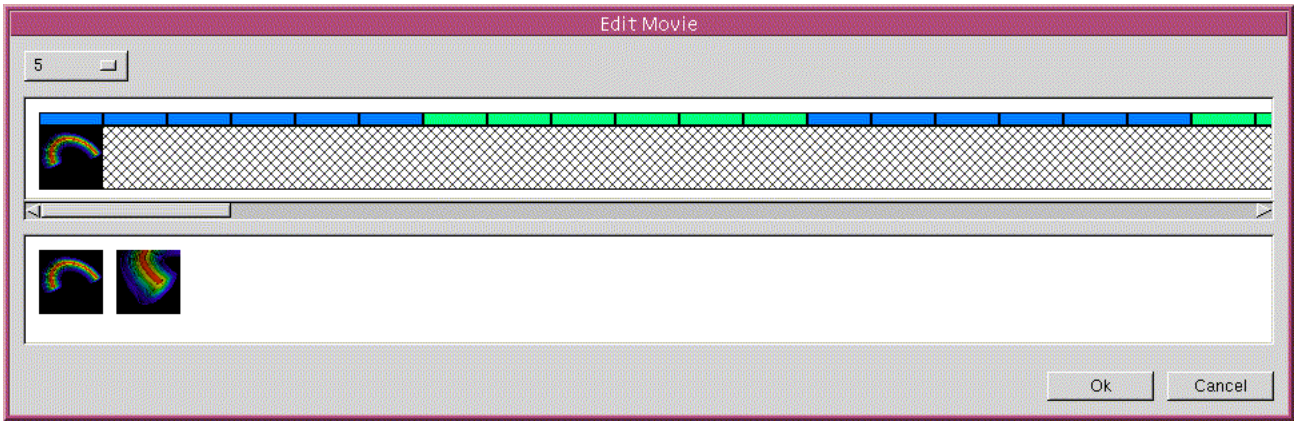
In the lower white box, there are small images of the object at the (two) control points that we have set. You do not have to use all of these control points in the movie, but you can delete control points that you may have saved but are not useful by right-clicking on them and selecting "Delete" from the pop-up menu.

The upper white box contains a "film strip" for the movie, and the menu above it allows you to specify the number of frames to generate for each output time. The alternating colors in the thick blue and green thick dashed line represents the 20 different output times in this

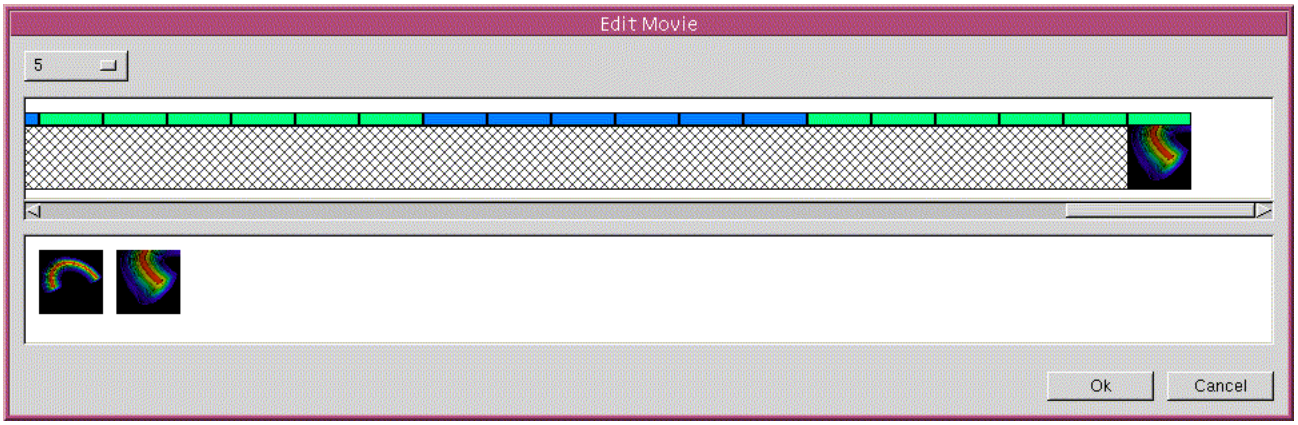
movie. Each dash corresponds to one frame, so in the dialog box shown above, there will be 6 frames generated per output time, for a total of 120 frames in the movie.

It is possible to generate lengthy movies from just one output time, or to change the number of frames to be generated using a particular output time. Simply right-click on the appropriate section of the colored dashed line and select "Increase time" or "Reduce time" from the pop-up menu.

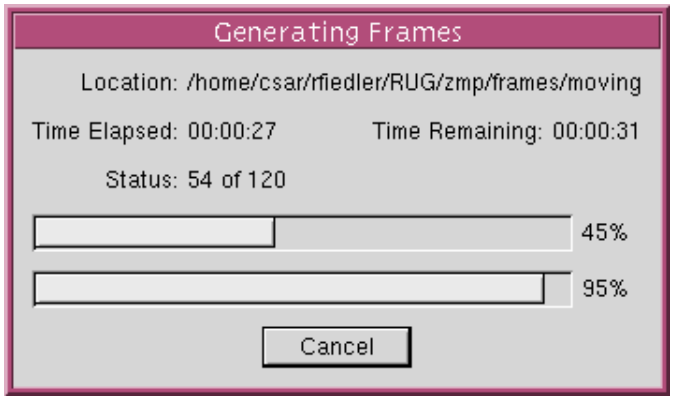
To control the camera position, simply drag the first small image to the beginning of the film strip:



