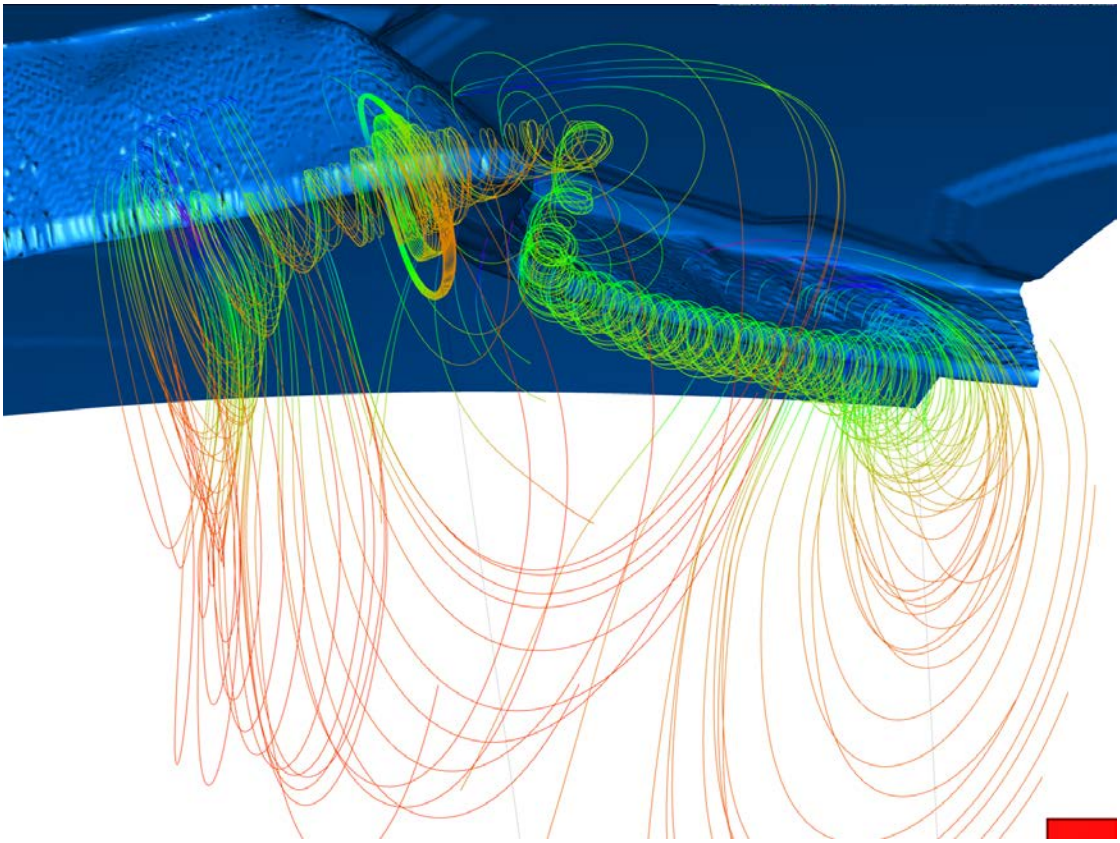
# 3D Visualizer Version 1.0
# User's Manual for Desktop Environments

Margarete A. Jadamec[*,†] Oliver Kreylos[‡,†] Magali I. Billen[*,†]

February 27, 2009



[*]Department of Geology, University of California, Davis
[†]W. M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES), University of California, Davis
[‡]Institute for Data Analysis and Visualization (IDAV), University of California, Davis

# Contents

# 1 Preface

## 1.1 About this Document

This document describes how to install the 3D Visualizer software and its supporting libraries, launch the 3D Visualizer application, and use it to view and analyze gridded three-dimensional (3D) data on desktop workstations or laptop computers. 3D Visualizer includes support to directly load data stored in several commonly used formats, such as the native formats used by CitcomS or CitcomCU. This document describes how to load data that is stored in one of the supported formats, and gives hints on how to modify 3D Visualizer's source code to add support for additional data formats. This document also contains tutorials showing how to use 3D Visualizer on several example data sets, one of which is provided for download.

The 3D Visualizer software, its supporting libraries, and the bundled example data set can be downloaded from http://www.keckcaves.org/software/index.html.

While this user's manual focuses on using 3D Visualizer in desktop environments, the software also works in semi-immersive or fully immersive virtual reality (VR) environments, such as Geowalls or CAVEs. Please contact the developers via email at visualizer-software@keckcaves.org for information on how to install and use 3D Visualizer in such environments. The software offers the same functionality in all supported environments, but its user interface is environment-dependent.

## 1.2 Citations

The 3D Visualizer software is an efficient means to view and analyze 3D data. Our goal is to make this software easily available and intuitive to use. Using the software can greatly increase the efficiency of 3D data analysis. In addition, it can provide a medium that improves the conceptualization of 3D scientific problems and processes. We ask that, in presentations and published work, you acknowledge the people who developed and tested this software. Please use the following citation:

> Billen, M. I., Kreylos, O., Hamann, B., Jadamec, M., Kellogg, L. H., Staadt, O., and Sumner, D. Y., (2008), A Geoscience Perspective on Immersive 3D Gridded Data Visualization. Computers and Geosciences.

## 1.3 Support

We greatly acknowledge the support of the UC Davis Institute for Data Analysis and Visualization (IDAV, http://idav.ucdavis.edu), the UC Davis W. M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES, http://www.keckcaves.org), the W. M. Keck Foundation, and the University of California, Davis. In addition, part of this work was carried out under the National Science Foundation Grant No. EAR–0537995.

## 1.4 Free/Open Source Software



## 1.5 Bug Reporting

For questions or comments about 3D Visualizer, or to report any bugs, please send email to:
visualizer-software@keckcaves.org and put the words "3D Visualizer" in the subject heading.

You can also go to the 3D Visualizer web site and follow the bug tracking link:
http://www.keckcaves.org/software/VISUALIZERCG/index.html

# 2 Introduction to 3D Visualizer

3D Visualizer is an open source software package developed for interactive, visual exploration of gridded 3D data in immersive VR environments. In addition to immersive VR environments, 3D Visualizer runs on Geowalls and on desktop workstations and laptop computers based on Linux or UNIX variants (including MacOS X). 3D Visualizer contains a variety of modules that enable reading gridded 3D data, such as those produced by finite-element method (FEM) simulations, with little to no required pre-processing.

## 2.1 History

This is the first public release of the 3D Visualizer software, written by Oliver Kreylos at the University of California, Davis. The software began its life in 1999 as a prototypical testbed for algorithms to view, and interact with, gridded 3D data in immersive environments, such as CAVEs. Around the same time, Kreylos started working on Vrui, a software development toolkit for virtual reality applications with a focus on portability between diverse environment types. Kreylos re-implemented the early version of 3D Visualizer using Vrui starting in 2000, and this led to the software running on a wide variety of visualization environments, including standard desktop environments. 3D Visualizer was rewritten from scratch starting in 2002, with the new goal to serve as a testbed for highly efficient generic grid-agnostic visualization algorithms implemented using the C++ template mechanism. This version of 3D Visualizer has been used in scientific research at the UC Davis W. M. Keck Center for Active Visualization in the Earth Sciences (KeckCAVES) since 2004. In 2006, the underlying Vrui toolkit (and, by extension, 3D Visualizer) was adapted to run on MacOS X, with several changes made by Braden Pellett.

## 2.2 Approach to Visualization

The fundamental goal behind any visualization is to use images to convey meaning. However, there are different approaches to realize that goal. Many visualization applications focus on creating high-quality publication-ready static images, with careful fine-tuning required to create those images that are best-suited to convey their intended meaning. Such applications work best when users know beforehand what they want to say with their images, i. e., if the information is contained in their data is already and how to best show it is already known. For example, a user might know a priori that an isosurface for a particular data value is important, and that a particular region of that surface shows an important detail feature. She might then use a visualization application to draw that surface, by entering the corresponding isovalue into a text field and waiting a few seconds until the isosurface is created, and the pointing a "virtual camera" at the known position of the detail feature it contains. She will then slightly adjust camera settings to improve the presentation of the detail feature, and generate a final image.

3D Visualizer, on the other hand, focuses on interactive exploration of data, and on interactive viewing. In other words, while 3D Visualizer is capable of generating static images for publication, its real purpose is for exploring data if the information contained in the data is *not* known a priori. We believe that this focus requires a different approach to the process of visualization. If, for example, a user suspects that there is a "bug" in he model, he will not know which isosurface will show that bug; he might not even know what that bug should look like. Using the standard approach, he might try creating several surfaces, but find nothing out of the ordinary. And, even if a suspicious feature is found, a standard visualization application might make it difficult to examine, or to measure its exact position and size.

3D Visualizer was designed to support exactly such *data exploration*, by focusing on interactivity and real-time response. Instead of entering an isovalue and waiting a few seconds for an isosurface to appear, users can directly select a point inside the data domain, and immediately – in less than 0.1 s – see a partial isosurface around the point selected. They can then drag the selection point through the data, and immediately see how the isosurface changes in response. Once a surface of interest is found, it can be examined by moving the camera around, using an intuitive user interface for *interactive navigation*.

In 3D Visualizer, almost all user actions have an immediate visual response. We believe that this immediate feedback makes 3D Visualizer ideally suited for exploration of previously unknown data. Put another way, 3D Visualizer does not aim to produce a single image to convey meaning to a larger audience, but produces rapid sequences of changing images to convey meaning to a single user – who can then use 3D Visualizer, or another visualization application, to communicate the found meaning to a larger audience.

Conceptually, 3D Visualizer consists of three interacting building blocks: *data representation*, *visualization elements*, and *algorithms*. Data representation holds the data to be visualized, i. e., gridded 3D data with multiple scalar and/or vector variables; visualization elements are the actual visualization drawn, e. g., color-mapped slices or isosurfaces; and algorithms are the functional elements that extract visualization elements from the data representation (e.g., a seeded isosurface).

### 2.2.1 Data Representation

In 3D Visualizer, all data sets that can be visualized are defined by a *domain*, i. e., a subset of three-dimensional space $D \subset \mathbf{R}^3$, a *grid*, i. e., a discretization of the domain $D$ into individual *grid cells* $C_i \subset D$ defined by *grid vertices* $v_i \in D$, and by a set of scalar variables $S_i \colon D \to \mathbf{R}$ and/or vector variables $V_i \colon D \to \mathbf{R}^3$ defined over the domain $D$.

**Domain** 3D Visualizer assumes that all domains are defined in Cartesian space $(x, y, z)$. Therefore, data defined in spherical coordinates is converted to Cartesian coordinates during loading, and any vector variables defined with spherical-coordinate data are converted to their analogous Cartesian representations at the same time. It is of course possible to visualize spherical-coordinate data in spherical coordinates by making 3D Visualizer believe that the coordinate axes $(\mathit{longitude}, \mathit{latitude}, \mathit{radius})$ are the Cartesian axes $(x, y, z)$, but we believe it is more appropriate to visualize spherical-coordinate data in Cartesian coordinates to avoid forming inaccurate mental models due to the shape distortions inherent in visualizing a sphere as a rectangular box.

**Grid** Any grid is defined by a set of *grid vertices*, i. e., points in (Cartesian) 3D space, and a set of *grid cells* connecting those vertices. Grid cells of a data set are assumed to overlap only at their faces, and to tile the data set's domain. The current version of 3D Visualizer supports three different grid types: Cartesian, curvilinear, and simplical (see Figure 1). Note a grid is different from a domain, therefore a data set can have a Cartesian domain (coordinates), but be defined on a simplical grid. In addition, a cartesian grid could be used with data in spherical coordinates. Cartesian grids are regular hexahedral grids: all grid cells are identical rectangular boxes, and all grid lines are aligned with the primary coordinate axes $x$, $y$, and $z$. The positions of the grid's vertices are implicitly defined by the grid's cell sizes. Curvilinear grids are irregular hexahedral grids: all grid cells are six-sided deformed cubes with straight edges, and the positions of grid vertices are explicitly defined. Simplical grids are similar to curvilinear grids in that all grid vertices have explicitly defined positions, but in simplical grids all grid cells are tetrahedra, i. e., four-sided pyramids. 3D Visualizer also supports multi-curvilinear grids, which are multiple curvilinear grids whose boundary cell faces are connected to each other. In the geosciences, such grids typically result from global simulations of the Earth's mantle, where a spherical shell is discretized as twelve curvilinear "caps."

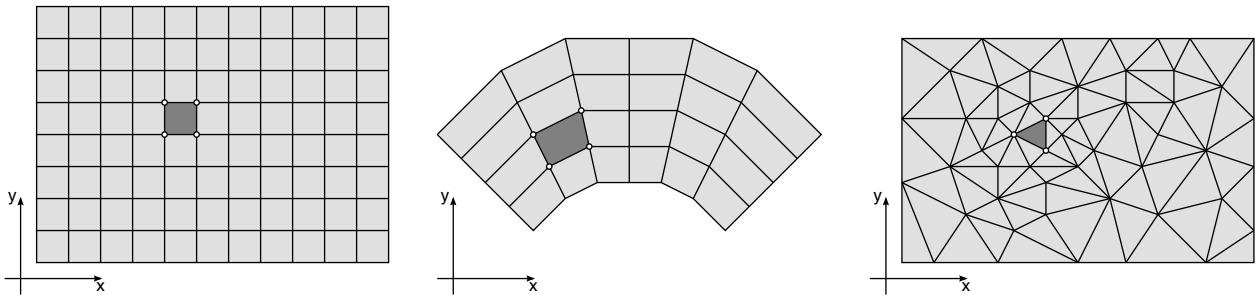



Figure 1: Grid types supported by 3D Visualizer. *Left*: Cartesian grid. *Center*: Curvilinear grid. *Right*: Simplical grid.

**Scalar and Vector Variables** 3D Visualizer assumes that, for all grid types, the scalar and vector functions of a data set are defined by explicit values assigned at all grid vertices, i. e., 3D Visualizer expects *vertex-centered data*. The underlying 3D functions are reconstructed by interpolating vertex data values inside cells. For Cartesian and curvilinear grids 3D Visualizer uses tri-linear interpolation; for simplical grids, it uses linear interpolation.

### 2.2.2 Visualization Elements

Visualization elements are the actual graphical components of a visualization. In desktop environments, they are drawn from the point of view of a "virtual camera" and create the 2D image that can be seen on the computer's screen or printed on paper. In immersive virtual reality environments, visualization elements are directly shown as 3D objects, allowing users to examine and manipulate them more intuitively. The following visualization elements are supported by the current version of 3D Visualizer:

**Slice** A slice is a planar surface cutting through a data set's domain, color-mapped by an arbitrary scalar variable.

**Isosurface** An isosurface is a surface connecting all points in a data set's domain that have identical values of an arbitrary scalar variable.

**Volume Renderer** A volume renderer visualizes an arbitrary scalar variable over the entire domain of a data set, by drawing the domain filled with a semi-transparent color-mapped material. *Caution: volume rendering requires a decent graphics card and driver. If your graphics card/driver does not have the required support 3DVisualizer will crash when you try to use volume rendering.*

**Arrow Rake** An arrow rake is a local regular 2D grid of arrows, whose directions and lengths visualize an arbitrary vector variable at the vertices of the 2D grid. The arrow rake can be generated at any arbitrary location in the data set's domain.

**Streamline** A streamline follows the path of a massless particle that is advected through the data set's domain by an arbitrary vector variable.

**Streamline Bundle** A streamline bundle is a set of related streamlines that originated from a small circle.

### 2.2.3 Algorithms

In 3D Visualizer, algorithms are the building block connecting data representation and visualization elements. Each algorithm creates a particular type of visualization element, based on an interactively selected point inside the data set's domain. For example, the "Seeded Isosurface" algorithm calculates the value of a scalar variable at the selected point by interpolating the scalar variable's values at the vertices of the cell containing the selected point, and then grows an isosurface for that value outwards from the selected point. A "Streamline" algorithm creates a streamline by releasing a massless particle into the data set's domain at the selected point, and then advecting that particle using an iterative solver for the set of ordinary differential equations defining advection (in the current version, an adaptive-step size fourth-order Runge-Kutta method).

In addition to the algorithms extracting the types of visualization elements listed in the previous section, there are several algorithms that do not directly create any visualization elements but are also useful for analyzing the data:

**Cutting Plane** This algorithm creates a plane centered around the selected point that will cut away any visualization elements behind that plane. In desktop environments, this plane is typically aligned with the screen, and cuts away any elements in front of the screen. In immersive environments, the plane can be oriented arbitrarily.

**Evaluate Scalars** This algorithm opens a dialog box that shows the position of the selected point (in Cartesian coordinates), and the numerical value of an arbitrary scalar variable at the selection point's position.

**Evaluate Vectors** This algorithm opens a dialog box that shows the position of the selected point (in Cartesian coordinates), and the numerical values of the (Cartesian) components of an arbitrary vector variable at the selection point's position. In addition, it draws an arrow glyph rooted at the selected point, which visualizes the direction and magnitude of the vector variable at that point.

## 2.3 Uses in Geoscience

The 3D Visualizer software can be used to visualize gridded 3D data containing scalar and/or vector variables. In the geosciences, 3D Visualizer can be used to explore and visualize a range of data from output of geodynamic

models to reconstructions of Archaean microbialites. In geodynamics, this software is particularly well-suited for the exploration and analysis of large 3D plate tectonics and mantle convection models. The ability of the software to allow the user to point and click anywhere inside the data and extract isosurfaces, slices, and streamlines is particularly effective. 3D Visualizer directly reads in the output from several Computational Infrastructure for Geodynamics (CIG) supported software programs, such as CitcomT, CitcomCU-1.0.2, CitcomS-3.0.1, and Gale-1.2.1 (without the particle tracers). 3D Visualizer can also visualize 3D volumes from local and global seismic tomography models. For these data, 3D Visualizer's volume rendering and isosurface extraction tools are particularly useful. These two tools also provide effective visualization and exploration of 3D reconstructions of sedimentary structures generated using Computed Tomography (CT) or serial sectioning.

## 2.4 Approach to Data Formats

Most visualization software only supports reading data from a relatively small set of fixed storage formats, and typically, these formats are different from the output formats used natively by simulation codes. This implies that users must convert any data they wish to visualize from their simulation codes' native data formats to their visualization software's native data format, a process that is tedious, time-consuming, and wastes space by requiring data to be stored in multiple versions. 3D Visualizer, on the other hand, was developed without particular data storage formats in mind. In a way, 3D Visualizer is not a packaged application, but a toolbox of interacting *visualization components*, such as grid representation or slice, isosurface, or streamline extraction, that are linked together into *modules* by a C++ compiler. Each module contains code that has been automatically generated and optimized by the C++ compiler for a particular data representation and storage format. 3D Visualizer itself is just a wrapper around the set of provided modules. The guiding principle behind 3D Visualizer's module approach is to closely couple a *data source*, such as a simulation code producing data, with a *data sink*, such as a visualization application analyzing the data. At a minimum, this coupling is achieved by 3D Visualizer directly reading output data from the simulation codes employed by a user; in the extreme, a user could write special modules that read data directly from a running simulation code's in-memory data structures without having to write the data to temporary files.