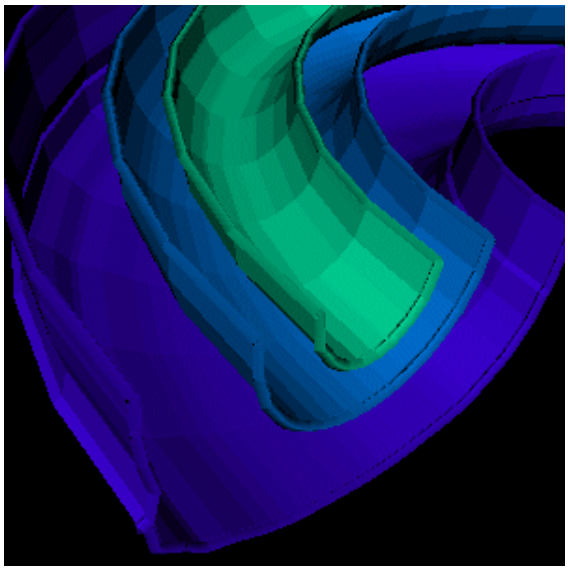
and drag the second small image to the end of the film strip:



After the camera positions for at least the first and last frames have been set, click "OK", and then choose, "File/Make Movie". Note that the "Camera Type" is now "In Motion" rather than "Stationary", as it was in the previous section. The "Edit Motion" button will take you back to the "Edit Movie" dialog box. Specify a directory for the frames, and click "OK" to generate the movie. While rendering images off screen, Rocketeer indicates its progress:



Finally, the 120 frames can be converted into an animation with ImageMagick, for example, as described above. Click the image below to view the animation with moving camera. Note how the camera position moves smoothly from the first to the second control point as the movie plays.



This example illustrated the simplest possible use of camera motion during animation. Much more complex motions of the camera can be prescribed by defining additional control points and dragging them to any desired position in the film strip.

7 Creating HDF Files

If you are using the Rocstar simulation code, you do not have to do any programming to generate HDF or CGNS files. The physics applications in Rocstar make use of Rocin and Rocout to read and write data files in either format. The API here is out of date; it will soon support Rocstar version 3. This API is still useful in reading the HDF dumps yourself using Fortran 90 for the purposes of data analysis.

7.1 A New API Simplifies the Process

We have written a new set of subroutines (or "API"; the tar file includes the API source code, nd xample codes, and a makefile) to make it easier for users to write (and read) HDF files for Rocketeer. This API simplifies the new code a user must write in order to generate output files that Rocketeer can read. It also includes an HDF file checker "Reader.x", and an HDF file rewriter "Rewriter.x" that calls routines in the API to read and store all the data in an HDF file, modify it as desired, and write it out into a new HDF file (the new HDF file name has "new_" prepended to it) The Rewriter.x program can be modified easily to manipulate the data, or used with a script to combine data from multiple blocks into a single file.

The typical usage of the API (F90 version; a similar C++ version also exists - just ask) to write data defined on a structured grid with a single block looks something like the following set of calls:

```
CALL w_block_header (...)

CALL w_geometry_str (...)
CALL w_scalar_str (...)
CALL w_scalar_str (...)
.
.
CALL w_vector_str (...)
CALL w_vector_str (...)
.
.
CALL w_tensor_str (...)
```

These calls create a block header, provide the mesh (geometry data), and then append any number of scalar, vector, and/or tensor data sets. Below we give a description of the arguments. Definitions are not repeated here for the same arguments in different routines. We also describe the corresponding routines for unstructured meshes; their names end with "_unstr" instead of "_str".

```
SUBROUTINE w_block_header (data_file, geom_file, blockname, time, material, mesh,
                           ng, ngn, append)

     data_file - name of the HDF file to contain field variable data
                 (and usually the geometry data as well)

     geom_file - name of the HDF file to contain the geometry (mesh) data
     blockname - name of the data block, of the form "block_0001"
     time - real number defining the physical time in the simulation (or a dump index)
     material - name of the material; blocks can have only one material
     mesh - an integer defining the mesh type
```

```
     1) rectilinear
     2) structured
     3) unstructured (or point data)
     4) discontinuous Galerkin
     5) unstructured surface mesh (quads)

     ng - number of ghost elements (number of ghost layers for structured meshes);
          values of variables in ghost elements are included in the HDF file but
          are ignored  when computing max/min and by Rocketeer when plotting field
          data. The ghost nodes and elements in an unstructured mesh must be the
          last ngn (ng) entries in the arrays that store the node- and
          element-centered data.

     ngn - number of ghost nodes (value of ngn is not used for structured meshes)

     append - set to true if this data block is to be added to an existing HDF
              file; false overwrites

SUBROUTINE w_geometry_str (data_file, geom_file, blockname, material, units,
                           xname, yname, zname, ni, nj, nk, ng, x, y, z)

    units - physical dimensions such as 'meters'

    xname, yname, zname - names for the 3 coordinates
    ni, nj, nk - number of nodes in each direction on the mesh
    x,y,z - the coordinate arrays (3 array subscripts)

SUBROUTINE w_geometry_rect (data_file, geom_file, blockname, material, units,
                           &xname, yname, zname, ni, nj, nk, ng, x, y, z)

    x,y,z - Defined as above, but for rectilinear grids the coordinate arrays
            have just 1 subscript

SUBROUTINE w_scalar_str (data_file, long_name, short_name, units, niq, njq,
                         nkq, ng, q)

    long_name - description of the scalar variable

    short_name - variable name for Rocketeer to display
    niq, njq, nkq - number of nodes or elements (depending on how the data
                    is centered) in each direction on the mesh
    q - the scalar variable, (3 array subscripts)

SUBROUTINE w_vector_str (data_file, long_name, short_name, units, niq, njq,
                         nkq, ng, q1, q2, q3)
```