Due to the way 3D Visualizer is designed, it would be relatively straightforward for a user familiar with C++ to create a new module supporting an additional data storage format. However, to make 3D Visualizer immediately useful to a large community, this release contains several pre-made modules to directly read a variety of common data formats out-of-the-box (see Table 1). Specifically, this release contains modules to read data files produced by CitcomS-3.0.1, CitcomCU-1.0.2, Gale-1.2.1, a module to assemble stacks of images into 3D volumes, and two generic modules to read common ASCII file formats, such as those produced by the older CitcomT, or those typically used as interchange formats between other simulation codes and visualization applications.

Data Formats Supported by 3D Visualizer Out-of-the-box

| File Format | Compatibility | Module Name in 3D Visualizer |
|---|---|---|
| Generic ASCII | Generic spherical or Cartesian ASCII data, split files | StructuredGridASCII |
| Generic ASCII | Generic spherical ASCII data, single file | SphericalASCIIFile |
| Legacy VTK | VTK data and output from Gale-1.2.1 | StructuredGridVTK |
| CitcomS ASCII | Data from global CitcomS-3.0.1 models | CitcomSGlobalASCIIFile |
| CitcomS ASCII | Data from regional CitcomS-3.0.1 models | CitcomSRegionalASCIIFile |
| CitcomCU binary | Data from CitcomCU-1.0.2 spherical models* | CitcomCUSphericalRawFile |
| CitcomCU binary | Data from CitcomCU-1.0.2 Cartesian models* | CitcomCUCartesianRawFile |
| Image stacks | Computed Tomography and serial sectioning data | ImageStack |

Table 1: Overview of file formats directly supported by 3D Visualizer. *Assumes CitcomCU-1.0.2 was compiled with Output_ucd.c output file (for more information on this output format please see the software download page).

We anticipate that the number of (slightly different) data formats used by the community will grow over time, as simulation codes are adapted to new problems, or modfied to calculate and store new types of data. While there is a move towards unified, "all-purpose" data interchange formats such as netCDF or HDF5, we believe that data types stored by novel applications will not always map one-to-one to generic container formats; therefore, we plan to extend the set of pre-made 3D Visualizer modules in the future. If you are using, or anticipating to use, a particular storage format not currently supported by 3D Visualizer, please feel free to contact us at visualizer-software@keckcaves.org

to work with you on developing a module tailored to that format, and potentially support it out-of-the-box in a future release.

## 2.5 Other Data Imported to and Exported from 3D Visualizer

In addition to data loaded from one of the supported data formats as described in the previous section, 3D Visualizer also reads and writes other data used for, or generated by, the visualization process.

**Meta Input Files** From its command line, 3D Visualizer reads all information about the data sets to visualize, and additional parameters required to load them. Depending on the type of visualization module used, this can lead to long command lines with arcane options that have to be remembered or written down by users (see Appendix A). To simplify this process, 3D Visualizer supports *meta input files*, which are simple ASCII files containing all command line parameters given to 3D Visualizer to load a particular data set, and which can be stored alongside the actual input data. In a way, meta input files are 3D Visualizer's native data file format, but they do not contain actual data. Meta input files have the file name extension ".in" and are loaded by giving their name on 3D Visualizer's command line. Since they contain all information required to load the data set they describe, only command line options controlling other aspects of 3D Visualizer, such as loading color maps or viewpoint files (see below), are necessary.

The exact format of meta input files is as follows: they contain the exact command line needed to load a data set, starting with the visualization module's class name (but not including the `-class` keyword). Command line parameters are separated by whitespace (including line breaks), and command line parameters containing whitespace need to be enclosed in double quotes. For example, to load in a data set in the CitcomCUSphericalRawFile format, on would enter the following command line,

% */path/to/3DVisualizer/executable/file/*`3DVisualizer -class CitcomCUSphericalRawFile` */dataPath/ModelName 0* `-vector` *velo temp* `-log` *visc*

To use the meta input file to run 3D Visualizer (on CitcomCUSphericalRawFile data in this example), one would create a meta input file named *ModelName.in* that contains the following information:

`CitcomCUSphericalRawFile`
*/dataPath/ModelName*
*0*
`-vector` *velo*
*temp*
`-log` *visc*

and run 3D Visualizer as

% */path/to/3DVisualizer/executable/file/*`3DVisualizer` */PathToInFile/ModelName.in*

For more information on the command line flags that are specific to each data format, see Appendix A. You can add the additional flags listed see Appendix A to your *ModelName.in* file.

**Color Maps** All visualization methods use *color maps* to display numerical data values as colors. 3D Visualizer can automatically generate a set of standard color maps when loading data, such as intensity ramps or color blends, but can also read pre-defined color maps from *palette files*. A palette file is a text file containing a sequence of two or more *control points*, one per line, which define a numerical data value in the first column, and the red, green, blue, and opacity components of the color to which that data value maps in the second to fifth columns. The numerical data values can be arbitrary, but must be monotonically increasing from the top to the bottom of the palette file. The red, green, and blue components directly define the visible color, and the opacity component is used to draw data value ranges as more or less transparent in volume rendering. A single palette file can be loaded into 3D Visualizer by providing its name on the application's command line, preceded by the **-palette** keyword.

3D Visualizer contains an interactive palette editor to create or manipulate color maps. The currently visible color map can be written to a palette file at any time. 3D Visualizer writes all palette files to the directory from which it was started, using file names "SavedPaletteXXXX.pal," with the Xs replaced by increasing numbers.

**Viewing Parameters** 3D Visualizer allows the user to save and load the position, viewing direction, and zoom factor of its virtual camera to *viewpoint files*. Viewpoint files are binary files, i. e., they are not directly human-readable, storing viewing parameters in an environment-independent format. This means that, for example, a viewpoint file saved in a desktop environment can be loaded in an immersive CAVE environment, and it will show the same parts of a data set from the same viewpoint as on the desktop. 3D Visualizer writes viewpoint files to the directory from which it was started, using files names "SavedViewpointXXXX.dat," with the Xs replaced by increasing numbers. A previously saved viewpoint file can be loaded into 3D Visualizer either by providing its name on the application's command line, preceded by the **-loadView** keyword, or by selecting the "Load View" entry in the "3D Visualizer" → "Vrui System" → "View" menu. However, there is currently no way to define the name of the viewpoint file that is loaded when selecting the menu entry; 3D Visualizer will always try to load a file named "Viewpoint.dat." To load a particular viewpoint file via the view menu, currently a user has to rename the desired file to "Viewpoint.dat" before starting 3D Visualizer. Alternatively, a user can load multiple viewpoint files in sequence during the same session by renaming viewpoint files from a second terminal window immediately before loading them from inside 3D Visualizer. *This feature will be improved in future releases.*

**Screen Shots** 3D Visualizer has a built-in mechanism to save the contents of its display window to an image file. This is the preferred way to generate screen shots; typically, operating system-provided means to capture screen shots do not work properly due to 3D Visualizer's way of using 3D graphics. A screen shot can be saved at any time by pressing the p key, or, unless the used window manager intercepts it, by pressing the "Print Screen" key. Screen shots are written to the directory from which 3D Visualizer was started, using file names "VruiScreenshotXXXX.ppm" (or "VruiScreenshotXXXX.png" if Vrui was compiled with support for PNG images), with the Xs replaced by increasing numbers.

**3D Measurements** Since 3D Visualizer's main focus is interactive data exploration, one of the more important results generated from using it is a set of quantitative measurements of the positions and sizes of found features. Measurements are performed using a *measurement tool*, which is by default configured to write any measurements made by a user to a file in the directory from which 3D Visualizer was started, using file names "MeasurementToolXXXX.dat," with the Xs replaced by increasing numbers. Measurement files are simple text files containing a sequence of position, distance, or angle measurements in Cartesian coordinates, in the same format as they are displayed in the measurement tool's dialog box.

# 3 Getting Started

Here we describe how to install Vrui and 3D Visualizer and how to open and load data into 3D Visualizer. In the installation directions below:

> `courier font`: indicates the exact commands to be entered by the user.
> *Italics font*: signifies the example variable that you will substitute.

If you have trouble, see the trouble-shooting appendix which lists some of the system issues we have encountered (Appendix C). For users that are unfamiliar with using the UNIX/Linux command line or installing software from within a terminal window, please see the "Tips on Command-Line Installation" in section 3.7.

## 3.1 System Requirements

1. UNIX, Linux, or Macintosh OS X systems. 3D Visualizer is not currently available for Windows OS.
2. X11, openGL, and a C/C++ compiler.
3. Mac OS X systems will also need Xcode.

Because X11, openGL, and a C/C++ compiler (such as gcc) are not part of the standard installation on Mac OS X, the Xcode package needs to be installed. Xcode can be found on the installation disks as an optional install or can be downloaded from: `http://developer.apple.com/tools/download/`. Download the Xcode 2.5 DMG if you are running Mac OS Tiger. Download the Xcode 3.0 DMG if you are running Mac OS Leopard. Follow the instructions on the website.

## 3.2 Downloading Vrui and 3D Visualizer

Download three files from the 3D Visualizer distribution website:
`http://www.keckcaves.org/software/VISUALIZERCG/index.html`

1. `Vrui-1.0-050.tar.gz` (The current release of Vrui.)
2. `3DVisualizer-1.0.tar.gz` (The current release of the 3D Visualizer software.)
3. `Slab3dB.tar.gz` (File containing an example data set and the accompanying input and color palette files.)

## 3.3 Installing Vrui

Install the Vrui software, before you install the 3D Visualizer software. (The Vrui toolkit was described in the Introduction.) In the instructions below, we assume the installation of Vrui takes place in the "src" directory, a subdirectory of the user's home directory. In addition, we refer to the path to the downloaded software files as: "/*download*/*path*/". Do the following to install Vrui:

Open a Terminal or X11 window. Change to the src directory, and unpack the Vrui-1.050 tarball. Then change to the Vrui-1.0-050 directory, and compile and install the software. For example*,

```
% cd src
% tar xfz /download/path/Vrui-1.0-050.tar.gz
% cd Vrui-1.0-050
% make && make install
```

*Note:* The default option is to install the Vrui-1.0 executables, libraries, and configuration files to the `$HOME/Vrui-1.0` directory. If you want Vrui installed to a different directory, edit the makefile *before* you make and install Vrui. For example, open the makefile located in ∼`/src/Vrui-1.0-050` and change the `INSTALL` parameter from:
`INSTALLDIR = $HOME/Vrui-1.0`
to a directory in your path, such as:
`INSTALLDIR = $HOME/`*Mybin/ExternalSoftware*`/Vrui-1.0`
Save and close the makefile. Then execute the `make && make install` commands.

It will take a couple of minutes to build Vrui. If there are error messages during the build or you want more details about the Vrui installation, read the README file in the ∼`/src/Vrui-1.0-050 directory`. For more information you can also go to: `http://graphics.cs.ucdavis.edu/`∼`okreylos/ResDev/Vrui/index.html`.

## 3.4 Installing 3D Visualizer

After Vrui has been successfully installed (see section 3.3), install the 3D Visualizer software. In the instructions below, we assume the installation of Vrui takes place in the src directory. Do the following to install 3D Visualizer:

Open a Terminal or X11 window. Change to the src directory, and unpack the 3DVisualizer-1.0 tarball. Then change to the 3DVisualizer-1.0 directory, and compile the software. For example*,

% `cd src`
% `tar xfz /`*download/path*`/3DVisualizer-1.0.tar.gz`
% `cd 3DVisualizer-1.0`
% `make`

*Note #1:* If you changed the install path of Vrui in Section 3.3, then, *before* you `make` 3D Visualizer, edit the makefile for 3D Visualizer to take into account the new Vrui install directory. For example, open the makefile located in ∼`/src/3DVisualizer-1.0` and change the `VRUIDIR` parameter from:
`VRUIDIR = $HOME/Vrui-1.0`
to directory you chose to install the Vrui executable files. In the example above this was:
`VRUIDIR = $HOME/`*Mybin/ExternalSoftware*`/Vrui-1.0`
Save and close the makefile. Then execute the `make` command.

*Note #2:* By default, the 3D Visualizer lib and bin folders are installed within the ∼`/src/3DVisualizer-1.0/` directory. To install these executables in a different directory, edit the makefile for 3D Visualizer. For example, open the makefile located in ∼`/src/3DVisualizer-1.0` and change the `INSTALLDIR` parameter from:
`INSTALLDIR = $(shell pwd)`
to a directory that is in your path, such as:
`INSTALLDIR = $HOME/`*Mybin/ExternalSoftware/*
Save and close the makefile. Then, in addition to entering the `make` command in the instructions above, also enter:
% `make install`
This will put 3 folders (named bin, lib, and share) within the install directory you specified. The 3D Visualizer executable file is located in the bin folder. Therefore for this example, the explicit path to the 3D Visualizer executable file is: `$HOME/`*Mybin/ExternalSoftware/* `bin/ 3DVisualizer`

*Note #3:* Do not move the 3D Visualizer bin, lib, or share directories after you install 3D Visualizer.

## 3.5 Customizing Your Configuration File Preferences

The configuration file, named Vrui.cfg, sets up the general configuration settings of 3D Visualizer, such as the background color of the display window. The default preferences are designed to be versatile and user friendly. You may

find that you have no need to modify them. But modifying the Vrui.cfg file is easy; you can change it and customize your preferences to set the background color of the display window to white instead of black, for example, or to change the visualizer keyboard assignments. If you did the default installation, the configuration file, Vrui.cfg, is located in:
% `$HOME/Vrui-1.0/etc/`

To change your configuration preferences, open the Vrui.cfg file and locate the heading called "section Desktop". You will find it near the beginning of the Vrui.cfg file. This section contains the desktop settings. The lines of code that correspond to the "section Desktop" are located between approximately lines 27 and 325 of the Vrui.cfg file. Find the preferences you want to change, and make the changes. For the changes to take effect, save the Vrui.cfg file and simply reopen 3D Visualizer. We recommend saving a backup copy of the Vrui.cfg file before you start making changes. Appendix B provides specific examples of how to modify the Vrui.cfg file to change the pre-defined mouse-key combinations.

## 3.6 Opening 3D Visualizer and Loading in a Data Set

Now that you have successfully installed Vrui and 3D Visualizer, you are ready to open 3D Visualizer and load in a data set. In a nut shell, this is done by opening an X11 or Terminal window, changing to your data directory, and entering the command line arguments which launch 3D Visualizer and open an X11 Graphical User Interface (GUI) window of the 3D Visualizer program. The command line arguments tell 3D Visualizer information such as the data format (Module Name) and the types of variables to read in. These are outlined in Table 1 of Section 2 and are described in detail in Appendix A. Each module type has slightly different command line arguments, however, the general form of the command line execution of 3D Visualizer is:

> % */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *OptionalSavedColorPaletteFile* `-class` *ModelPrefixOrBaseDirectory timestep OtherCommandLineArguments*
>
> For example, to unpack the example data set and run 3D Visualizer with this example data set, do the following (*Note:* on Mac OS X, start X11 before running 3D Visualizer):
> % `cd` */Path/To/Your/Data/Directory/*
> % `tar xfz` */download/path/*`3DVisualizer-1.0.tar.gz`
> % */Path/to/3DVisualizer/executable/*`3DVisualizer -class CitcomCUSphericalRawFile slab3dB 0 -vector velo temp -log visc`
>
> This command loads the temperature, viscosity (log-base-10 of viscosity is taken upon read) and velocity fields for the zeroth time-step solution (*See Table 1, Section 3.6 and Appendix A*). A window with the outline of a white box should appear once the data are loaded. *Note:* Because there are many command line arguments, it is often easier to put these arguments into a meta input file (Section 2.5). In this case 3D Visualizer would be executed as:
> % */Path/to/3DVisualizer/executable/*`3DVisualizer slab3dB.in`

Here are a few important notes on the command line format:

1. When you execute the command to open 3D Visualizer, you can do so from any directory on your computer if you use the explicit path: % */path/to/3DVisualizer/executable/file/*`3DVisualizer`
2. The explicit path is not needed if 3D Visualizer is installed in a directory that is included in your search path (e.g., *$PATH*): % `3DVisualizer`
3. The `-palette` flag is optional, but this flag, followed by the name of the color palette file, must appear as the *first* argument when it is used.
4. The `-class ModuleName` flag and keyword sets the module type, which determines both what other flags and information are required on the command line and the format of the data within the input data files.
5. The parameters, *ModelPrefixOrBaseDirectory timestep* indicate which data will be read in and for what time-step (if relevant for the particular module type). Note that *ModelPrefixOrBaseDirectory* may be a prefix, a log file, or even a directory name depending on the module type.

6. Following the required flags and information for each module type, the data variables are listed using keywords. The keywords will depend on the module type. Scalar variables are listed without any flags, while a vector variable is preceded by the flag `-vector`. The base-10 logarithm of a scalar can be calculated as the variable is read in by using the flag `-log` preceding the scale variable keyword

When you run 3D Visualizer, a window will open with a black screen and a white rectangular outline in the center of the window. This is the 3D Visualizer GUI. Loading your data can take from 10 seconds to several minutes depending on the data size. We give examples how to use the command line to open 3D Visualizer using each data format supported out-of-the-box by 3D Visualizer in Appendix A. A complete example of how to use 3D Visualizer (from open to close) on the example data set is also given in Section 8.

## 3.7 Tips on Command-Line Installation

On Mac OS X: the X11 executable can be found in ∼`/Applications/Utilities`.

To determine the explicit path of your "`$HOME`" directory, type:
% `echo $HOME`
in my case, this command echos back:
% `/Users/maj/`
indicating that the explicit path to my `$HOME` directory is `/Users/maj/`. On the command line you can use the environment variable `$HOME` in place of the explicit path to your home directory.

The software source files can be downloaded to any preferred directory in the users home-space, referred to as the "*/download/path/*" in the installation steps. For example, if you downloaded the files to your desktop, then your download path will be:
`$HOME/Desktop/`

To create a directory (i.e., folder) named "src" (if you don't have one already), use the UNIX Command "mkdir" . You can do this in the location where you want to unpack the software tarballs. For example:
% `mkdir src`

To change directories from the command line, use the UNIX Command "cd". For example:
% `cd src`

Unpacking the tar-file. This will make a directory called Vrui-1.0-050 within the current directory. If the tarball is zipped, (i.e., appended .gz), type:
% `tar xfz` */download/path/*`Vrui-1.0-050.tar.gz`
*or* if the tarball is unzipped already, type:
% `tar xf` */download/path/*`Vrui-1.0-050.tar`
*or* if you unzipped and untarred the tarball during download (and you already are in the src directory), then just move the Vrui-1.050 folder to your current directory (i.e., to the directory indicated by the "dot" on the line below):
% `mv` */download/path/*`Vrui-1.0-50 .`
For example, here is what I entered to make the src directory and to unzip/untar the Vrui-1.0-050.tar.gz file:
% `cd $HOME`
% `mkdir src`
% `cd src`
% `tar xfz /Users/maj/Desktop/Vrui-1.0-050.tar.gz`

To run 3D Visualizer from any data output directory (see below), you need to add the explicit path (the default path: `$HOME/src/3DVisualizer-1.0/bin/` or your edited path, e.g.,: `$HOME/Mybin/ExternalSoftware/bin/`) to your search path in your .cshrc (or .bash) file in your `$HOME` directory. After your have updated your shell preferences file, you can check if this search path was successfully added.
For example:
% `which 3DVisualizer`
echos back
% `Users/maj/Mybin/ExternalSoftware/bin/3DVisualizer`
indicating my search path was successfully updated.

# 4 Quick Reference Guide to Navigation, Tools, and Menus

This section provides a quick reference to the 3D Visualizer Main Menu (and submenus), the default mouse-button key combinations used to access tools, the Tool Selection Menu (and submenus), and the method for binding tools to keyboard buttons. This quick reference will be of use to more seasoned 3D Visualizer users. We recommend viewing the movie accompanying the manuscript, which shows 3D Visualizer used on a desktop computer. This will give you a better idea of how to interact with the menus and visualization elements. In addition, if this is the first time using 3D Visualizer, please read the more detailed instructions in Sections 5, 6.1, 6.6, and 7 of this user's manual.

1. **The Main Menu and SubMenus (See Section 5 for more details)**
   These menus are used for selecting the variable and algorithm to assign to a mouse-button-key combination, select color maps, and manage the visualization elements (Table 2 and Section 5). Note that the Scalar Variables listed will vary depending on what variables are loaded into the program. In the table below, the variables are listed assuming a spherical geometry data set with three components of velocity and scalars 1 to N and M were read in from the command line.

2. **Default Mouse & Button Assignments (See Section 6.1 for more details)** (press and hold)

| | |
|---|---|
| Right mouse button | Main Menu |
| Left mouse button: | |
| If mouse is pointing at a Parameter or Tool Dialog Box | Interact with Dialog Box |
| If mouse is pointing within the data set | Rotating |
| 'z' key (or middle mouse button, if available) | Panning |
| Left mouse button and z key | Zooming |
| Left mouse button and z key and left-shift key | Dollying |

3. **Tool Selection Menu and Submenus (See Section 6.6 for more details)**
   Learn to choose a tool that will bind with variable and algorithm selections from the Main Menu (Tables 2 and 3). Note, DraggingTool and NavigationTool has the same submenus as the LocatorTool. The submenus of the UserInterfaceTool and UtilityTool are not described because these are advanced features.

4. **Binding a Tool to a Mouse-Button Combination (See Section 7 for more details)** (press and release)

   (a) Choose Variable from the Main Menu (e.g., temp)

   (b) Choose Algorithm from the Main Menu (e.g., Seeded Slice)

   (c) Select Color Map from the Main Menu (e.g., Saturation Palette –>Rainbow)

   (d) Bind Mouse-Button-Key combination to visualization tool.
   Example: press control, then s-key; select LocatorTool–>ScreenLocatorTool; this binds the combination *Ctrl and s-key* to the tool *Seeded-Slice-Temperature*.

   (e) Explore data set.
   Example: slide mouse over data set; press control, then s-key and hold; release s-key to keep visualization actively updating while rotating/panning/etc.; press and release s-key to complete and keep visualization element; repeat for next slice....

   (f) Unbinding a Tool from a Mouse-Button-Key combination.
   Example: 1. Slide mouse over red rectangle. 2. Press Ctrl-key then s-key.

Table of the Submenus Contained within the 3D Visualizer Main Menu

| Submenu | Functionality |
|---|---|
| **Rendering Modes** | |
| Bounding Box | Displays the edges of the box bounding the limits of the data set. |
| Grid Outline | Displays the edges of the the actual mesh grid. |
| Grid Faces | Displays the mesh grid faces on the exterior of the mesh domain. |
| Grid Cells | Display each individual cell outline. |
| Draw Earth Model | Displays an image of the surface of the Earth with the data set. |
| **Scalar Variables** | |
| velo Colatitude | Displays the colatitude (latitude depending on format) velocity component. |
| velo Longitude | Displays the longitudinal velocity component. |
| velo Radius | Displays the radial velocity component. |
| velo X | Displays the X velocity component (for cartesian grid). |
| velo Y | Displays the Y velocity component (for cartesian grid). |
| velo Z | Displays the Z velocity component (for cartesian grid) |
| velo Magnitude | Displays the velocity magnitude. |
| scalar variable 1 | Displays the scalar variable loaded in such as temperature. |
| scalar variable N | Displays the scalar variable loaded in such as temperature. |
| $\log_{10}$(scalar variable M) | Displays the log of a scalar variable loaded in if specified. |
| **Algorithms: (*see* sections 2.2.2 and 2.2.3)** | |
| Cutting Plane | Displays only data that is behind the cutting plane. |
| Evaluate Scalars | Evaluates the magnitude of a scalar variable anywhere in the data set. |
| Seeded Slice | Creates colored, planar slices through the data at any desired orientation. |
| Seeded Isosurface | Creates seeded surface of constant magnitude starting from any point in grid. |
| Volume Renderer | Generates semi-transparent volume of scalar data. |
| Evaluate Vectors | Evaluates the magnitude of a vector component anywhere in the data set. |
| Arrow Rake | Displays a rake of velocity vectors. |
| Streamline | Generates streamlines from velocity data starting at any point in grid. |
| Streamline Bundle | Generates bundles of streamlines from velocity data at any point in grid. |
| **Show Element List** | Displays separate submenu listing current elements (toggle on/off) |
| **Clear Visualization Elements** | Clears *all* elements from active memory. |
| **Color Maps** | |
| Create Luminance Palette | Progression from black to white with color hue options. |
| Create Saturation Palette | Progression from colors to their compliments. Also rainbow. |
| Show Color Bar | Displays color bar. Magnitudes correspond to toggled scalar variable. |
| Show Palette Editor | Displays color palette window, where palette can be adjusted. |
| **Vrui System** | |
| View | Contains options to load, save, and restore view. |
| Create Input Device | Creates a mouse input device *(advanced feature).* |
| Destroy Input Device | Destroys a mouse input device *(advanced feature).* |
| Show Scale Bar | Displays a scale bar *(not yet implemented).* |
| Quit Program | Quits 3D Visualizer. |

Table 2: This is a compilation table of all of the submenus contained within the 3D Visualizer Main Menu. These submenus menus are used for selecting the variable and algorithm to assign to a mouse-button-key combination, select color maps, and manage the visualization elements. They are described in detail in Section 5.

Tool Selection Menu and Submenus

| **Submenu** | **Functionality** |
|---|---|
| **LocatorTool** | Notify Vrui that an action is to take place at a particular position in 3D space |
| SixDofLocatorTool | Select together with mouse tool to create a LocatorTool connected to the mouse. |
| WaldoLocatorTool | *advanced feature* |
| ScreenLocatorTool | Directly connects LocatorTool to the mouse. |
| **DraggingTool** | Similar to LocatorTool, but selects an object when button is pressed and drags the object while button stays pressed. |
| **NavigationTool** | Rotating, panning, zooming and dollying. Default buttons for these activities are listed above. |
| **TransformTool** | |
| MouseTool | Creates a virtual pseudo-3D input device pointing at the screen-plane at mouse's position on the screen-plane. |
| WaldoTool | *advanced feature* |
| OffsetTool | *advanced feature* |
| ClutchTool | *advanced feature* |
| **UserInterfaceTool** | *advanced feature* |
| **Utility Tool** | *advanced feature* |

Table 3: This is a compilation table of all of the submenus contained within the 3D Visualizer Tool Selection Menu. These submenus menus are used for assigning tools. They are described in detail in Section 6.6.

# 5 The 3D Visualizer Main Menu

The Main Menu is the navigation center for 3D Visualizer. From it you can select the types of data you want to view such as scalar or vector variables, select the types of visualization algorithms needed to view the data such as seeded slices and isosurfaces, and display the color palette information. There are nine buttons on the 3D Visualizer main menu (Figure 2). The buttons with white open arrow to their right indicate that they lead to submenus. Recall that the menus (and dialogue boxes) in 3D Visualizer are dynamic, that is they appear on the screen when you click the correct mouse button combination. The default button to open the main menu is the right mouse button. Holding down the right mouse button will make the main menu appear on the screen. In general, this mouse button combination can be defined by the user using the Vrui configuration file (Vrui.cfg).

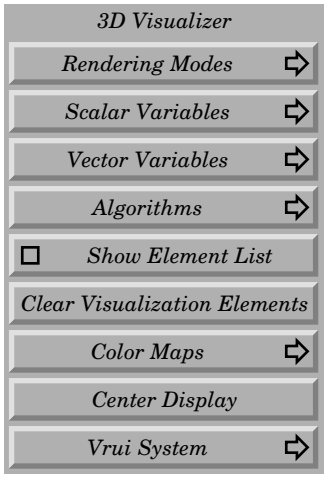



Figure 2: 3D Visualizer Main Menu. Hold down the *right* mouse button to display this menu on the screen.

## 5.1 Rendering Modes

At the top of the Main Menu is the **Rendering Modes** button (Figure 2). Holding down the "Rendering Modes" button with the right mouse button will make the Rendering Modes submenu appear on the screen next to the Main Menu. The Rendering Modes sumenu contains options for how to render the data domain. These options are given in Table 4. The default rendering mode is the Bounding Box, which displays a rectangular outline of the data domain. The Grid Cells option can take a long time to display and is not recommended for large data sets. Note, a data set and Earth Surface image will only be properly aligned if reading in data with longitude, latitude, and radial coordinates.

| Rendering Modes Submenu | |
|---|---|
| Rendering Modes Submenu Option | Functionality |
| Bounding Box | Displays the edges of the box bounding the limits of the data set. |
| Grid Outline | Displays the edges of the the actual mesh grid. |
| Grid Faces | Displays the mesh grid faces on the exterior of the mesh domain. |
| Grid Cells | Displays each individual cell outline. |
| Draw Earth Model | Displays an image of the surface of the Earth with the data set. |

Table 4: Holding down "Rendering Modes" on the Main Menu (right mouse button) will open a submenu that contains several options for viewing the data grid.

## 5.2 Scalar Variables and Vector Variables

Next on the Main Menu is the **Scalar Variables** button (Figure 2). Holding down "Scalar Variables" with the right mouse button will make the Scalar Variables submenu appear on the screen next to the Main Menu. Scalar Variables contains a list of the scalar variables that were loaded in with your data. A list of scalar variables loaded in for an example data set is shown in Figure 3 and Table 5. To select a scalar variable from this list, you hold down the right mouse button over the scalar variable you want to display. A highlighted green diamond next to the variable name indicates the scalar variable currently chosen for display.

**Vector Variables** follows Scalar Variables on the Main Menu (Figure 2). Holding down "Vector Variables" with the right mouse button will make the Vector Variables submenu appear on the screen next to the Main Menu. Vector Variables contains a list of the vector variables that were loaded in with your data. If velocity vector data are loaded in a model with spherical coordinates, the veloX, veloY, and veloZ will be displayed on the submenu but can be ignored. To select a vector variable from this list, you hold down the right mouse button over the vector variable you want to display.

Scalar Variables Submenu

| Scalar Variables Submenu Option | Functionality |
|---|---|
| velo Colatitude | Displays the colatitude (latitude depending on format) velocity component. |
| velo Longitude | Displays the longitudinal velocity component. |
| velo Radius | Displays the radial velocity component. |
| velo X | Displays the X velocity component (for cartesian grid). |
| velo Y | Displays the Y velocity component (for cartesian grid). |
| velo Z | Displays the Z velocity component (for cartesian grid) |
| velo Magnitude | Displays the velocity magnitude. |
| scalar variable 1 | Displays the scalar variable loaded in such as temperature or $V_p$. |
| scalar variable N | Displays the scalar variable loaded in such as temperature or viscosity. |
| $\log_{10}$(scalar variable M) | Displays the log of a scalar variable loaded in if specified. |

Table 5: Holding down "Scalar Variables" on the Main Menu (right mouse button) opens a submenu that contains the scalar variables loaded into 3D Visualizer, such as velocity, temperature, and the $\log_{10}$(viscosity) (Figure 3).

## 5.3 Algorithms

**Algorithms** is the next button on the Main Menu (Figure 2). The Algorithms submenu is important and will be highly used by you. The Algorithms submenu contains a list of the algorithms by which you can view your data. This menu contains algorithms that will generate Visualization Elements (such as a seeded slice), and it also contains algorithms that do not create visualization elements, but instead interact with existing visualization elements (such as the cutting plane). Holding down "Algorithms" with the right mouse button will make the Algorithms submenu appear on the screen next to the Main Menu. To select an algorithm, you hold down the right mouse button over the individual algorithm you want to use. The highlighted green diamond indicates the active algorithm. A list of the algorthims used by 3D Visualizer is shown in Figure 3 and Table 6. In addition each of these visualization elements and algorithms are explained in more detail in Sections 2.2.2 and 2.2.3

## 5.4 Show Element List

Next on the Main Menu is **Show Element List** (Figure 2). Holding down "Show Element List" with the right mouse button will make the Show Element List submenu appear on the screen as a small window. You can move this window to somewhere else on the screen with the left mouse button. Show Element List contains a list of the visualization elements that you have drawn so far. The visualization elements displayed can be toggled on and off by selecting them on this list using the left mouse button. This is useful once many slices or isosurfaces have been made and the viewing space becomes cluttered.

Algorithms Submenu

| Algorithms Submenu Option | Functionality |
|---|---|
| Cutting Plane | Displays only data that is behind the cutting plane. |
| Evaluate Scalars | Evaluates the magnitude of a scalar variable anywhere in the data set. |
| Seeded Slice | Creates colored, planar slices through the data at any desired orientation. |
| Seeded Isosurface | Creates seeded surface of constant magnitude starting from any point in grid. |
| Volume Renderer | Generates semi-transparent volume of scalar data. |
| Evaluate Vectors | Evaluates the magnitude of a vector component anywhere in the data set. |
| Arrow Rake | Displays velocity vector starting at points on a planar grid. |
| Streamline | Generates streamlines from velocity data starting at any point in grid. |
| Streamline Bundle | Generates bundles of streamlines from velocity data at any point in grid. |

Table 6: Holding down "Algorithms" on the Main Menu (right mouse button) opens submenu containing scalar variables loaded into 3D Visualizer (Figure 3).

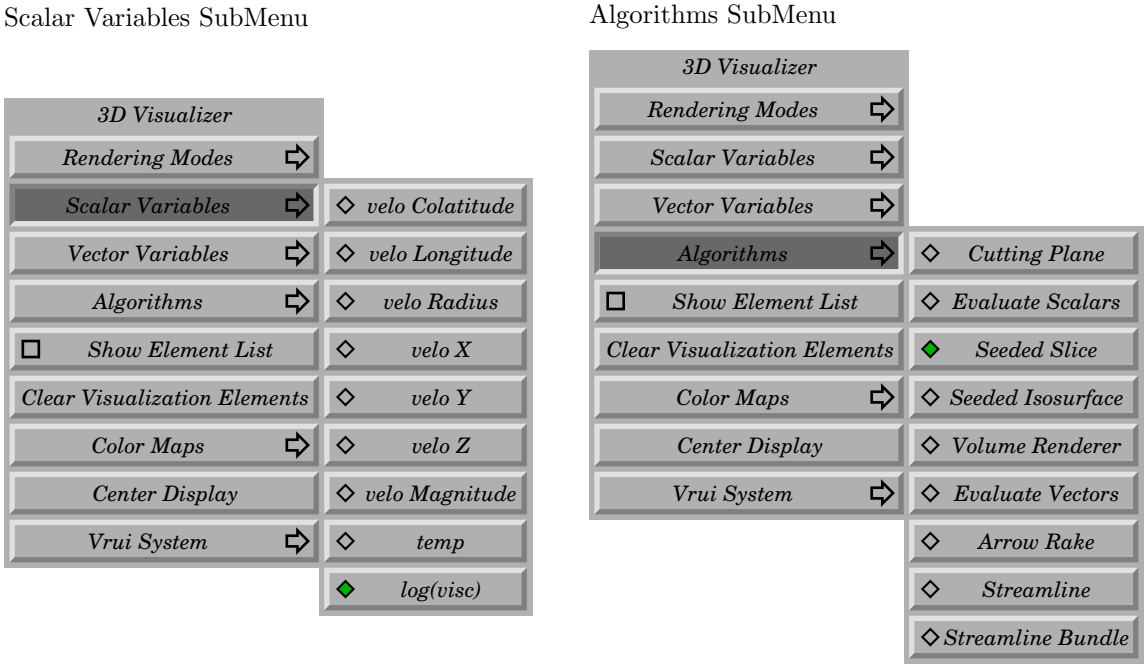



Figure 3: Main menu with Scalar Variables submenu options (*left*) and Algorithms (*right*) displayed. A highlighted green diamond next to an algorithm indicates which algorithm is currently chosen for display.

## 5.5 Clear Visualization Elements

After Show Element List on the Main Menu is the **Clear Visualization Elements** button (Figure 2). Holding down "Clear Visualization Elements" with the right mouse button will *clear all* visualization elements from active memory. This will not erase the actual data, it just removes all of the isosurfaces and seeded slices you had selected from active memory. This is useful if you want to start fresh in viewing your data. *Note* there is not an "undo" button. So, if you cleared the visualization elements, but did not mean to, you will have to go ahead and regenerate them.

## 5.6 Color Maps

Next on the Main Menu is **Color Maps** (Figure 2). Holding down "Color Maps" with the right mouse button will make the Color Maps submenu appear on the screen next to the Main Menu (Figure 4 and Table 7). The submenu allows the user to interactively set the color palette used with color slices and isosurfaces, for example.

Color Maps Submenu

| Color Maps Submenu Option | Functionality |
|---|---|
| Create Luminance Palette | Progression from black to white with color hue options. |
| Create Saturation Palette | Progression from colors to their compliments. Also rainbow. |
| Show Color Bar | Displays color bar. Magnitudes correspond to toggled scalar variable. |
| Show Palette Editor | Displays color palette window, where palette can be adjusted. |

Table 7: Holding down "Color Maps" on the Main Menu (right mouse button) opens a submenu that contains the submenues for the Luminance and Saturation color palettes, and the options to display a Color Bar and the Color Palette Editor.

There are two classes of standard palettes:

**Luminance:** A luminance palette is a progression from black to white with a detour into a particular hue. Luminance palettes are for data with high-frequency variation, where you want to see details.

**Saturation:** A saturation palette is a progression from a color to its complement, with constant luminance. Saturation palettes are for low-frequency variation, where you want to see overall trends. There is also a rainbow color choice located within the saturation color palette.

These can be selected by holding down the right mouse button over the **Create LuminancePalette** and **Create SaturationPalette** options in the Color Maps submenu (Figure 4).

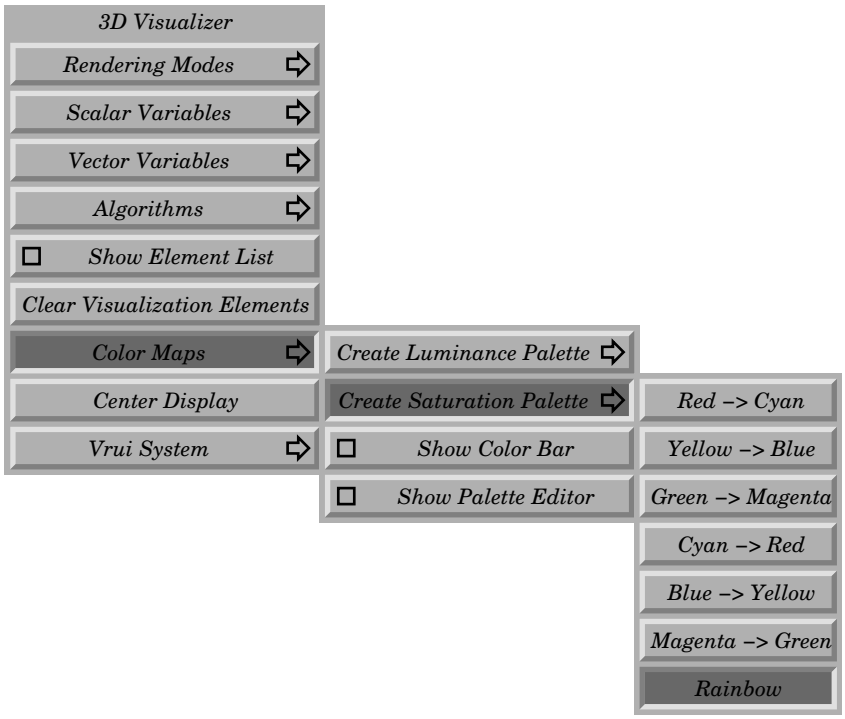



Figure 4: Main menu (*left*) with Color Map submenu (*middle*) and Create Saturation Palette choice of Rainbow (*right*) selected.