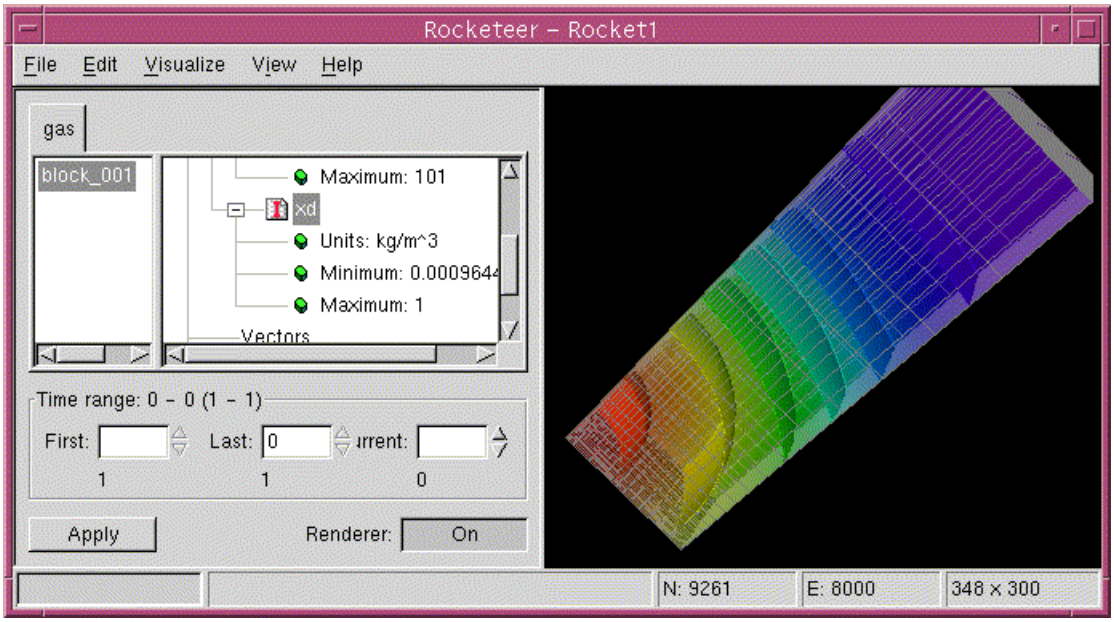
```
    q1, q2, q3 - the vector components; each array has 3 subscripts

SUBROUTINE w_tensor_str (data_file, long_name, short_name, units, niq, njq,
                     nkq, ng, q11, q12, q13, q21, q22, q23, q31, q32, q33)

    q11, q12, q13, q21, q22, q23, q31, q32, q33 -- the tensor components;
                                            each array has 3 subscripts

SUBROUTINE w_geometry_unstr (data_file, geom_file, blockname, material, units,
                xname, yname, zname, nodes, ngn, elems, nconn, conn, x, y, z)

    nodes - number of vertices in the unstructured mesh

    ngn - number of ghost nodes (the last ngn nodes are ghost nodes)
    elems - number of elements in the mesh; ignored for point data (nconn = 1)
    nconn - (largest) number of vertices each element has; 4 for tets, 8 for
          hexahedrons, etc.
            If nconn is set to 1, that implies point data and no connectivity
             data is needed or written.
    conn - connectivity array dimensioned (elems,nconn). If your arrays
            are dimensioned (nconn,elems), the images will not look right.
    x, y, z - coordinates, each 1-D arrays of length "nodes"

SUBROUTINE w_scalar_unstr (data_file, long_name, short_name, units, nq, ng, q)

    nq - number of nodes or elements, depending on how the scalar variable is centered
    ng - number of ghost nodes or elements (the last ng entries are ghost values)
    q - scalar variable, 1-D array of length nq

SUBROUTINE w_vector_unstr (data_file, long_name, short_name, units, nq, ng, q1, q2, q3)

    q1, q2, q3 - vector components, 1-D arrays of length nq

SUBROUTINE w_tensor_unstr (data_file, long_name, short_name, units, nq, ng,
                           q11, q12, q13, q21, q22, q23, q31, q32, q33)

    q11, q12, q13, q21, q22, q23, q31, q32, q33 -- the tensor components;
                                        each is a 1-D array of length nq
```

7.2 Example Codes

To link these codes, you will need the HDF library compiled for your system. Various utilities are also available, including "hdp", which can dump the content of an HDF file in text format. You may use the executable "Reader.x", which calls routines in the to check your HDF files. You may need to modify the makefile provided in the tar file in order to use the example codes on a system not supported therein.