You can create some slices or isosurfaces, and then experiment with the palettes afterwards. Changes take effect immediately. For temperature values, we recommend the cyan -> red palette, it has a perceptual progression from cold to warm, as, to a lesser extent, the blue -> yellow palette.

The remaining two options in the Color Maps submenu are **Show Color Bar** and **Show Palette Editor** (Figure 4 and Table 7). Show Color Bar displays a color bar on the screen that corresponds to the toggled scalar variable. The Show Palette Editor option brings up a box oshowing the current color palette. From here, the tie points can each be moved within the color spectrum to adjust the gradient between colors. To add a tie point to the color palette, use the *left* mouse button and click on the diagonal line that extends across the color palette editor dialogue box.

## 5.7 Center Display

Second to last on the Main Menu is **Center Display**. Holding down "Center Display" move the data back to the center of the screen and resizes the data to the initial zoom level.

## 5.8 Vrui System

The last option on the 3D Visualizer Main Menu is **Vrui System**. Holding down "Vrui System" on the Main Menu with the right mouse button will make the Vrui System submenu appear on the screen next to the Main Menu (Table 8). The Vrui System submenu contains a list of options to interact with the program. The first of these is **View**. Holding down "View" with the right mouse button will cause a submenu to appear that allows you to load, save, and restore a 3D Visualizer screen view. The **Destroy Input Device** option will destroy the input device that you created. The **Quit Program** option quits the 3D Visualizer program.

Vrui System Submenu

| Vrui System Submenu Option | Functionality |
|---|---|
| View | Contains options to load, save, and restore view. |
| Create Input Device | Creates a mouse input device (advanced feature). |
| Destroy Input Device | Destroys a mouse input device (advanced feature). |
| Show Scale Bar | Displays a scale bar (not yet implemented). |
| Quit Program | Quits 3D Visualizer. |

Table 8: Holding down "Vrui System" (right mouse button) opens a submenu that contains the options that allow you to interact with the Vrui system functions.

# 6 Using Vrui Applications

We recommend viewing the movie on the software download page, which shows 3D Visualizer used on a desktop computer. This will give you a better idea of how to interact with the menus and use the visualization tools that are describe in this section.

## 6.1 Viewpoint Navigation with the Mouse and Keyboard

The default Vrui application user interface on the desktop uses the mouse (assuming a two-button mouse with scroll wheel) and the left shift key for navigation. Navigation can be understood by imagining a "virtual trackball", where an imagined free-spinning sphere is centered on the screen, and can be dragged with the left mouse button. All objects in the scene react to dragging as if they were attached to the imaginary sphere. To aid navigation in 3D scenes, the navigation tool displays a 3D crosshair in the screen plane when it is active. The crosshair is mostly used to judge whether 3D objects are located in front of or behind the screen plane.

Figure 5 shows the four navigation modes (rotating, zooming , panning, and dollying) and how they are mapped to the mouse buttons/keys in the Vrui default configuration for desktop environments. In Figure 5, the dark-gray shaded buttons indicate the buttons to be pressed (and held) down to achieve the desired effect. In addition, the arrows in Figure 5 indicate the motion with which you (the user) slide the mouse as the appropriate buttons are pressed down to achieve the desired navigation. *Note* that the view in the figure is looking down at the computer and desk, so the gray line indicates the front of the computer monitor.

In addition to the virtual trackball, the navigation offers panning in the screen plane, dollying, which translates the object orthogonal to the screen plane, and zooming, which scales the the scene with respect to the screen center. Table 9 lists each navigation mode and the default mouse-button and key combinations. Panning is scaled such that 3D objects directly in the screen plane move exactly with the mouse cursor. Zooming out (reducing the scale of the scene) is achieved by moving the mouse up (forward) while pressing the Left mouse button and z-key, whereas zooming in (increasing the scale of the scene) is achieved by moving the mouse down. The center point of scaling is the screen center (the center of the 3D crosshair). This means that objects behind the screen plane move farther away when zooming in, which might make it appear as if they are not becoming bigger because scaling is cancelled out by perspective foreshortening. When scaling to "zoom in" on a particular object, it is best to first bring that object, or a point of interest on that object, to the screen plane. A mouse wheel can also be used to zoom. Rolling the mouse wheel up or down will zoom out or in, respectively, by a fixed factor.

As opposed to zooming, dollying moves the entire scene towards or away from the viewpoint. Due to perspective foreshortening, dollying has a similar effect to zooming, but is a fundamentally different operation. When the left mouse button, the z key, and the left shift key are pressed simultaneously, dragging the mouse up will dolly out, i.e., move the scene farther away from the viewpoint, and dragging the mouse down will dolly in, i.e., move the scene closer to the viewpoint. The 3D crosshair can be used to judge when dollying brings an object, or a point of interest

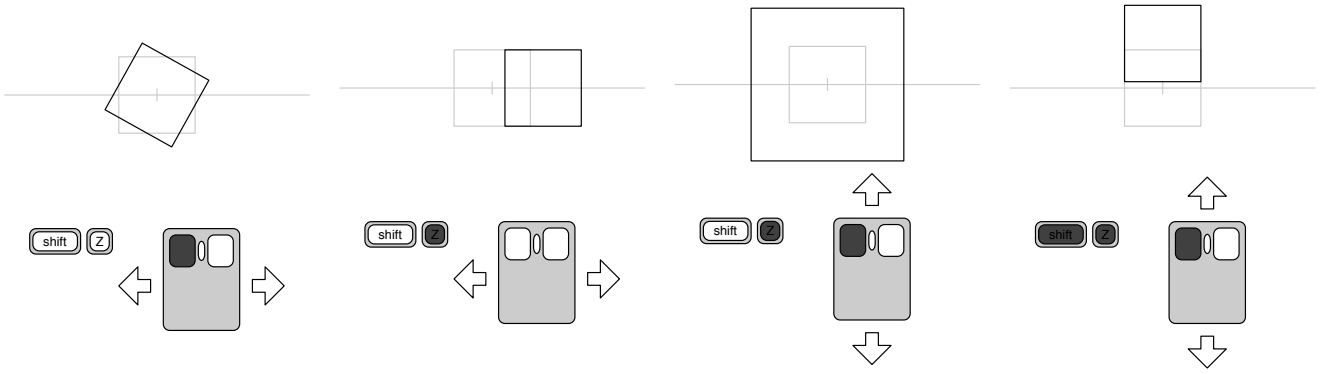



Figure 5: Four navigation modes in the Vrui default configuration for desktop environments. Diagrams show desktop from above, with screen plane denoted by horizontal grey line, and screen center (center for rotation and zooming) denoted by the grey tick mark. Buttons/keys shaded dark gray indicate what the user holds down.

The Four Navigation Modes

| Tool Name | Mouse-Key Combination | Type of Navigation |
|---|---|---|
| Rotating | Left mouse button | Pressing only the left mouse button will rotate the scene around the screen center. |
| Panning | z key (or middle mouse button) | Pressing only the z key button and moving the mouse will pan the scene inside the screen plane. |
| Zooming | Left mouse button and z key | When the left mouse button and the z key are pressed simultaneously, dragging the mouse up will zoom out, i.e., reduce the scale of the entire scene, and dragging the mouse down will zoom in, i.e., increase the scale of the entire scene. |
| Dollying | Left mouse button and z key and left shift key | As opposed to zooming, dollying moves the entire scene towards or away from the viewpoint. |

Table 9: All mouse-key combinations are kept pressed down while the navigation is being done (i.e., press and hold down the left mouse button to rotate; do not click and let go of the left mouse button). Multiple mouse buttons and/or keys listed with *and* indicates that both or all three buttons and/or keys must be held down simultaneously.

on an object, into the screen plane: an object or portion of an object that is in front of the screen plane will obscure the cross-hairs. The mouse wheel in combination with the left shift key can also be used to dolly. Rolling the mouse wheel up or down with the left shift key pressed will dolly out or in, respectively, by a fixed distance. Continuous dollying using the mouse wheel is a convenient way to "fly through" a 3D scene.

The default mapping is intended for systems without a middle mouse button, or where the middle mouse button cannot be used by applications without changes to the windowing system (such as Mac OS X). If the system has a usable middle mouse button, it is recommended to use the middle mouse button in place of the z key. This can easily be achieved by changing one line in the Vrui configuration file (see Appendix B).

## 6.2 Simplified Mouse Navigation Tool

The multiple buttons and keys used by the standard mouse navigation tool can be confusing for new users. To help them get used to Vrui's navigation metaphor, Vrui offers a simplified mouse navigation tool using only a single mouse button and a dialog window to switch between navigation modes (rotating, panning, etc.). This tool can be associated with any desired mouse button by selecting a "MouseDialogNavigationTool" from the "NavigationTool" submenu in the tool selection menu (see Section 6.6). When a MouseDialogNavigationTool is assigned to a button, it will open a dialog box in the center of the screen. The dialog contains four radio buttons for the different navigation modes, and a toggle button to turn the navigation crosshair on or off. Users can interact with the dialog as described in Section 6.4. Each of the tool's four navigation modes works exactly the same way as the corresponding mode in the standard mouse navigation tool.

## 6.3 Menu Interaction

Most Vrui applications contain a main menu that appears dynamically on the screen when indicated by the user (Figure 2). Whereas the left mouse key is involved in panning, zooming and general mouse navigation, the right mouse key is used to access the main menu. The main menu can be opened by pressing and holding the right mouse button. Although the menus (and all other widgets) are represented as 3D objects, they behave almost exactly like regular 2D menus in desktop mode. To select a menu entry, press the right mouse button, move the mouse to the desired menu option (until the menu option becomes highlighted), and then release the right mouse button.

## 6.4 Dialog Box Interaction

Many Vrui applications contain stand-alone dialog boxes to change settings inside the application. These dialog boxes are usually brought up by selecting options in an application's main menu (such as "Show Render Dialog"), and

appear in the screen plane, at the position the selection was made. Dialog boxes can usually be closed by deselecting the associated option in the main menu. Interacting with dialog boxes (and the widgets they contain) is mapped to the left mouse button. If the left mouse button is clicked while over a dialog box, it interacts with the dialog box; if it is clicked anywhere else, it is used for navigation as explained above. Vrui dialog boxes behave almost identically to regular 2D dialog boxes and widgets, e.g., dialog boxes can be moved by dragging the blue title bar.

## 6.5 Available Buttons in Desktop Mode

Vrui applications do not preassign mouse-button-key combinations to Tools used for creating visualization elements: instead, a large number of buttons are made available and the user chooses which buttons to assign to a particular Tool. Besides the buttons/keys used for navigation, menu interaction, and dialog box interaction, the following buttons are available in the default configuration: left/middle/right mouse button, left shift key, Z key, the Q, W, A, S, and D keys, the number keys on the main keyboard, the Tab key, the number keys on the numeric key pad, and the Enter key on the numeric keypad. More buttons/keys can be made available by modifying the Vrui configuration file. The left ctrl and alt keys serve as modifier keys: pressing any combinations of these gives access to an entire new set of buttons, referred to as button-planes. In other words, the Vrui default configuration has 32 buttons in each button-plane, and four button-planes, totalling 128 available buttons. All these buttons can be assigned to Vrui or Vrui application functions using the dynamic tool assignment mechanism described below.

**"Sticky" Keys:** When switching between planes of buttons using any combination of modifier keys (such as left alt or left ctrl), any buttons that were pressed in the previous plane will stay pressed until the plane is activated the next time. This allows one to "hold" buttons from different planes while working with the same buttons in other planes. For example, by pressing and holding the left ctrl key, then pressing and holding the Tab key, and then releasing the left ctrl key (while still holding the Tab key pressed), the left ctrl and Tab virtual button stays pressed until the next time the left ctrl button-plane is activated by pressing the left ctrl key. This sticky key behavior allows for convenient *tricks* in many Vrui applications. This *sticky-key* option is useful, for example, if you want to place a slice and then translate the data grid through the slice. Note, that this is an adaptation for use on the desk-top: in a VR environment the wand could be physically pulled through the virtual data set to achieve the same effect of dragging a slice through the data set.

**Spinning Animation:** The image can be rotated in the screen by pushing down on the left mouse button and moving the mouse around. However, the image can also be made to spin continuously in any direction. Spinning will occur if you hold the left mouse button down and drag the mouse in the direction you want the image to spin. Let go of the left mouse button before you stop moving the mouse (like your throwing a frisbee) and the image will start spinning. To stop spinning, simply click the left mouse button. The speed of spinning is determined by how fast the mouse is moving when spinning is started.

## 6.6 Tool Management

The basic facility supporting the portability of Vrui applications between VR environments (and the desktop) is the use of so-called "tools" to map input device events such as button or key presses to program functionality. Examples of these tools are shown in the 3D Visualizer Tool Selection Menu (Figure 6). Although Vrui applications start with a default set of tool bindings (which is described in the preceding sections), these bindings can be changed at will.

### 6.6.1 Classes of Tools

The user interactions described above make use of several classes of tools:

**NavigationTool:** A NavigationTool maps input device events and motion to 3D viewpoint navigation. The concrete navigation tool described in the first section is a MouseNavigationTool, connected to the left button the Z key, the mouse wheel, and the left shift key.

**MenuTool:** A MenuTool supports popping up, and interacting with, context menus. The ScreenMenuTool described in the second section always opens menus in the screen plane, and is connected to the right mouse button in the default Vrui desktop configuration.

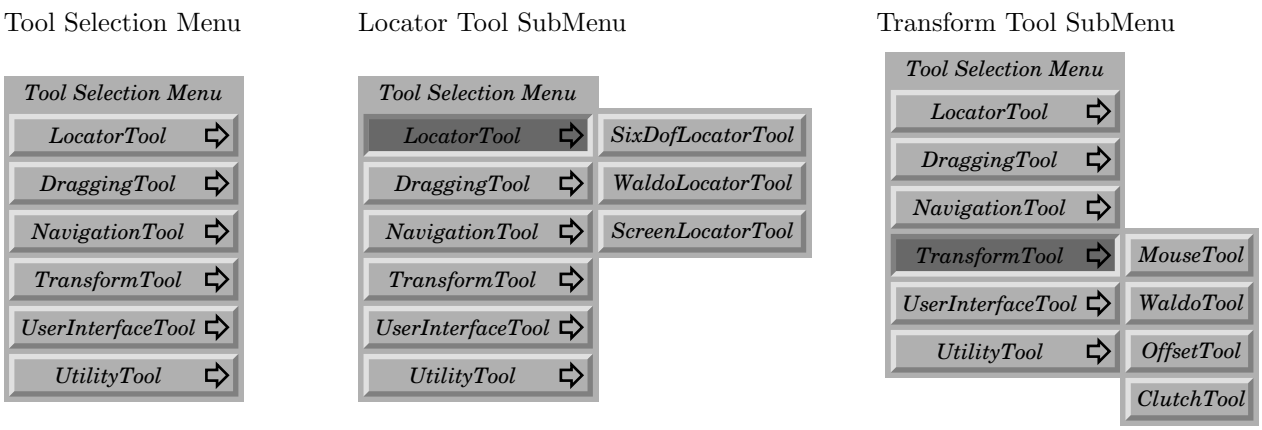



Figure 6: A. The 3D Visualizer Tool Selection Menu. This is accessed by pressing unbound keys (see Section 7) for how to assign tools to keys). B and C show two of the submenus of the Tool Section Menu: The Locator Tool submenu and the Transform Tool submenu.

Other classes of tools are not part of the default set of bindings, but can be bound at runtime to get access to other functions inside the Vrui toolkit itself, or inside Vrui applications:

**LocatorTool:** A LocatorTool is used to notify a Vrui application that an action is to take place at a particular position in 3D space. What exactly this action is is defined by the Vrui application. For example, in the ShowEarthModel example application, a LocatorTool is used to select and query information about earthquakes. The typical way to bind a LocatorTool is to first bind a MouseTool to some mouse button/modifier key combination, and then bind a SixDofLocator tool to the same mouse button/modifier key combination. As a result, the action will take place at the mouse's position inside the screen plane.

**DraggingTool:** A DraggingTool is similar to a LocatorTool in that it works in cooperation with a Vrui application, but it has different semantics. Whereas a LocatorTool simply causes an action at a particular 3D location, a DraggingTool selects an application object at its 3D position when its associated button is pressed, and then drags that object while the buttons stays pressed. DraggingTools are usually bound to the mouse in the same way as LocatorTools, by first binding a MouseTool and then a SixDofDraggingTool.

**TransformTool:** There are four kinds of transform tools in Vrui: a MouseTool, WaldoTool, OffsetTool and ClutchTool. We explain the MouseTool here as it is used often in 3D Visualizer. **MouseTool:** The desktop version of Vrui handles the mouse as a pseudo-3D input device located at the viewpoint, and pointing towards the screen plane. A MouseTool creates a virtual input device moving at the mouse's position inside the screen plane, and can be used to perform 3D interactions by attaching other tools to the virtual mouse device. The MouseTool can be found in the "TransformTool" submenu of the Vrui tool selection menu.

**MeasurementTool:** The MeasurementTool is an example of a high-level tool inside the Vrui toolkit itself. It allows a user to measure positions, distances, and angles in 3D space, either in physical or navigational (model) coordinates. As a Measurement tool measures at 3D positions, it is usually bound to the mouse via a MouseTool. When created, a MeasurementTool will pop up a measurement dialog in the center of the screen. This dialog is used to change the mode of the MeasurementTool, and displays the results of the last measurement operation. The MeasurementTool can be found in the "UtilityTool" submenu of the Vrui tool selection menu.

### 6.6.2 Binding Tools

Tools can be bound to a previously unbound button by pressing that button and selecting the desired tool from the tool selection menu. Tools that require multiple buttons to work, such as the MouseNavigationTool, can be bound by pressing the first button, holding it pressed, and briefly tapping all other buttons in order before the tool is selected from the tool selection menu. For example, to bind a MouseNavigationTool exactly like in the default configuration one would:

1. press (and hold) the left mouse button,

2. press (and release) the Z key,
3. press (and release) the left shift key,
4. roll the mouse wheel in both directions,
5. release the left mouse button while pointing at the MouseNavigationTool in the tool selection menu.

More explanation on binding tools is given in Section 7.

### 6.6.3 Unbinding (Destroying) Tools

Tools can be unbound (destroyed) by pointing the mouse at the "tool destruction zone", the red box in the lower-right hand corner of the screen, and pressing the button (or one of the buttons) the tool is bound to (for example, see the red box in the lower right of Figure 10). The tool will be destroyed immediately; afterwards, a new tool can be bound to the same button or buttons as described above.

*Warning*: any assigned mouse-key combination can be deleted in this way including the navigation mouse-key assignments.

# 7 How to Bind Visualization Elements to Keyboard Buttons

Now that you are familiar with the 3D Visualizer tools and how to navigate with the mouse, you must be eager to start assigning tools and exploring your data. In this section, we provide detailed instructions for how to go about mapping a visualization element to a button/key combination. This is a sequence of steps that you will perform regularly while using the 3D Visualizer program. It may seem unfamiliar at first but after a few times through, you will get the hang of it. Basically, because there are more tool options in 3D Visualizer than there are buttons on your mouse, you have to assign the extra tools you want to use to the keys on the keyboard. This is referred to as binding the tool to the button or keyboard key. Once you have bound the tool, to a particular key on the keyboard, pressing down that key on the keyboard has a desired effect much like pressing down one of the buttons on your mouse. But in this case, you get to decide what function you want the particular key to do, such as draw a seeded slice through your data.

## 7.1 Step 1. Choose your Variable

With the 3D Visualizer screen open:

Hold down the *right* mouse button to activate the Main Menu.
From the Main Menu, hold down the Scalar Variables option.
Then, while still holding down the *right* mouse button, slide the mouse along the screen until the cursor rests above the scalar variable you want to plot, the $\log_{10}$(viscosity) for example (Figure 7).
And then release the right mouse button.

To check if it worked, activate the Main Menu again and then open the Scalar Variables submenu again. The diamond next to the scalar variable you selected, $\log_{10}$(viscosity) in this case, should be highlighted green (Figure 7).

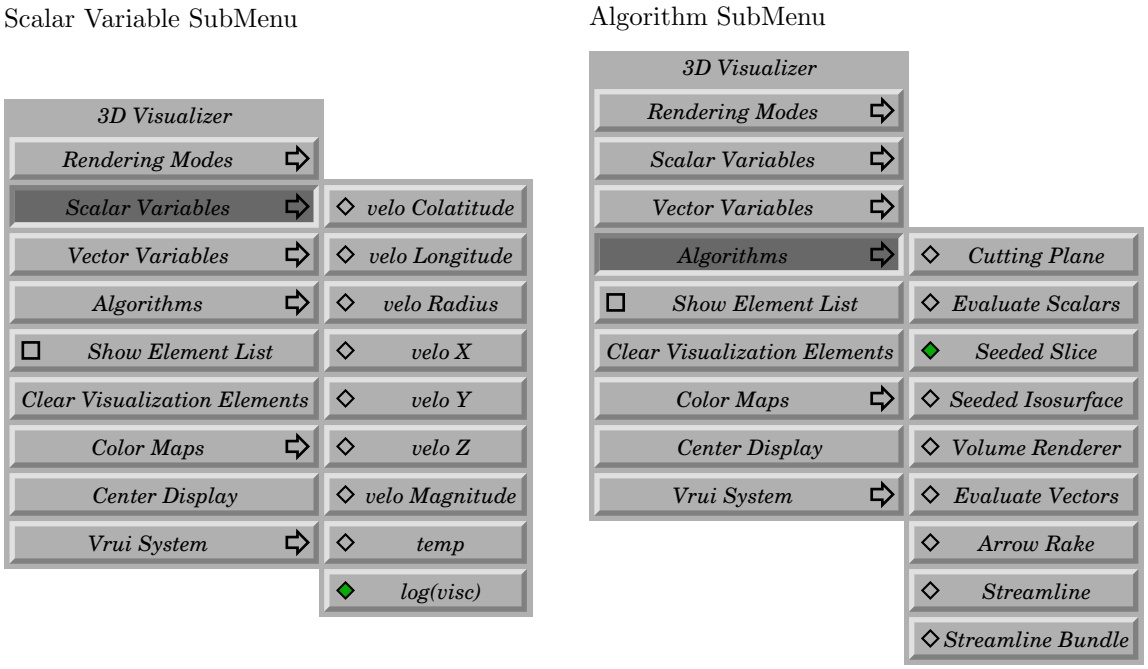



Figure 7: Choose your variable (Step 1). Left image shows Main Menu with Scalar Variables submenu open and the log of viscosity selected. Choose your algorithm (Step 2). Right image shows Main Menu with Algorithms submenu open and the Seeded Slice option selected.

## 7.2 Step 2. Choose your Algorithm

With your variable selected, now decide on the algorithm you want to use to view the variable.

Hold down the right mouse button to activate the Main Menu.
From the Main menu, hold down the Algorithms option (Figure 7).
Then, while still holding down the right mouse button, slide the mouse along the screen until the cursor rests above the algorithm you want to use, seeded slice for example.
And then release the right mouse button.

To check if it worked, activate the Main Menu again and open the Algorithms submenu. The diamond next to the algorithm you selected, seeded slice in this case, should be highlighted green (Figure 7).

## 7.3 Step 3. Select Your Color Scale

To select your color scale:

Hold down the right mouse button to activate the Main Menu.
From the Main menu, hold down the Color Maps option (Figure 8).
Then, while still holding down the right mouse button, slide the mouse along the screen until the cursor rests above the color palette option you want (either Create Luminance Palette or Create Saturation Palette). For this example, pause the mouse over Create Saturation Palette and and submenu will appear (Figure 8).
While still holding down the right mouse button, slide the cursor over the Rainbow option (Figure 8).
And then release the right mouse button.

Next, to display on the screen the color scale you just chose:

Activate the Main Menu again.
Hold down the Color Maps option.
Then, while still holding down the right mouse button, slide the cursor along the screen until it rests above the Show Color Bar option.
A small window with the color bar will appear on the screen.
To move the color bar window to somewhere else on the screen, hold down the *left* mouse button over the color bar window and move it to where you want it.

## 7.4 Step 4. Bind Your Button-Key Combination to the Visualization Tool

To bind your keyboard key(s) to the visualization tool, decide what key combination you want to use. We recommend the following sets of key combinations per visualization tool: left ctrl and s; left ctrl and d; left ctrl and w; left ctrl and a; and left ctrl and q. For this example, use the left ctrl and s keys. There are two ways you can bind the keys (left ctrl and s) to the tool (Figure 9). We recommend the two-step option (option 2).

**Option 1:** Hold down the ctrl and s keys at the same time. As long as those keys are free (unbound to a tool), the Tool Selection Menu will appear on the screen. Keep holding down the ctrl and s keys and slide the cursor over the Locator Tool option. This will open a submenu right next to the Tool Selection Menu (Figure 9, part B). Keep holding down the left ctrl and s keys and slide cursor over and then down to the ScreenLocatorTool, then release s key and then release the left ctrl key. The cursor will turn into small black cross hairs to indicate that the tool is activated and bound to the screen locator tool.

**Option 2:** Since you may have just used the ctrl and s key combination, we will bind this tool to the ctrl and w key combination. Step 1: Hold down the left ctrl and w keys at the same time, the Tool Selection Menu will appear on the screen. Keep holding down the leftctrl and w keys and slide the cursor over the Transform Tool option. This will open a Transform Tool submenu next to the Tool Selection Menu (Figure 9, part B). Keep holding down the left ctrl and w buttons and slide cursor over the Mouse Tool, and then release the buttons.

Step 2: Hold down the left ctrl and w keys again. Keep holding down the left ctrl and w keys and slide the cursor over the LocatorTool option. A submenu will appear. Keep holding down the left ctrl and w buttons and slide the cursor over the SixDofLocator Tool, and then release the w key and then the left ctrl key (Figure 9, part C). The keys should be activated and bound to the tool, however, unlike in option one, the mouse cursor does not become cross hairs.

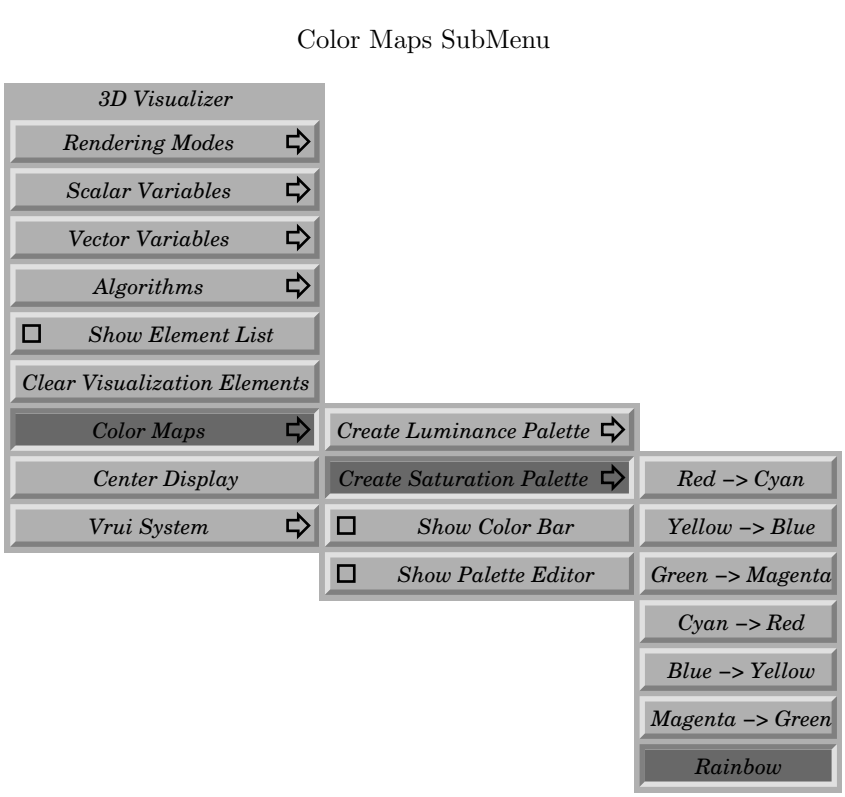



Figure 8: Select your color scale (Step 3). Image shows Main Menu (*left*) with Color Maps submenu open (*middle*) and the Create Saturation Palette submenu also open with the Rainbow color scale selected (*right*).

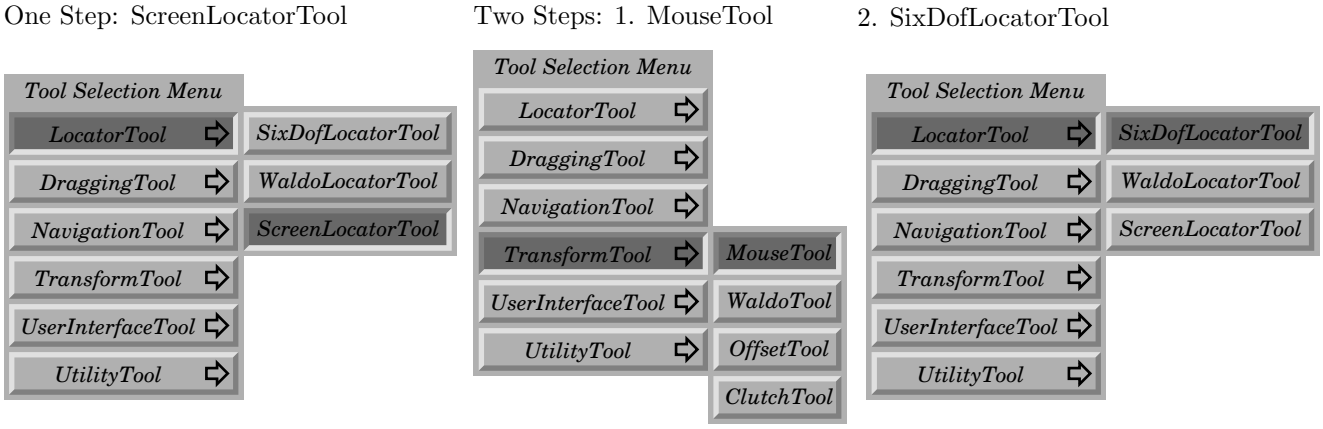



Figure 9: Choose your Button-Key combination (Step 4). Option 1: One-step process that assigns tool to a ScreenLocatorTool selected from the LocatorTool submenu (*left*). C. Option 2: Two-step process that assigns tool by first selecting MouseTool from the TransformTool submenu (*center*), and then selecting SixDofLocatorTool from the LocatorTool submenu (*right*).

## 7.5 Step 5. Explore Your Data Set

Now, slide the mouse over your data set. With the mouse cursor inside your data grid, hold down the left ctrl and s keys (or the left ctrl and w keys depending on which tools you just assigned above). A rainbow colored seeded slice should appear. Use the mouse navigation tools and your newly assigned seeded slice tool to explore the data. As you then move the mouse through the image the visualization element will update (e.g., the color slice will move across the data). To stop scanning the data set and fix the position of the slice, release the key combination.

### 7.6 Step 6. Unbind (Destroy) Your Button-Key Combination

To unbind (or free up) your button-key combinations, with *no* buttons pressed down, slide the cursor over to the red rectangle in the bottom right of your 3D Visualizer window. This is the "total destruction zone". Now, press the left ctrl and s keys together while the cursor is in the red box. This will clear those buttons. (Note, only press down the left ctrl and s keys that you want to clear. Do *not* press down any of the buttons on the mouse itself. Pressing down the buttons on the mouse in the total destruction zone will clear the 3D Visualizer preset mouse navigation tools.) To confirm that your control and s key combination is free, hold down the left ctrl and s keys anywhere on the screen and the Tool Menu should appear again. This indicates the button-key combination is clear and is ready for another tool assignment.

# 8 Tutorial: Vertical Slab Example Data Set

Before you start this Section make sure you have looked over Sections 5, 6, and 7. As described in Section 5, there are a variety of algorithms used by 3D Visualizer to display data. For example, scalar data can be viewed with the volume renderer, as isosurfaces, or as color slices. All isosurface or color slices use seeded algorithms. Seeded algorithms will display the part of the surface or slice closest to the pointer and update this quickly as the mouse is moved around. Once the location of the isosurface or slice is chosen by releasing the assigned key combination, then the rest of the surface or slice will fill in. In addition, the cutting plane can be used to make a virtual cross-section through volume rendered data. Vector data can be viewed using streamlines, streamline bundles, and rakes of arrows. In addition, the individual velocity components can be viewed using the algorithms for scalar data. Recall, that any key combination can, in theory, be chosen to bind a tool (see Section 7). Refer to the Vrui.cfg file for a list the available key combinations. Common key combinations that we use to bind visualization elements to the keyboard keys are: ctrl and s; ctrl and d; ctrl and w; and ctrl and q.

## 8.1 Open 3D Visualizer with the Vertical Slab Example Data

Here we describe how to open 3D Visualizer using an example data set from a finite element model with a vertically dipping slab. The slab is defined by a vertically dipping negative temperature anomaly is located in the center of the data grid. The example is a gridded data set with brick elements. This data set is in a binary format that corresponds to the CitcomCUSphericalRawFile module. The data set includes two scalar fields (temperature and viscosity) and one vector field (velocity) that will be loaded into 3D Visualizer.

1. (On Mac OS X: Start X11.)
2. Open an X11 or Terminal window.
3. Change into your data directory:
   % `cd` */Path/To/Your/Data/Directory/*
4. Execute the tar command to unpack the slab3dB.tar.gz tarball:
   % `tar xfz` */download/path/*`slab3dB.tar.gz`
5. Run 3D Visualizer:
   % *path/to/3DVisualizer/executable/file/*`3DVisualizer -class CitcomCUSphericalRawFile slab3dB 0 -vector velo temp -log visc`

   If you used the default installation and set-up options, this is the line you would type:
   % `/$HOME/src/3DVisualizer/bin/3DVisualizer -class CitcomCUSphericalRawFile slab3dB 0 -vector velo temp -log visc`

This will load a grid file with temperature, viscosity, and vector velocity stored at each vertex. A window will open with a black screen and a white rectangular outline in the center of the window. Loading the data sample data can take from 10 to 60 seconds.

## 8.2 Data Grid Visualization

The first step in exploring a data set is usually to display the data grid. The data grid can be displayed as a cartesian bounding box, as the spherical grid outline, as a grid outline with the element faces displayed on the grid walls, or as the grid outline with all of the cells inside the grid displayed. For large data grids ($10^6$ mesh nodes), we recommend using the grid outline or grid faces option. Do not use the grid cells option (it takes *forever!*).

To display the grid faces as shown in Figure 10:

1. Load the slab3dB example data set as described in Section 8.1.
2. With the 3D Visualizer screen open, activate the main menu by holding down the *right* mouse button.

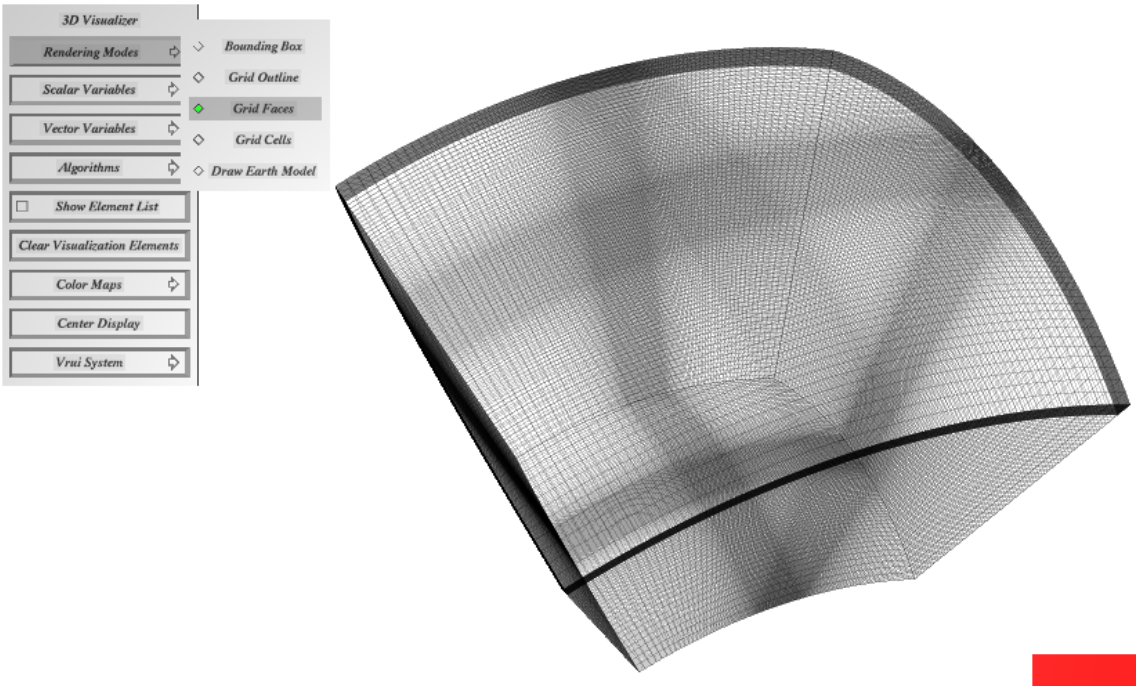



Figure 10: Main menu with Rendering Modes submenu options displayed. Grid faces option is selected. Corresponding multi-resolution model mesh with grid faces on the model walls is shown. Mesh is from Slab3dB example data set. Red box in lower *right* is tool destruction zone, where you unbind the keys (i.e., destroy tools).

3. Keep holding down the *right* mouse button and slide the cursor over "Rendering Modes".
4. A submenu will appear. While still holding down the *right* mouse button, slide the mouse along the screen until the cursor rests over the "grid faces" option.
5. Hold it there for three seconds, then release the *right* mouse button.