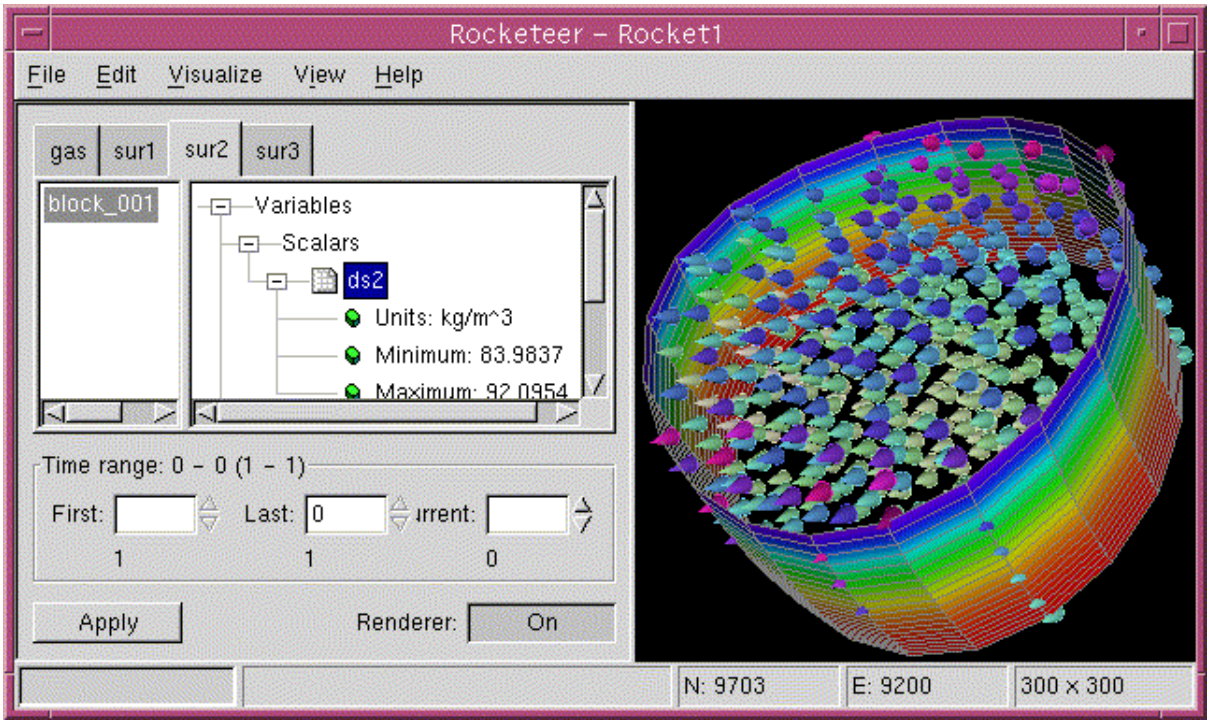
7.2.1 Rectilinear Grid

This Fortran code uses the o write out an HDF file rect.hdf containing a non-uniform rectilinear grid, which is characterized by the fact that the grid can be specified by three 1-D arrays for each of the 3 coordinate directions. The figure below shows isosurfaces, a translucent surface, and the grid for the vertex centered scalar "xd".



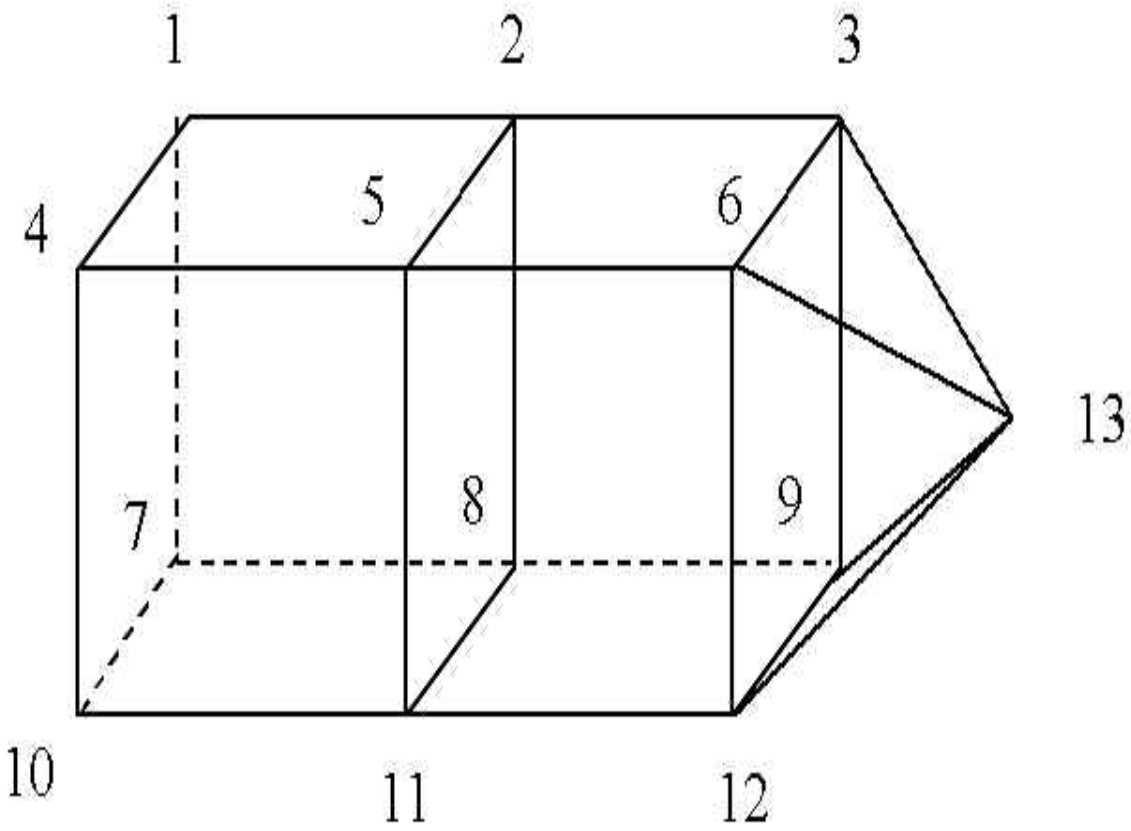
7.2.2 Structured Grid

This Fortran code uses the o write out an HDF file stsurf.hdf containing a structured grid (an orthogonal non-uniform cylindrical mesh), a cell centered 3-D scalar data set, a vertex centered 3-D vector data set, and a cell centered 3-D tensor data set. It also includes scalar and vector data sets defined on 3 different 3-D surface meshes. Here the density (ds2) and the grid are plotted on the curved surface of the distorted cylinder, while the displacement vectors are shown throughout the 3-D volume.