You can now use the left mouse button to interact with the model grid. Move the mouse pointer to the point on the data that you want to 'grab'. Then rotate the data grid by pressing the *left* mouse button and moving the mouse pointer side-to-side, up-down or any other direction on the screen. To stop rotating, release the *left* mouse button. To explore the geometry of the grid faces, zoom-in to inspect the grid spacing. To zoom-in, press down the *left* mouse button and z key together and then move the mouse pointer down the screen. Release the buttons to stop zooming. If your mouse has a wheel. You can zoom in and out with the wheel. Note that moving a point of interest in the data to the center of the screen will cause the zooming to center around this point.

## 8.3 Scalar Visualization

Scalar data can be visualized as a seeded slice, seeded isosurface, or with a volume renderer. Recall that a seeded slice is a planar surface cutting through a data set's domain, color-mapped by an arbitrary scalar variable. A seeded isosurface is a surface connecting all points in a data set's domain that have identical values of an arbitrary scalar variable. The volume renderer visualizes an arbitrary scalar variable over the entire domain of a data set, by drawing the domain filled with a semi-transparent color-mapped material. The value of scalar data at a point can also be queried using the measurement tool.

### 8.3.1 Example: Making Color Slices of the Data

To a display color slice of temperature from the example slab3dB data set (Figure 11), we must assign a key combination. Follow closely the steps outlined in Section 7 of this user's manual. If you did not read Section 7, read it, and then return to this section.

Open up the slab3dB data set as described in Section 8.1. Display the grid outline (or grid faces) as described in section 8.2. Now, once you have the grid outline, follow the steps to bind the seeded slice tool as specifically outlined in Step 1 - Step 5 of Section 7 in this user's manual:

Step 1: Choose your Variable (temperature).

Step 2: Choose your Algorithm (seeded slice).

Step 3. Select Your Color Scale (rainbow).

Step 4. Bind Your Button-Key Combination (left ctrl and s) to the Visualization Tool.

Step 5. Explore Your Data Set.

Once you have assigned your key combination, press down the keys (left ctrl and s) and slide the cursor on the screen within the grid outline. Release the key combinations (left ctrl and s) when you have the data oriented as you want them. After you release the keys (left ctrl and s), the slice will remain on the screen. To remove the slice, open the element list from the main menu (*right* mouse button) and deselect the visualization element. Continue to generate slices. Pan, zoom, and rotate the data as described in the viewpoint navigation section of Section 6.

For example, to explore the geometry of the thermal field within your data set, zoom-in to the data. To zoom-in, press down the *left* mouse button and z key together and then move the mouse pointer down the screen. Release the buttons to stop zooming. If your mouse has a wheel. You can zoom in and out with the wheel. Note that moving a point of interest in the data to the center of the screen will cause the zooming to center around this point.

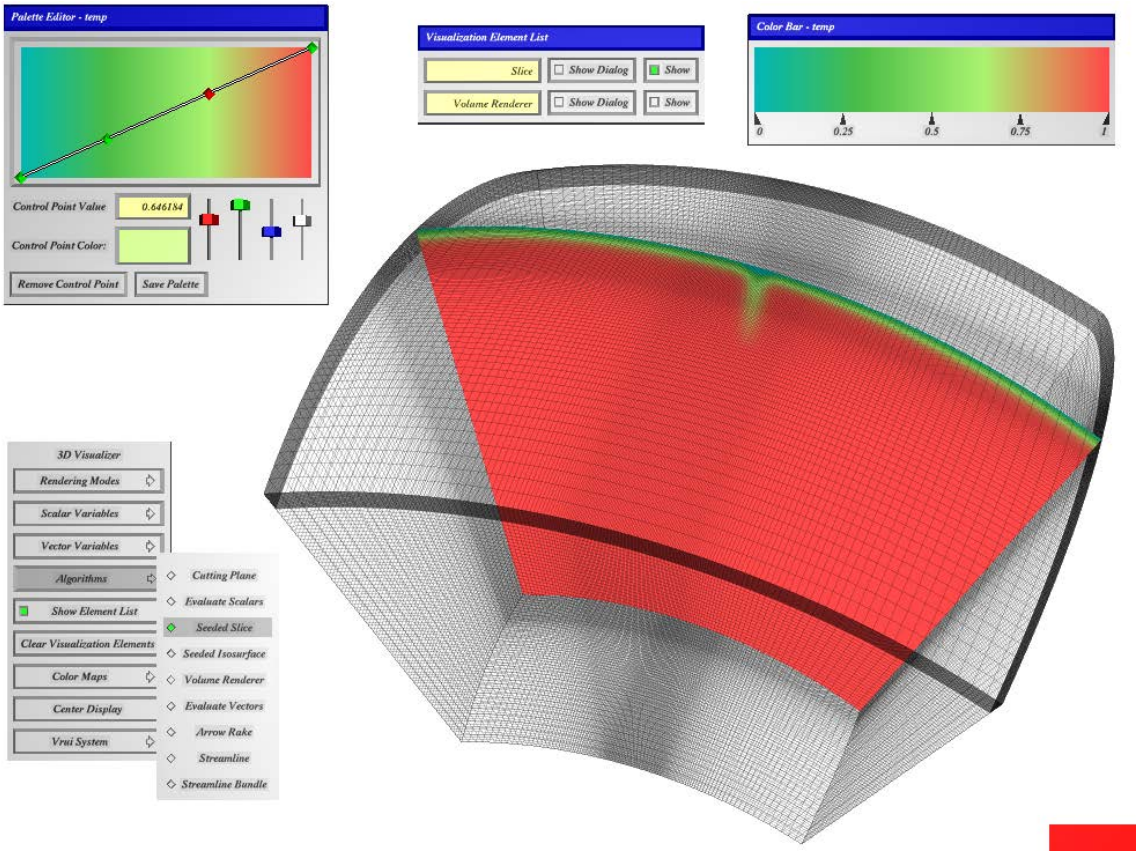



Figure 11: Seeded slice of temperature from example slab3dB data set which contains a vertically dipping negative thermal anomaly in the center of the grid. Also shown are the color bar (upper *right*), the element list dialogue box (upper center), color palette editor dialogue box (upper *left*), and main menu (lower *left*). Use the left mouse button to interact with the dialogue boxes and the right mouse button to interact with the main menu. To add a tie point to the color palette, use the left mouse button and click on the diagonal line that extends across the color palette editor dialogue box.

Move the mouse pointer inside the data volume. A green box should appear highlight the element in which the mouse pointer is located. Using sticky keys, you can down rotate the image around to view the slice from different angles. Follow the same procedure to assign a different key combination to add isosurfaces. An example of an isosurface of temperature is displayed in Figure 12. You could generate an isosurface by now selecting the isosurface algorithm and binding a tool to the left ctrl and w keys, for example.

## 8.4 Vector Visualization

Vector data can be visualized as a stream line, streamline bundle, or as an arrow rake. A streamline follows the path of a massless particle that is advected through the data set's domain by an arbitrary vector variable. A streamline bundle is a set of related streamlines that originated from a small circle. An arrow rake is a local regular 2D grid of arrows, whose directions and lengths visualize an arbitrary vector variable at the vertices of the 2D grid, which can be arbitrarily placed in the data set's domain. In addition, the individual components of the velocity can be visualized using the scalar visualization tools above. For example, an slice of the radial velocity can be displayed. Note that for spherical model, it is the Cartesian components of the velocity that are used to calculate the velocity magnitude. Although the Cartesian components are used in calculating the velocity magnitude,in plotting the individual velocity components on a spherical grid, you plot the spherical components of the velocity.

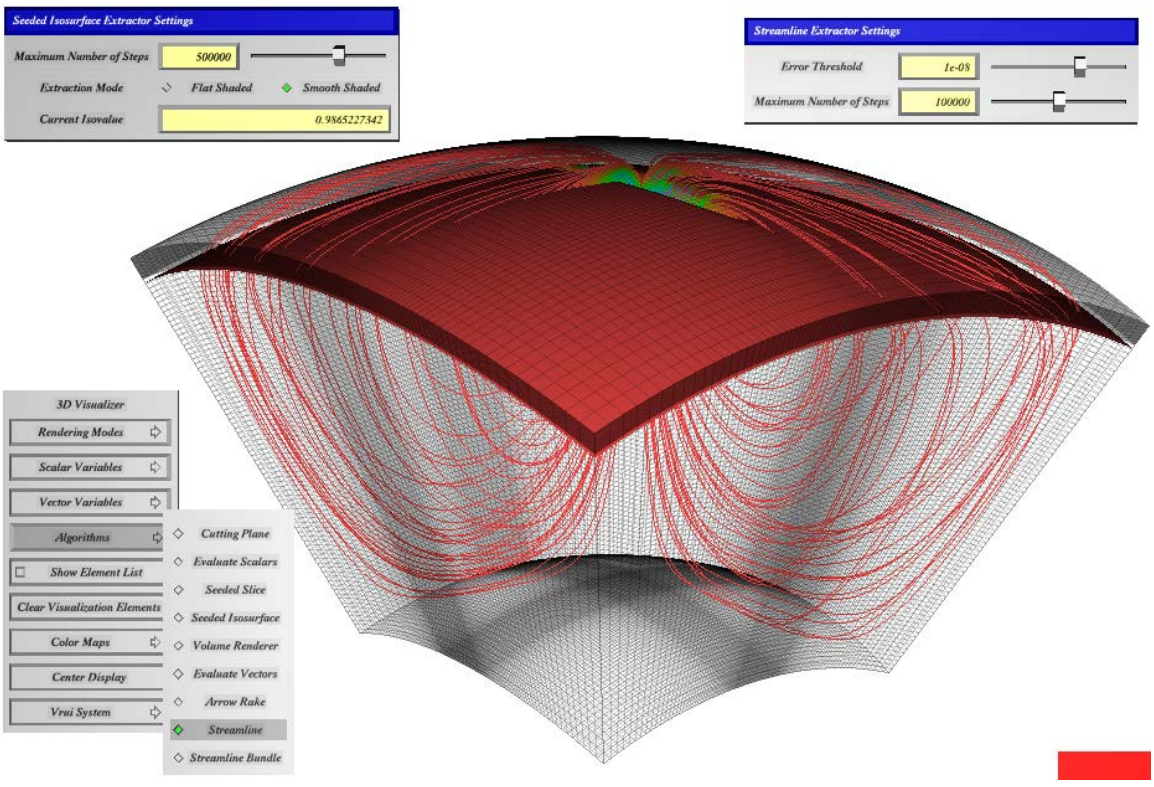



Figure 12: Streamlines (from velocity field) and isosurface of temperature for example Slab3dB data set. Also shown are the streamline dialogue box which contains the threshold and maximum steps slider bars (upper *right*), the seeded isosurface dialogue box which contains the number of steps slider bar (upper *left*), and main menu (lower *left*). Use the *left* mouse button to interact with the dialogue boxes and the *right* mouse button to interact with the main menu.

### 8.4.1 Example: Plotting Streamlines

Open up your data set or the slab3dB data set as described in Section 8.1. Display the grid outline (or grid faces) as described in section 8.2. Now, once you have the grid outline, follow the steps to bind the streamline tool as specifically outlined in Step 1 - Step 5 of Section 7:

Step 1: Choose your Variable (velocity magnitude).

Step 2: Choose your Algorithm (streamline).

Step 3. Select Your Color Scale (rainbow).

Step 4. Bind Your Button-Key Combination (left ctrl and d) to the Visualization Tool.

Step 5. Explore Your Data Set.

Once you have assigned your key combination, press down the keys (left ctrl and d) and slide the cursor on the screen within the grid outline. Release the key combinations (left ctrl and d) when you have the data oriented as you want them. After you release the keys (left ctrl and d), the a streamline will be generated initiating at the point of your cursor on the screen (Figure 12). Drop as many streamlines as you like, by pressing down together your assigned key combination (left ctrl and d) within the model domain and then releasing the keys (left ctrl and d) simulataneously. To remove the streamline, open the element list from the main menu (*right* mouse button) and deselect the visualization element. Pan, zoom, and rotate the data as described in the viewpoint navigation section of Section 6. It is useful to have isosurfaces of temperature or viscosity in the model to better interpret the fluid flow (Figure 12).

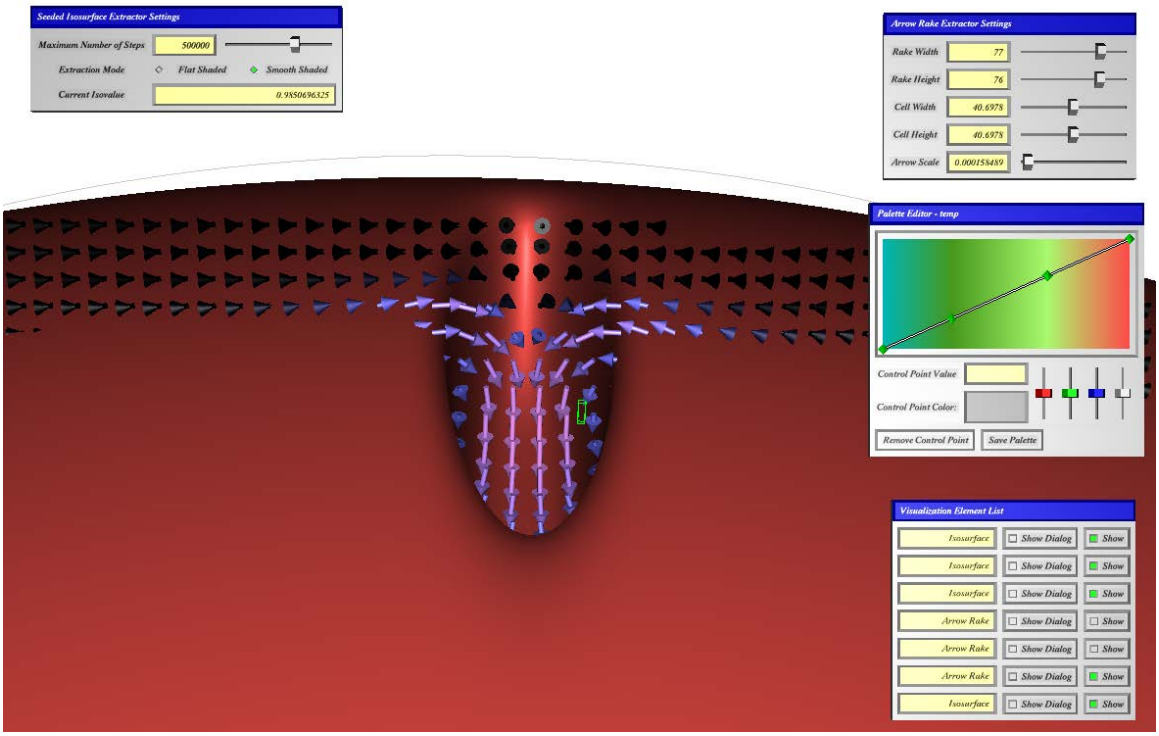



Figure 13: Arrow rake (from velocity field) and isosurface of temperature for example Slab3dB data set. Also shown are visualization element list (lower *left*), the color palette editor (middle *left*), and the arrow rake dialogue box (upper *right*), and the seeded isosurface dialogue box (upper *left*). Use the *left* mouse button to interact with the dialogue boxes.

### 8.4.2 Example: Plotting A Rake of Arrows

Open up your data set or the slab3dB data set as described in Section 8.1. Display the grid outline (or grid faces) as described in section 8.2. Now, once you have the grid outline, follow the steps to bind the ArrowRake tool as specifically outlined in Step 1 - Step 5 of Section 7:

Step 1: Choose your Variable (velocity magnitude).

Step 2: Choose your Algorithm (arrow rake).

Step 3. Select Your Color Scale (rainbow, or what ever you prefer).

Step 4. Bind Your Button-Key Combination (left ctrl and q) to the Visualization Tool.

Step 5. Explore Your Data Set.

Once you have assigned your key combination (left ctrl and q), press down the keys and slide the cursor on the screen within the grid outline. Release the key combinations (left ctrl and q) when you have the data oriented as you want them. After you release the keys (left ctrl and q), a plane of arrows will be generated initiating at the point of your cursor on the screen (Figure 14). Generate as many rakes of arrows as you like, by pressing down together your assigned key combination (left ctrl and q) within the model domain and then releasing the keys simulataneously. To remove the arrow rake, open the element list from the main menu (*right* mouse button) and deselect the visualization element. Pan, zoom, and rotate the data as described in the viewpoint navigation section of Section 6. It is useful to have isosurfaces of temperature or viscosity in the model to better interpret the fluid flow (Figure 14).

The density and size of the arrows generated by the arrow rake algorithm can be optimized for your data set by adjusting the parameters in the arrow rake dialogue box (upper *right* of Figures 14 and 13). The Rake Width is the width of the field of view of the plane of arrows (in the units of your model). The Rake Height is the height of the field of view of the plane of arrows (in the units of your model). The Cell Width is the arrow spacing in the horizontal direction in the plane of the arrow rake. The Cell Height is the arrow spacing in the vertical direction.

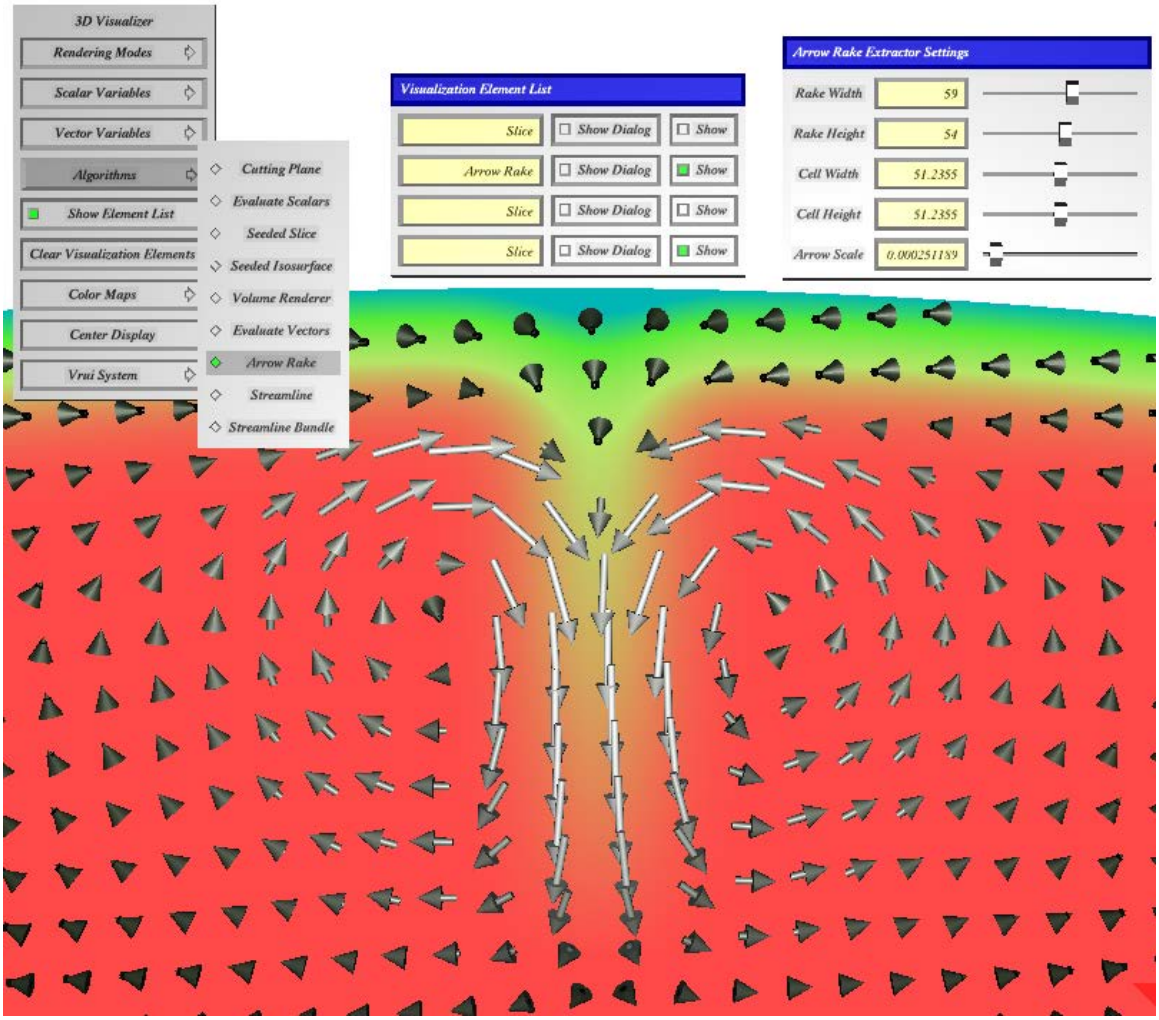



Figure 14: Arrow rake (from velocity field) and seeded slice of temperature for example Slab3dB data set. Also shown are the arrow rake dialogue box (upper *right*), visualization element list (upper middle), and the main menu (upper *left*). Use the *left* mouse button to interact with the dialogue boxes.