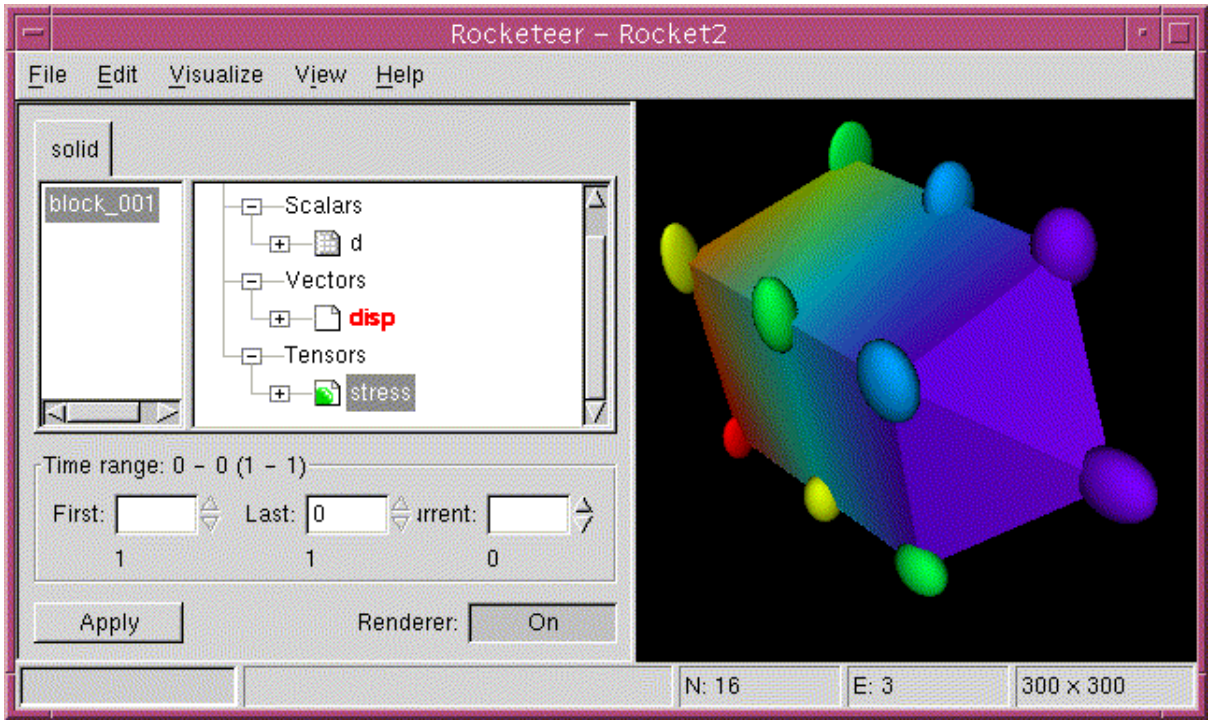


7.3 Unstructured Grid

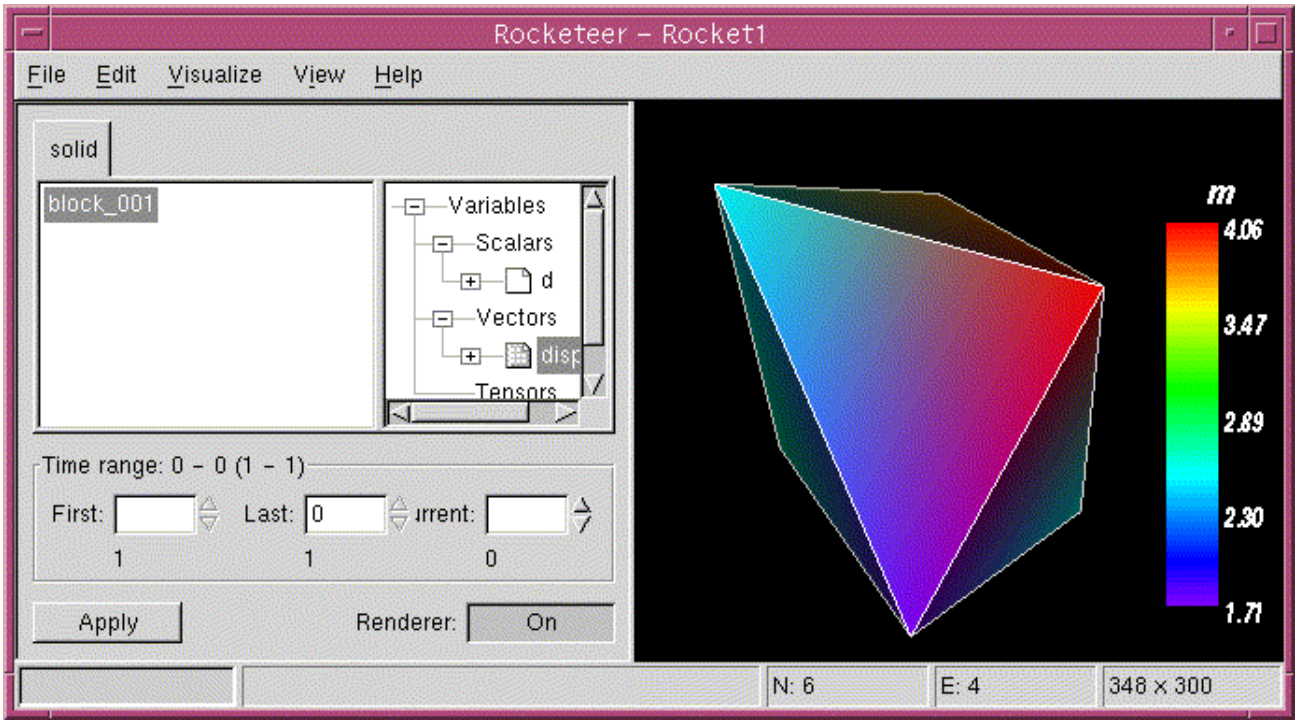
This Fortran code uses the o write out HDF files unstr0.hdf and unstr1.hdf for the very small unstructured mesh example from Figure 2.2 of the Silo User's Guide (LLNL). Note that this grid consists of two hexahedrons and one pyramid. The vertices are numbered as in the following diagram:



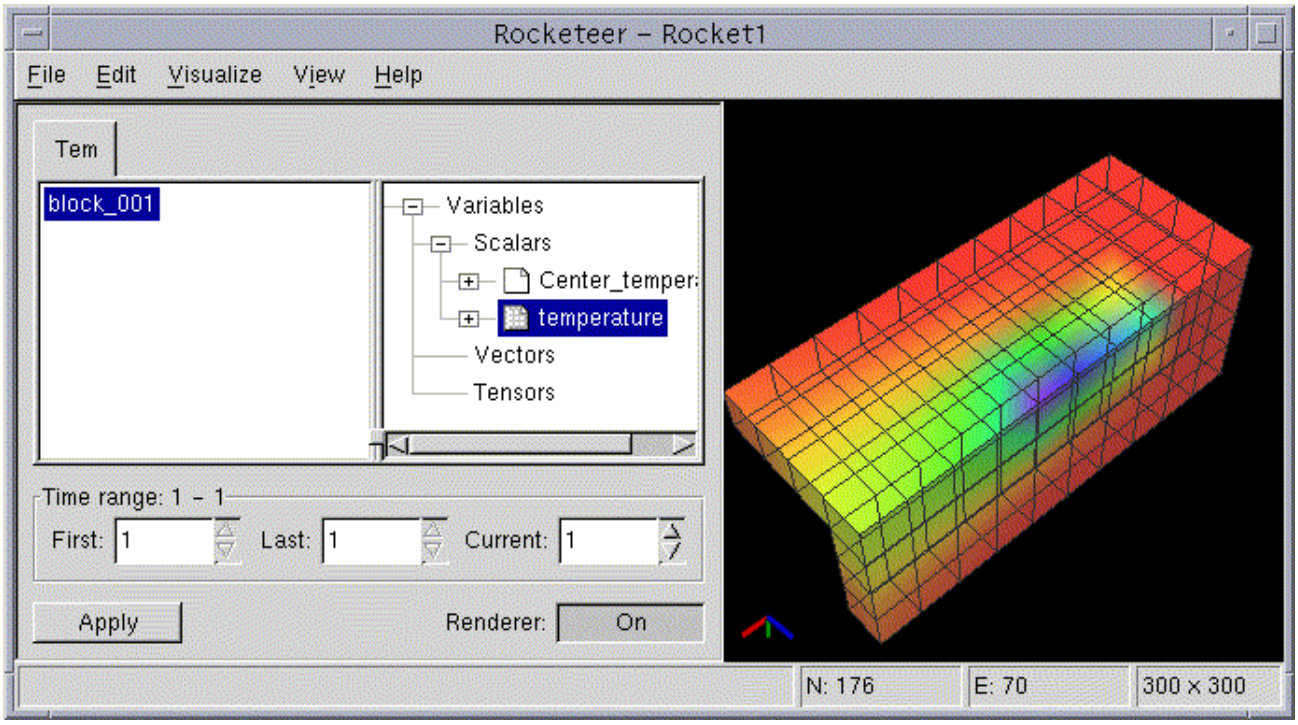
The scalar is shown on the surface, the displacement is used to deform the coordinates, and ellipsoidal glyphs are used to represent the simple tensor field in the figure below. The scale factor had to be increased to 0.1 to show these glyphs.



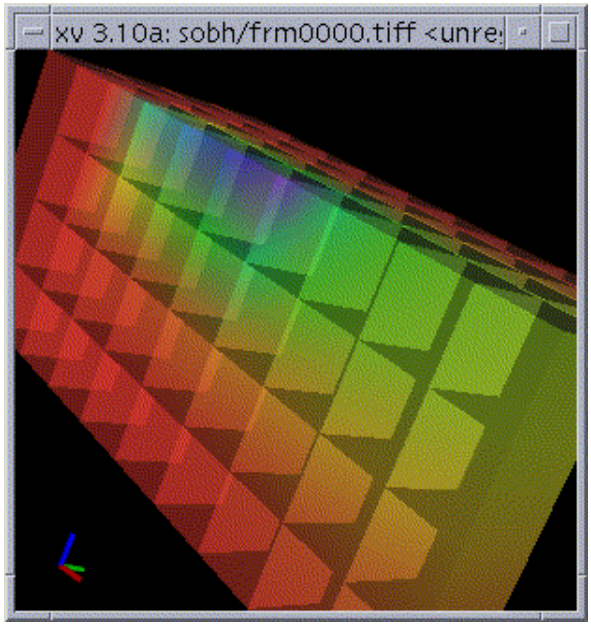
This Fortran code uses the o write out an HDF file ustets.hdf with a second example of an unstructured mesh. A cell centered scalar and a displacement vector are included in the HDF file. In this case, the mesh is composed of four tetrahedra. The magnitude of the "disp" vector and the surface grid lines are shown below.