Arrow scale scales the length of the arrow. To modify the field of view, density, or scale of the arrows, adjust these parameters in the dialogue box.

## 8.5 How to Export and Import Data Viewpoints and Other Data Fields

During a 3D Visualizer user session, it is sometimes useful to extract data for the current viewpoint or that calculated with a measurement tool. Therefore, there are several options to extract data from a given 3D Visualizer user session. The data extraction files will be written to the directory from which you are running 3D Visualizer.

1. Saving viewpoints
   To save a viewpoint, open the main menu and slide the cursor down to the Vrui system button. Select the save view option. From the Show Palette Editor dialogue box, select Save Palette. By default, the exported viewpoint data files are named: SavedViewpointXXXX.dat

2. Saving color palettes
   To save a color palette, open the main menu and slide the cursor down to the Color Maps button. Select the Show Palette Editor By default, the exported color palette file is named: SavedPaletteXXXX.dat

3. Saving screenshots
   To save a screen shot, simultaneously press the: shift p
   keys. By default, the screenshot files are named: VruiScreenshotXXXX.png/.ppm

To load in a saved viewpoint during a user session, rename the viewpoint file: Viewpoint.dat. Saved color map files are loaded in on the command line.

# A Appendix: Data Format Descriptions and Command Line Flags

## A.1 Generic Data Formats Supported by This Release of 3D Visualizer

In this section we describe in detail the data formats supported by modules packaged in this release of 3D Visualizer, and the command lines to load them into the application. Read the description, and in particular, locate the part in bold face font titled **Module Command Line**. The module name is the word in bold face font that immediately follows **-class**.

### A.1.1 StructuredGridASCII

This module reads scalar- and/or vector-valued data in Cartesian or spherical coordinates, stored as a collection of related ASCII files. This is not necessarily a format used by any existing simulation codes, but it is intended as a guideline on how to write new 3D Visualizer modules, and as a recommended file format for users who need to write gridded 3D data, but have not yet decided on a particular data format. The format is self-describing, flexible, and can easily be adapted for binary data storage to increase read/write performance.

Each data set consists of exactly one *grid file* defining the layout of the data set's grid, and the position of each *grid vertex* (aka *node*) in Cartesian or spherical coordinates, and one or more *variable files*, each defining exactly one scalar or vector variable. Either file type begins with a *header* defining the type and layout of the data contained in the file, followed by a sequence of *vertex attributes* defining vertex positions and data values, with attributes for one vertex in each line. Variable files for scalar values contain one value per vertex, i. e., one value per line. Grid files and variable files for vector values contain three values per vertex (position or direction in Cartesian or spherical coordinates), which are written as three whitespace-separated columns per line. Lines starting with "#" in any of the files are ignored and can be used to annotate files with comments.

**Grid File Format** The grid file's header consists of two lines. The first line defines the data layout, i. e., the number of grid vertices in the $i, j, k$ directions, and the second line defines the coordinate mode, i. e., spherical or Cartesian.

**gridsize *NI NJ NK*** Defines the number of grid vertices in $i, j, k$.

**coordinate *coordinateMode*** Defines the coordinate system mode of the data set. *coordinateMode* must be either **Cartesian** or **spherical**.

After this header follow the positions of the grid vertices, one per line, with each vertex either defined by Cartesian coordinates *x y z* in arbitrary units, or by spherical coordinates *longitude latitude radius*, with longitude and latitude in radians and radius in meters. Longitudes increase from west to east, either from 0 to $2\pi$ or $-\pi$ to $\pi$, latitudes increase from south to north, with the south pole at $-\pi/2$, the equator at 0, and the north pole at $\pi/2$, and radii increase from 0 at the Earth's center to $6378.14 \cdot 10^3$ at the surface (3D Visualizer does not use a flattened geoid). The total number of vertices in the grid file, i. e., the number of lines after the header, is $NTOTAL = NI \times NJ \times NK$.

**Vertex Counting Order** The order of vertex attributes (positions or values) is in Fortran memory order. In other words, when reading through a file from beginning to end, the $i$ index component varies fastest, and the $k$ index component varies most slowly. In spherical coordinate mode, $i$ corresponds to longitude, $j$ corresponds to latitude, and $k$ corresponds to radius. Therefore, vertices are enumerated starting at the deepest, south-westernmost corner, scanning through longitude first, then latitude, then radius. The mapping between $i, j, k$ and $x, y, z$ in Cartesian coordinate mode can be arbitrary; however, 3D Visualizer assumes that the Cartesian direction vectors associated with $i$, $j$, and $k$, respectively, form a right-handed coordinate frame. If a grid's mapping defines a left-handed frame instead, 3D Visualizer will display surfaces with wrong illumination. If this (obvious) problem occurs, it is necessary to "flip" the grid, by either inverting one coordinate axis or swapping the order of two axes.

**Variable File Format** A variable file's header consists of two lines. The first line defines the data layout, identically to the grid file, and the second line defines the type and name of the variable stored in the file.

**gridsize *NI NJ NK*** Defines the number of grid vertices in $i, j, k$. The layout of each variable file must match the layout of the associated grid file.

***variableType variableName*** Defines the type of the stored variable, and its name. *variableType* must be either **scalar** to define a scalar variable, or **vector** to define a vector variable. *variableName* can be an arbitrary descriptive name for the variable stored in the file. This name is used to select variables for visualization inside 3D Visualizer.

After this header follow the data values associated with grid vertices, in the same vertex counting order as used in grid files. In scalar files, each vertex has a single data value; in vector files, each vertex has three vector component values. In Cartesian coordinate mode, each vector is defined directly by its $x$, $y$, and $z$ components; in spherical coordinate mode, each vector is defined by three components *in length units per time unit* along the local longitude, latitude, and radius directions. This is done to ensure compatibility with existing simulation codes storing vector values; it means that the length of a vector in spherical coordinates does not scale with the radius of its associated vertex.

**Module Command Line** The StructuredGridASCII module takes the names of grid and variable files to read from its command line, which is defined as follows:

**-class StructuredGridASCII** *gridFileName variable1 ... variableN*

where *gridFileName* is the fully qualified path name of the grid file, and each *variableI* is either *scalarFileName*, the fully qualified path name of a scalar variable file, to visualize a scalar value, or **-log** *scalarFileName*, the keyword **-log** followed by the fully qualified path name of a scalar variable file, to visualize the decadic logarithm of a scalar value, or *vectorFileName*, the fully qualified path name of a vector variable file, to visualize a vector value.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the StructuredGridASCII module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile*
`-class StructuredGridASCII` *ModelPrefixOrBaseDirectory timestep ScalarVariableType1*
*ScalarVariableTypeN VectorVariableType*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -class StructuredGridASCII model1 0 temp velocity`

This will load a grid file with temperature, log(viscosity), and vector velocity stored at each vertex, assuming those scalar and vector files actually contain what they advertise. Variable names are not taken from the scalar or vector file names (they are arbitrary), but from the scalar and variable vector file headers.

**Recommended Directory Layout** A data set in the format described here consists of several files: one defining the grid, and one defining each scalar or vector variable. These files do not have to be stored in the same directory; each one is given to the StructuredGridASCII module by its fully qualified path name. However, to simplify data management and loading data sets into 3D Visualizer, we recommend the following directory structure: All files belonging to the same (time-varying) data set are stored in the same base directory, named to identify the data set. Directly inside that directory is the file defining the grid structure common to all time steps, named "grid." Each time step has a subdirectory inside the base directory, named "time$X$", and each time step directory in turn contains all variable files defining that time step, named by the (abbreviated) name of the variable they contain, such as "temp," "visc," or "velo." Any metadata or annotations pertaining to the entire data set can be stored in the base directory, for example in a file named "metadata" (which is ignored by 3D Visualizer). Metadata or annotations pertaining only to individual time steps could be stored in "metadata" files inside each time step directory. This recommended layout keeps all files belonging to the same data set together, and allows copying or moving an entire data set from one computer to another by simply copying the base directory. Similarly, data sets can be packaged for archival storage or transmission by storing the base directory in a ZIP or tar archive.

### A.1.2 SphericalASCIIFile

In many cases, data sets to be visualized are stored in single ASCII files of varying formats, which only have in common that the attributes of a single grid vertex, i. e., vertex positions and data values, are written as columns in a single line. These files typically differ in the number of columns, ordering of columns, vertex counting order, header format, and amount of information about the file stored in the file itself. SphericalASCIIFile is a "fallback" module that can read many such files, but relies on the user to provide all information about the file layout on the module's command line.

**Module Command Line** This module recognizes a large number of command line options describing all aspects of a particular file format that need to be known to successfully read the file. These command line options can be supplied to the module in any order. The complete command line is defined as:

**-class SphericalASCIIFile** *dataFileName option1 ... optionN*

where *dataFileName* is the fully qualified path name of the file to read, and *option1* to *optionN* are the options defining the file format and data layout.

The full set of available options are:

**-headers *numHeaderLines*** Specifies the number of header lines at the beginning of the file. Since the module ignores any header information in the file, this is the number of lines that need to be skipped when reading the file to reach the actual vertex data. If not specified, this number defaults to 0.

**-nodes *NI NJ NK*** Defines the number of nodes in $i, j, k$. Since this module only reads data in spherical coordinates, $i$ always corresponds to radius, $j$ corresponds to (co)latitude, and $k$ corresponds to longitude.

**-nodecount *slow medium fast*** Defines the node counting order used in the file, i. e., which of the components $i, j, k$ varies fastest or most slowly. *slow*, *medium*, and *fast* correspond to the components $i, j, k$, where $i$ is given as 0, $j$ as 1, and $k$ as 2. The component given first varies most slowly when reading the file top to bottom, the component given last varies fastest. For example, **-nodecount 0 1 2** means that $i$ (radius, 0) varies most slowly, and $k$ (longitude, 2) varies fastest – the file scans through longitude, then (co)latitude, then radius. To give another example, **-nodecount 1 2 0** means that (co)latitude varies most slowly, and radius varies fastest. If not specified, this setting defaults to 0 1 2, i. e., radius varying most slowly and longitude varying fastest.

**-coords *longitudeCol (co-)latitudeCol radiusCol*** Defines which columns in each row define a vertex' longitude, (co)latitude, and radius, respectively. Columns are counted starting at 0 for the first column. For example, if the second column in each row defines latitude, the third column longitude, and the fourth column radius, this would be specified as **-coords 2 1 3** (column counting is zero-based).

**-colat** If this flag is present, the module interprets latitudes stored in the file as colatitudes, increasing from 0 at the north pole to $\pi$ at the south pole. If not specified, the latitude mode defaults to latitude with the south pole at $-\pi/2$, the equator at 0, and the north pole at $\pi/2$.

**-degree** If this flag is present, the module interprets all angle components (longitude and (co)latitude) as degrees instead of radians. If not specified, all angles are in radians.

**-radius *radiusScale*** Defines a scaling factor for all radius components. The module expects radii to be given in meters; if, for example, radii in the file are given in kilometers, this would be specified as **-radius 1000**. If not specified, this value defaults to $6378.14 \cdot 10^3$, so that non-dimensional radii varying from 0 at the Earth's center to 1 at the surface are properly scaled to meters (3D Visualizer does not use a flattened geoid).

**-scalar *scalarName valueCol*** This option defines a scalar variable stored in the file. *scalarName* is an arbitrary name used to select the variable for visualization inside 3D Visualizer, and *valueCol* is the (zero-based) index of the column containing the variable's value for each vertex.

**-vector *vectorName radiusCol latitudeCol longitudeCol*** This option defines a vector variable stored in the file. *vectorName* is an arbitrary name used to select the variable for visualization inside 3D Visualizer, and *radiusCol*, *latitudeCol*, and *longitudeCol* are the (zero-based) indices of the columns containing the variable's

components for each vertex. All three vector components are defined *in length units per time unit* along the local longitude, (co)latitude, and radius directions. This is done to ensure compatibility with existing simulation codes storing vector values; it means that the length of a vector does not scale with the radius of its associated vertex.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the SphericalASCIIFile module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile* `-class SphericalASCIIFile` *ModelName* `-headers` *nlineHeader* `-nodes` *nlon nlat nrad* `nodecount` *lonvary latvary radvary* `-coords` *loncol latcol radcol* `-degree -radius` *1000.0* `-scalar` *VariableName VariableColumn* `-scalar` *VariableName VariableColumn*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -palette palette2.pal -class SphericalASCIIFile tomog.dat -headers 1 -nodes 81 81 41 nodecount 2 1 0 -coords 2 1 0 -degree -radius 1000.0 -scalar Vp 3 -scalar Vs 4`

where the file name is tomog.dat, the file starts with one header line (which is subsequently ignored). The file has 81 nodes in longitude, 81 nodes in latitude, and 41 nodes in radius. The node counting order is longitude varies fastest (2), radius varies most slowly (0). The nodes' longitudes, latitudes, and radii data are stored in columns 2, 1, and 0, respectively (columns are counted starting with zero). Longitude and latitude are given in degrees. radii are given in km (the value is a conversion factor to m, defaults to the Earth's radius in m). There is a scalar variable called "Vp" in column 3, and there is a scalar variable called "Vs" in column 4.

### A.1.3 ImageStack

The ImageStack module reads 3D volume data as a stack of slice images, stored in PPM, PNG, or JPEG formats. Stacks of slice images are a common data exchange format for 3D volume data generated from serial sectioning, confocal microscopy, X-ray or neutron Computed Tomography (CT), or Magnetic Resonance Imaging (MRI). Each slice image defines a single layer of *voxels*, 3D boxes of a fixed width, height, and thickness, corresponding to the pixels of the input images. The voxels defined by each slice are layered on top of each other, resulting in a regular (Cartesian) 3D grid. The layout of an image stack (number of images, pixel size, slice distance, etc.) is defined in a special *stack header file*, whose format is defined below.

**Module Command Line** The ImageStack module takes the name of the stack header file defining an image stack's layout, and a set of optional filtering parameters, from its command line, defined as follows:

**-class ImageStack** *stackHeaderFileName* [**medianfilter**] [**lowpassfilter**]

(the square brackets denote optional parameters), where *stackHeaderFileName* is the name of the stack header file to read (defined below), and **medianfilter** and **lowpassfilter** are the optional filtering parameters. If **medianfilter** is present, each voxel in the image stack is replaced by the median value of the voxel and its two immediate neighbors in the slice below it and above it. This filtering parameter can reduce noise created by different lighting conditions when capturing each slice image. If **lowpassfilter** is present, each voxel (after optional median filtering was applied) is replaced by a weighted average of its immediate neighbors. This filtering parameter can reduce random noise in the source images, at the cost of resolution.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the ImageStack module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -class ImageStack` *stackHeaderFileName* [`medianfilter`] [`lowpassfilter`]

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -class ImageStack microfossilA medianfilter`

**Stack Header File Format** A stack header file is an ASCII file defining the layout of a stack of slice images. It is a sequence of settings written to individual lines. The available settings are:

**numSlices = *numSlices*** Specifies the number of slice images in the image stack.

**imageSize = *width height*** Specifies the width and height of the pixel rectangle to be used from each slice image in the image stack, in pixels.

**regionOrigin = *originX originY*** Specifies the lower-left corner of the pixel rectangle to be used from each slice image in the image stack, in pixels. The lower-left most pixel in an image has coordinates $(0, 0)$. If this setting is not specified, it defaults to 0 0.

**sampleSpacing = *voxelThickness voxelWidth voxelHeight*** Defines the size of a voxel in arbitrary units. *voxelThickness* is the distance between two slice images in the image stack, and *voxelWidth* and *voxelHeight* are the width and height of a pixel in each slice image, respectively.

**sliceDirectory = *directoryName*** Defines the name of a directory containing all slice image files.

**sliceFileNameTemplate = *fileNameTemplate*** Defines a template to construct the names of slice image files based on a slice number. *fileNameTemplate* is a printf-style format string (see "man 3 printf" for details) defining how to format the slice number of a slice and insert it into the template name. The format string must contain exactly one **%d** conversion flag at the point where the slice number should be inserted. The number of digits to use for the slice number, and whether to pad slice numbers with leading zeros, can be specified by modifying the conversion flag: **%0***numDigits***d** converts numbers using always *numDigits* digits, padded with leading zeros. For example, the format string **sliceImage%04d.png** will define slice image file names **sliceImage0000.png**, **sliceImage0001.png**, etc.

**sliceIndexStart = *sliceIndexStart*** Defines the slice number to use for the first slice image in the image stack. If this setting is not specified, it defaults to 0.

**sliceIndexFactor = *sliceIndexFactor*** Defines the increment in slice numbers between adjacent slice images in the image stack. If this setting is not specified, it defaults to 1.

**Slice Number Generation** The slice image file names generated for each slice image are defined by the **numSlices**, **sliceFileNameTemplate**, **sliceIndexStart**, and **sliceIndexFactor** settings. For an image stack containing *numSlices* slice images, (internal) slice indices always range from 0 to $numSlices - 1$. Internal slice indices are converted to slice numbers, used to generate slice image file names, using the specified *sliceIndexStart* and *sliceIndexFactor* values. The exact formula is $sliceNumber = sliceIndex \cdot sliceIndexFactor + sliceIndexStart$. For example, with **numSlices = 10**, **sliceIndexStart = 5**, and **sliceIndexFactor = 2**, the generated slice numbers are 5, 7, 9, ..., 23. The generated slice numbers are then inserted into the slice file name template as described above, and the ImageStack module looks for slice image files of the generated names in the specified slice directory.

**Pixel Rectangles** The ImageStack module can extract subregions from slice images, either to cut away unneeded boundary regions on-the-fly, or to account for slice image files with different sizes. When assembling an image stack, each slice images are aligned according to their lower-left corners, which have pixel coordinates $(0, 0)$. The **regionOrigin** and **imageSize** settings define the lower-left corner, width, and height, respectively, of a subregion to be extracted from each image slice. To succeed, each slice image must completely contain the specified rectangle.

## A.2 Community Software Data Formats Supported by This Release of 3D Visualizer

To make 3D Visualizer immediately useful to the community, it contains seven other pre-made modules corresponding to data storage formats used by several commonly-used simulation codes.