This Fortran code (with input file) uses the o write out an HDF file Plot_T.hdf containing a discontinuous Galerkin finite element mesh. The surface temperature (translucent) and grid are shown in the image below. Rocketeer automatically merges nodes on surfaces shared by adjacent elements for this mesh type.



If this mesh were presented to Rocketeer as an ordinary unstructured mesh and the surface were made translucent, the unmerged surfaces between elements would be visible:



This feature may be useful for spotting cracks that have closed up.

8 Appendix on HDF

The ides HDF from the user, so you may not need to refer to this Appendix unless your data does not fit the API very well.

One can use just the venerable Single File Scientific Data Set Interface to create HDF 4 files for Rocketeer. (Alternatively, you can use the Multiple File Scientific Data Set Interface if, for example, you need to perform more complex output operations that would require more than one HDF file to be open at a time in the same process).

The basic structure of an HDF file for Rocketeer is the "data block", which consists of three parts: the block header, the mesh, and the variables defined on the mesh. The HDF file may contain more than one data block. Each data block has its own separate header, mesh, and field variable arrays.

8.1 Block Headers

The block header provides the name of the data block, the time level, and the material type. This information is stored in the HDF file as four attributes (text strings) of a character data array:

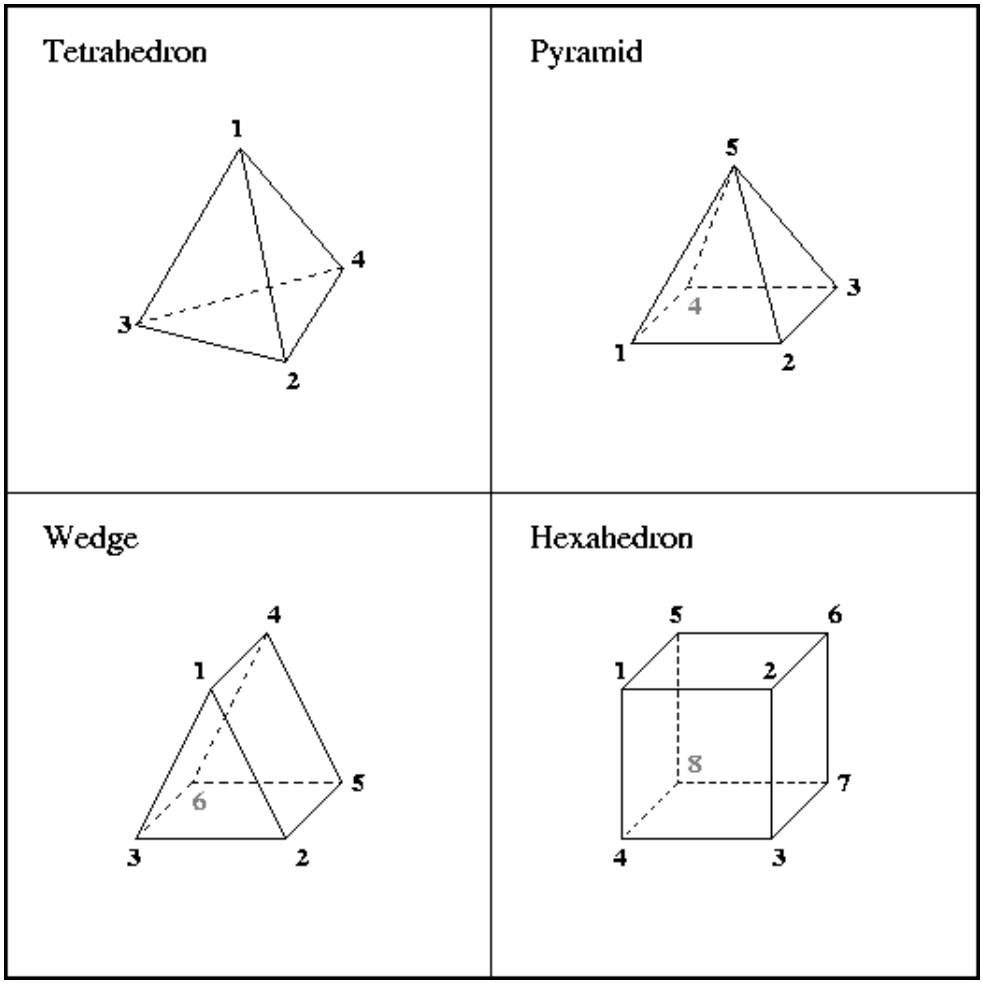
- **Block Name** — uniquely identifies the mesh block, e. g., "block_001".
- **Time Index** — identifies the time level, e. g., "20" may correspond to 10 ms. Real numbers and units are also allowed.
- **Block Header Attribute** — a label for Rocketeer to identify a block header. It is always set to "block header".
- **Material Type** — identifies the type of material, e. g., "gas". Rocketeer assumes that materials do not mix; different materials are assumed to exist on separate, non-overlapping grids. If the grids were to overlap, Rocketeer would display the properties of the last material that you visualize rather than a sum over material properties.

The character data array itself is a string containing the mesh type, number of ghost zone layers, and (optionally) the name of the HDF file containing the undeformed coordinates and connectivity data. See the examples for details.

8.2 Meshes

The precise content of the mesh section depends on the type of mesh on which the data is defined. The four types of mesh are:

- **Uniform** — the mesh need not be described at all. Rocketeer handles a uniform mesh in a reasonably efficient manner, but the whole point of developing Rocketeer was to handle more general meshes.
- **Rectilinear** — the mesh can be described by 3 one-dimensional Cartesian coordinate arrays. A "uniform" grid is a special case; more general rectilinear grids are not uniformly spaced. Note that non-Cartesian meshes are treated as "structured", even if they can be described by three one-dimensional curvilinear coordinate arrays. See the rectilinear example (Section **??**) above.
- **Structured** — the mesh is described by three three-dimensional Cartesian coordinate arrays. Finding neighboring cells is easy on a structured mesh, since the mesh point or cell (x(i, j, k), y(i, j, k), z(i, j, k)) has neighbors at (i+1, j, k), (i, j-1, k), (i, j, k+1), etc. Cartesian coordinates must be given. If your coordinate system is really cylindrical (z, r, phi) for example, convert r and phi to x and y. Spherical coordinates (r, theta, phi) do not work correctly if phi goes through pi (or more) radians because Rocketeer does not see neighbors in the theta direction across the z axis. See the structured example (Section **??**) above.
- **Unstructured** — the 3-D mesh with N vertices and M elements is represented by 3 arrays of length N containing Cartesian coordinate values. The connectivity information for a brick element is given by an M by 8 array (in C, the array is 8 by M) of indices corresponding to the vertices. The connectivity data should be generated using a consistent ordering such as that shown in the figure below.



A tetrahedral mesh would have an M by 4 connectivity array. If you have a mesh consisting of both tetrahedra and hexahedra, the connectivity array would be M by 8. You can enable Rocketeer to correctly interpret the connectivity data for a tetrahedron in two ways, 1) by setting the last 4 values to -1, or 2) by treating it as a hexahedron and repeating the indices of "degenerate" vertices. For example, vertex 4 could be repeated once to complete the bottom quadrilateral, and vertex 1 could be repeated 3 times to complete the top quadrilateral. See the unstructured examples (Section **??**) above.

Rocketeer can handle meshes consisting of 2-D triangles, provided you supply three Cartesian coordinate arrays. The elements of the third coordinate array can be set to zero. If you wish to provide a surface mesh containing quadrilaterals, set the grid type index to 5 rather than 3 in a separate data set.

- **Point Data** — there is no grid, just a set of points. This is presented to Rocketeer as an unstructured grid, with the connectivity data omitted. See the example (Section **??**) above.

8.3 Field Variables

The field variables are quantities defined on the mesh. They are passed as 3-D arrays for uniform, rectilinear, and structured meshes, and as 1-D arrays for unstructured meshes.

The field variables may be cell-centered or vertex-centered. Rocketeer determines the centering by comparing array sizes with those of the coordinate arrays.

Attributes stored with the field variables provide both a long and short name, units, and the component (scalar: 0; vector: 1, 2, 3; tensor: 11, 12, 13, 21, 22, 23, 31, 32, 33). The component can't have any other values. The short name is used by Rocketeer to identify the field variable when multiple data blocks and/or multiple data files are read in. The units are used to determine whether two different functions should be plotted using the same color table. If the units do not agree, Rocketeer displays the second variable using a different color table. The user can override this behavior, if desired. Finally, the component tells Rocketeer whether the variable is a scalar or a component of a vector or tensor. See the discussion of HDF routine in Section 8.4 (dssdast) below for details.

8.4 Essential HDF Routines

Here are the only HDF routines your Fortran code needs to call in order to write out HDF files with all the information needed by Rocketeer. They are integer functions that return 0 if successful and -1 otherwise. For more details, see the on-line HDF Reference Manual.

dssdims (rank, shape)

Specifies the dimensions of arrays subsequently written to the HDF file.

rank – integer; the number of spatial dimensions (e.g., 3)

shape – integer array; the number of elements in each dimension (e.g., (36,36,36)).

dssnt (i_type)

Specifies the data type of the data to be written in the next write operation.

i_type – integer; common choices include 5 (32-bit float; the default), (64-bit float), and 24 (32-bit signed integer). Note: be careful if you are writing out 32-bit floats from a program that works with double-precision variables. In this case you need to copy the double precision values to a single precision array.

dssdast (label, units, format, coorsys)

Specifies the "label", "units", "format", and "coordinate system" attributes for the next data set to be written. These attributes are defined by the HDF library, but it is up to the application (Rocketeer) to read and interpret them. Thus, we have taken liberties with their meaning and use them to help interpret the data.

label – character string; long name for a field variable (e.g., 'Density at t = 0.23e-04 sec'). Rocketeer does not even display this, but HDFView does, and it is a good way to store the physical time value in the file.

units – character string; units (e.g. 'kg/m^3')

format – character string; Rocketeer uses this to define a short name for a field variable (e.g., 'density').

coorsys – character string; Rocketeer uses this to determine the type of variable. Select from '0' for scalars, '1', '2', or '3' for the x, y, or z components of a vector, and '11', '12', '13', '21', ... , or '33' for the xx, xy, xz, yx, ... , or zz components of a tensor.

This routine is also used to provide the block header data as the attributes of a character data array that specifies the type of mesh. See the example codes given above.

dssrang (max, min)

Writes out the maximum and minimum values of a data set. This saves Rocketeer some work when reading in a series of data sets.

max, min – same type as the data

dspdata (filename, rank, shape, data)

Open an HDF file and write a Scientific Data Set. Use this call once per HDF file; subsequent data sets must be written using dsadata.

filename – character string; name of the new HDF file

rank – integer; the number of spatial dimensions (e.g., 3)

shape – integer array; the number of elements in each dimension (e.g., (32,32,32))

data – data type specified by the last call to dssnt; values of the scalar variable. Rocketeer will compare this array's dimensions with those of the coordinate arrays to

determine if this data is defined on cell vertices or cell centers. Rocketeer currently does not fully support staggered meshes, i.e., it does not automatically handle face-centered quantities (except on surface meshes).

dsadata (filename, rank, shape, data)

Like dspdata, but appends data to an existing HDF file instead of overwriting it.