### A.2.1 StructuredGridVTK – Gale 1.2.1

This module reads *structured grids*, i. e., grids with explicit Cartesian coordinates for each vertex, from ASCII files in legacy VTK file format up to and including version 3.0 (see http://www.vtk.org/pdf/file-formats.pdf). This is also the native data storage format of the CIG-supported Gale-1.2.1 particle-in-cell simulation code, meaning that 3D Visualizer can read and visualize Gale simulation results without any work required from the user. This release of 3D Visualizer does not yet read particle tracer output from Gale simulations. For more information on Gale-1.2.1, visit the CIG web site at:
http://www.geodynamics.org/cig/software/packages/long/gale .

**Module Command Line** The StructuredGridVTK module takes the name of the VTK structured grid file to read from its command line, which is defined as follows:

**-class StructuredGridVTK** *vtkFileName*

where *vtkFileName* is the fully qualified path name of the file to read. The module will automatically read all scalar and/or vector variables defined in the VTK file, and does not recognize any other command line options.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the StructuredGridVTK module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -class StructuredGridVTK` *vtkFileName*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -class StructuredGridVTK` *Model3.dat*

### A.2.2 CitcomSGlobalASCIIFile – CitcomS-3.0.1 Global ASCII Output

This module reads ASCII output files written by whole-sphere CitcomS-3.0.1 simulations. It also reads output files written by regional CitcomS-3.0.1 simulations, but we recommend using the more efficient CitcomSRegionalASCIIFile module for those. The module reads the separate files written by each CPU in a parallel simulation run and combines them into a single multi-block grid on-the-fly; there is no need for users to combine per-CPU files in a pre-processing step. This release of 3D Visualizer does not yet read particle tracer output from CitcomS simulations. For more information on CitcomS-3.0.1, visit the CIG website at:
http://www.geodynamics.org/cig/software/packages/mc/citcoms .

**Module Command Line** The CitcomSGlobalASCIIFile module takes the name of the configuration file describing the CitcomS run, and the names of all variable files to be read, from its command line, which is defined as follows:

**-class CitcomSGlobalASCIIFile** *pidFileName timeStep variable1 ... variableN*

where *pidFileName* is the fully qualified path name of the simulation run's configuration file (typically **pid***Number***.cfg**), *timeStep* is the number of the simulation time step to load, and each *variableI* is either *scalarName* to visualize a scalar value, or **-log** *scalarName* to visualize the decadic logarithm of a scalar value, or **-vector** *vectorName* to visualize a vector value.

The **velo** velocity vector variable (specified on the command line as **-vector velo**) is a special case. For some reason, velocity files additionally contain temperature values. If the **velo** variable name is listed on the module's command line, the module will automatically read the temperature scalar value in addition to the velocity vector value. If the **velo** variable name is preceded by the keyword **-log** (either written **-log -vector velo** or **-vector -log velo**), the module will visualize the decadic logarithm of temperature instead of the given values.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the CitcomSGlobalASCIIFile module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile*
`-class CitcomSGlobalASCIIFile` *pidFile timestep ScalarVariableType1 ScalarVariableTypeN VectorVariableType*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -palette PaletteGreen.pal`
`-class CitcomSGlobalASCIIFile pid30171.cfg 100 temp -log visc -vector velo`

**Directory Layout** According to CitcomS' directory layout, the actual file names read by this module are *dataSetName*.**coord.***cpuNumber*, the file defining the grid coordinates, and *dataSetName*.*variableName*.*cpuNumber*.*timeStep*, the files defining the data values for a selected time step. *dataSetName* is the common prefix of all files belonging to a data set, and is defined in a simulation run's configuration file.

### A.2.3 CitcomSRegionalASCIIFile – CitcomS-3.0.1 Regional ASCII Output

This module reads ASCII output files written by regional (single cap) CitcomS-3.0.1 simulations. It is a special-case optimization of the more general CitcomSGlobalASCIIFile module, and has the same module command line. It only reads data sets containing a single spherical cap, and will print an error message and exit if presented with a multi-cap or whole-sphere CitcomS data set.

**Module Command Line** The CitcomSRegionalASCIIFile module's command line is defined as follows:

**-class CitcomSRegionalASCIIFile** *pidFileName timeStep variable1 ... variableN*

where all command line options are exactly the same as for the CitcomSGlobalASCIIFile module (see Section A.2.2).

**Running 3D Visualizer on the Command Line** The general form of the command-line for the CitcomSRegionalASCIIFile module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile*
`-class CitcomSRegionalASCIIFile` *pidFile timestep ScalarVariableType1 ScalarVariableTypeN*
*VectorVariableType*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -palette PaletteGreen.pal`
`-class CitcomSRegionalASCIIFile pid51271.cfg 100 temp -log visc -vector velo`

### A.2.4 CitcomCUSphericalRawFile – CitcomCU-1.0.2 Spherical Binary Output

This module reads binary output files written by CitcomCU-1.0.2 simulations performed in spherical coordinates. The module assumes that CitcomCU-1.0.2 was compiled using the special output filter Output_ucd.c rather than the standard output filter Output.c. Output_ucd.c, written by Magali I. Billen at the University of California, Davis, writes binary files that contain scalar, vector, and tensor model output. Binary files are more compact, more precise, and much more efficient to read than ASCII files. The module reads the separate files written by each CPU in a parallel simulation run and combines them into a single multi-block grid on-the-fly; there is no need for users to combine per-CPU files in a pre-processing step. The current version of the CitcomCUSphericalRawFile module assumes that data was written by a computer with little endian byte ordering, e. g., a computer using an AMD or Intel CPU. It can currently not read data written by big-endian computers, such as PowerPC Mac computers; however, it can read data written by little-endian computers even when 3D Visualizer is run on a big-endian computer. For more information on CitcomCU-1.0.2, visit the CIG website at:
http://www.geodynamics.org/cig/software/packages/mc/citcomcu .
To read a detailed description of Output_ucd.c, a detailed list of the differences between Output_ucd.c and the standard CitcomCU Output.c, and to download Output_ucd.c, visit the UC Davis KeckCAVES web site at:

http://www.keckcaves.org/software/VISUALIZERCG/index.html
and download the following files: citcomcu_output_info.pdf, Output_ucd.c, and convertcu_ucd.c.

**Module Command Line** The CitcomCUSphericalRawFile module takes the name of a header file describing the layout of a data set, the number of the time step to load, and the names of scalar or vector variables to visualize from its command line, which is defined as follows:

**-class CitcomCUSphericalRawFile** *headerFileName timeStep variable1 ... variableN*

where *headerFileName* is the fully qualified path name of the header file describing the data set's layout, *timeStep* is the number of the simulation time step to load, and each *variableI* is either *scalarName* to visualize a scalar value, or **-log** *scalarName* to visualize the decadic logarithm of a scalar value, or **-vector** *vectorName* to visualize a vector value.

**Running 3D Visualizer on the Command Line** The general form of the command-line for the CitcomCU-SphericalRawFile module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile* `-class CitcomCUSphericalRawFile` *ModelPrefix timestep ScalarVariableType1 ScalarVariableTypeN VectorVariableType*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -class CitcomCUSphericalRawFile slab3dB 0 -vector velo temp -log visc`

where the command will plot the 0*th* time-step results of a multiprocessor CitcomCU run where the data output files all have the prefix "slab3dB".

**Directory Layout** According to the directory layout encoded in Output_ucd.c, the actual file names read by this module are *dataSetName***.hdr**, the header file defining a data set's layout, *dataSetName***.x.***cpuNumber*, *dataSetName***.y.***cpuNumber*, and *dataSetName***.z.***cpuNumber*, the files defining the grid coordinates, and *dataSetName***.***variableName***.***cpuNumber***.***timeStep*, the files defining the data values for a selected time step. *dataSetName* is the common prefix of all files belonging to a data set.

### A.2.5 CitcomCUCartesianRawFile – CitcomCU-1.0.2 Cartesian Binary Output

This module reads binary output files written by CitcomCU-1.0.2 simulations performed in Cartesian coordinates, assuming that CitcomCU-1.0.2 was compiled with the Output_ucd.c output filter (see Section A.2.4).

**Module Command Line** The CitcomCUCartesianRawFile module's command line is defined as follows:

**-class CitcomCUCartesianRawFile** *headerFileName timeStep variable1 ... variableN*

where all command line options are exactly the same as for the CitcomCUSphericalRawFile module (see Section A.2.4).

**Running 3D Visualizer on the Command Line** The general form of the command-line for the CitcomCU-CartesianRawFile module is:
% */Path/to/3DVisualizer/executable/*`3DVisualizer -palette` *SavedColorPaletteFile*
`-class CitcomCUCartesianRawFile` *Path/to/data/followedby/ModelPrefix timestep ScalarVariableType1 ScalarVariableTypeN VectorVariableType*

For example,
% `$HOME/3DVisualizer-1.0/bin/3DVisualizer -class CitcomCUCartesianRawFile box3d 0 -vector velo temp -log visc`

where the command will plot the $0th$ time-step results of a multiprocessor CitcomCU run where the data output files all have the prefix "box3d".

## A.3 Creating New Modules to Read Additional Data Formats

Creating new modules to read previously unsupported data formats requires a familiarity with C++, but is relatively straightforward. The 3D Visualizer release contains the sources for all existing modules, and those sources can serve as guidelines or templates on how to create new modules. Due to 3D Visualizer's design, writing a new module does not require writing any visualization algorithms, such as isosurface or streamline extraction or volume rendering. All that code is automatically generated by the C++ compiler when a module is compiled. Writing a module only requires specifying *which* visualization algorithms to use, and how to read data sets in the data format embodied by the new module.

Modules are implemented as subclasses of the "Module" base class, and typically written to a header (.h) and implementation file (.cpp) of the new module's name. These C++ files are stored in the "Concrete" directory in 3D Visualizer's source directory, e. g., $HOME/src/3DVisualizer-1.0/Concrete. After the source code for a new module is finished, it can be integrated into 3D Visualizer by adding its name to the MODULE_NAMES list at the beginning of 3D Visualizer's makefile, and re-building 3D Visualizer by typing "make" (or "make install") from inside 3D Visualizer's source directory (see Section 3 for details).

As a concrete example, here is the *complete* source code implementing a simple module that reads single-valued data in Cartesian coordinates from text files with a single header line defining the number of vertices in the grid, followed by one line per vertex, with four columns per line defining a vertex' $x$, $y$, and $z$ coordinates and its temperature value, respectively (this example source code is also included, with complete comments, with the 3D Visualizer software package, as "ExampleModule" in the "Concrete" subdirectory under 3D Visualizer's source directory):

```
#include <stdio.h>
#include <Plugins/FactoryManager.h>
#include <Wrappers/SlicedCurvilinearIncludes.h>
#include <Wrappers/SlicedScalarVectorDataValue.h>
#include <Wrappers/Module.h>

namespace Visualization {
namespace Concrete {
namespace {
typedef float Scalar;
typedef float VScalar;
typedef Visualization::Templatized::SlicedCurvilinear<Scalar,3,VScalar> DS;
typedef Visualization::Wrappers::SlicedScalarVectorDataValue<DS,VScalar> DataValue;
typedef Visualization::Wrappers::Module<DS,DataValue> BaseModule;
}

class ExampleModule:public BaseModule
   {
   public:
   ExampleModule(void)
     :BaseModule("ExampleModule")
     {
     }
   virtual Visualization::Abstract::DataSet* load(const
   std::vector<std::string>& args) const
```

```
      {
      DataSet* result=new DataSet;
      FILE* file=fopen(args[0].c_str(),"rt");
      DS::Index numVertices;
      fscanf(file,"%d %d %d",&numVertices[0],&numVertices[1],&numVertices[2]);
      DS& dataSet=result->getDs();
      dataSet.setGrid(numVertices);
      dataSet.addSlice();
      DataValue& dataValue=result->getDataValue();
      dataValue.initialize(&dataSet);
      dataValue.setScalarVariableName(0,"Temperature");
      for(DS::Index index(0);index[0]<numVertices[0];index.preInc(numVertices))
        {
        double pos[3],temp;
        fscanf(file,"%lf %lf %lf %lf",&pos[0],&pos[1],&pos[2],&temp);
        dataSet.getVertexPosition(index)=DS::Point(pos);
        dataSet.getVertexValue(0,index)=DS::ValueScalar(temp);
        }
      fclose(file);
      dataSet.finalizeGrid();
      return result;
      }
    };
}
}

extern "C" Visualization::Abstract::Module* createFactory(
     Plugins::FactoryManager<Visualization::Abstract::Module>& manager)
    {
    return new Visualization::Concrete::ExampleModule();
    }
extern "C" void destroyFactory(Visualization::Abstract::Module* module)
    {
    delete module;
    }
```

# B Appendix: Example of How to Modify the Configuration File

This appendix provides an example of how to modify the pre-defined mouse-key combinations in the Vrui.cfg file. The file Vrui.cfg is the configuration file that tells Vrui, and therefore 3D Visualizer, the user preferences and options for how 3D Visualizer looks and how interaction is achieved, in any user environment. The main Vrui.cfg file is kept in `/Vrui-1.0/etc` subdirectory.

## B.1 Default Button-Key Configuration

The mouse buttons and any key on the keyboard can be used as a button to which a Tool can be assigned. In the default configuration, the available buttons are: left/middle/right mouse button, left shift key, Z key, the Q, W, A, S, and D keys, the number keys on the main keyboard, the Tab key, the number keys on the numeric key pad, and the Enter key on the numeric keypad. The available buttons are defined in the Vrui.cfg file, under the MouseAdapter section:

```
\section MouseAdapter
       inputDeviceAdapterType Mouse
       numButtons 3
       buttonKeys (LeftShift, z, q, w, a, s, d, \
              1, 2, 3, 4, 5, 6, 7, 8, 9, 0, \
              Tab, Num0, Num1, Num2, Num3, Num4, \
              Num5, Num6, Num7, Num8, Num9, NumEnter)
        modifierKeys (LeftCtrl, LeftAlt)
endsection
```

In the default configuration the MouseAdapter section also defines the Left-Control and Left-Alt keys as *modifier* keys. Using the modifier keys together with any other available button creates a new *plane* of buttons, each of which can also be assigned to a tool. For example, pressing the Left-Control *and* Tab is a different button than just pressing the Tab key. Modifying the list of buttonKeys above will change the buttons available for Tool assignment in 3D Visualizer.

## B.2 Button Numbering

Each of the mouse-buttons and buttonKeys defined in the MouseAdapter section is assigned a number, which is used in defining the default buttons used for navigation (i.e., panning, rotating, zooming, dollying). The numbering starts with the mouse buttons and then continues in order through the buttonKeys. For a three-button mouse the numbering order is:

0: Left button

1: Middle button and/or scroll button

2: Right button

while a two button mouse is numbered as:

0: Left button

1: Right button

In the MouseAdapter section, the variable numButtons is set to 3, which indicates the *maximum* number of buttons on the mouse.

Additional *buttonKeys* are numbered in the order they appear in the MouseAdapter section starting with either 3 (for a three button mouse) or 2 (for a two button mouse). The table below shows the numbering for a two button mouse:

| Key | LeftShift | z | q | w | a | s | d | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 |
| Key | 5 | 6 | 7 | 8 | 9 | 0 | Tab | Num0 | Num1 | Num2 | Num3 |
| Number | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| Key | Num4 | Num5 | Num6 | Num7 | Num8 | Num9 | NumEnter | | | | |
| Number | 26 | 27 | 28 | 29 | 30 | 31 | 32 | | | | |

## B.3 Default Navigation Configuration

The navigation tool and the menu tool (for the main menu) are the only tools that have a default setting for the binding of the tool to particular buttons on the mouse and keyboard. All other tools are bound to buttons by the user when the tool is needed. The section in the Vrui.cfg file called MouseNavTool assigns the default buttons for navigation:

```
section MouseNavTool
       toolClass MouseNavigationTool
       deviceName0 Mouse
       device0ButtonIndex0 0
       # Set device0ButtonIndex1 to 1 instead of 4 to use the
       # middle mouse button for panning and zooming/dollying
       # instead of the "z" key
      device0ButtonIndex1 4
      device0ButtonIndex2 3
      device0ValuatorIndex0 0
endsection
```

Each navigation tools correspond to the a particular device0ButtonIndex:

Rotating: deviceButtonIndex0

Panning: deviceButtonIndex1

Zooming: deviceButtonIndex0 & deviceButtonIndex1

Dollying: deviceButtonIndex0 & deviceButtonIndex1 & deviceButtonIndex2

Note that button-key combination for zooming and dollying are predefined by the choice of buttons selected for rotating and panning: zooming is the combination of these two buttons, while dollying also requires a third button.

The default configuration is for a two button mouse:

Rotating: Left Button

Panning: z key

Zooming: Left Button & z key

Dollying: Left Button & z key & LeftShift key

For a three button mouse, deviceButtonIndex1 is set to 1 instead of 4 as noted in the comment in the MouseNavTool section of Vrui.cfg, and the z key is replaced by the Middle mouse button:

Rotating: Left Button

Panning: Middle Button

Zooming: Left Button & Middle Button

Dollying: Left Button & Middle Button & LeftShift key

## B.4 Default MenuTool Configuration

The default menu tool is assigned to the Right mouse button in the MenuTool section of the Vrui.cfg file:

```
section MenuTool
      toolClass RayScreenMenuTool
      deviceName0 Mouse
      device0ButtonIndex0 2
endsection
```

From the numbering defined by the MouseAdapter section of Vrui.cfg, button number 2 is the right mouse button (for either a two or three button mouse), therefore deviceButtonIndex0 in the MenuTool section, which is set to 2, has been assigned to the right mouse button.

# C Appendix: Trouble-shooting

This section is not meant to be a complete list of trouble-shooting issues, but instead includes some of the issues we have encountered in installing the demonstration version.

- On MAC OS X systems, even if you launch Visualizer using a terminal window, you need to start X11 first.
- On MAC OS X systems it is recommended to have to the most up-to-date version of Xcode installed. We have faced problems with older versions of the gcc compiler included in earlier version of Xcode.
- If you have upgraded the source kernel for your unix or linux workstation, the driver for your graphics card may have been deleted, and the driver needs to be re-installed for the openGL to work with Visualizer. Drivers for Nvidia graphics cards can be downloaded from http://www.nvidia.com. Note X11 needs to be turned off while installing the driver (consult your linux manual).
- If you have a programmable mouse you may need to disable some of the special settings. For example, on one of our MAC OS X installations, a Logitech 7-button mouse was used. The three main mouse buttons were set to be standard clicks instead of any other special action. To set the mouse buttons we used the Logitech Control Center software that came with the mouse (from System Preferences, this will be listed in the Other category). The left and right mouse button were set to "click", the middle wheel button was set to "advanced click" and all other buttons were set to "nothing".

# D Appendix: CitcomS-3.0.1 Cookbook 1 Visualization

We ran the cookbook 1 example global model from the CitcomS-3.0.1 User Manual. The output from each processor, which were written in the standard ASCII file format, were loaded directly into 3D Visualizer by executing the following command on the command line:

```
% 3DVisualizer -class CitcomSGlobalASCIIFile pid30171.cfg 100 -log visc -vector velo
```

Note, the pid30171.cfg file is generated by CitcomS and is named by the *process identification number (pid)* of the run (i.e., your file will have a different number). Temperature is stored in the velocity file, so it is necessary to read in velocity if you want to plot temperature (i.e., do not list temp as one of the variables to read in, this will produce an error). The images below were generated using the methods described in Section 8.

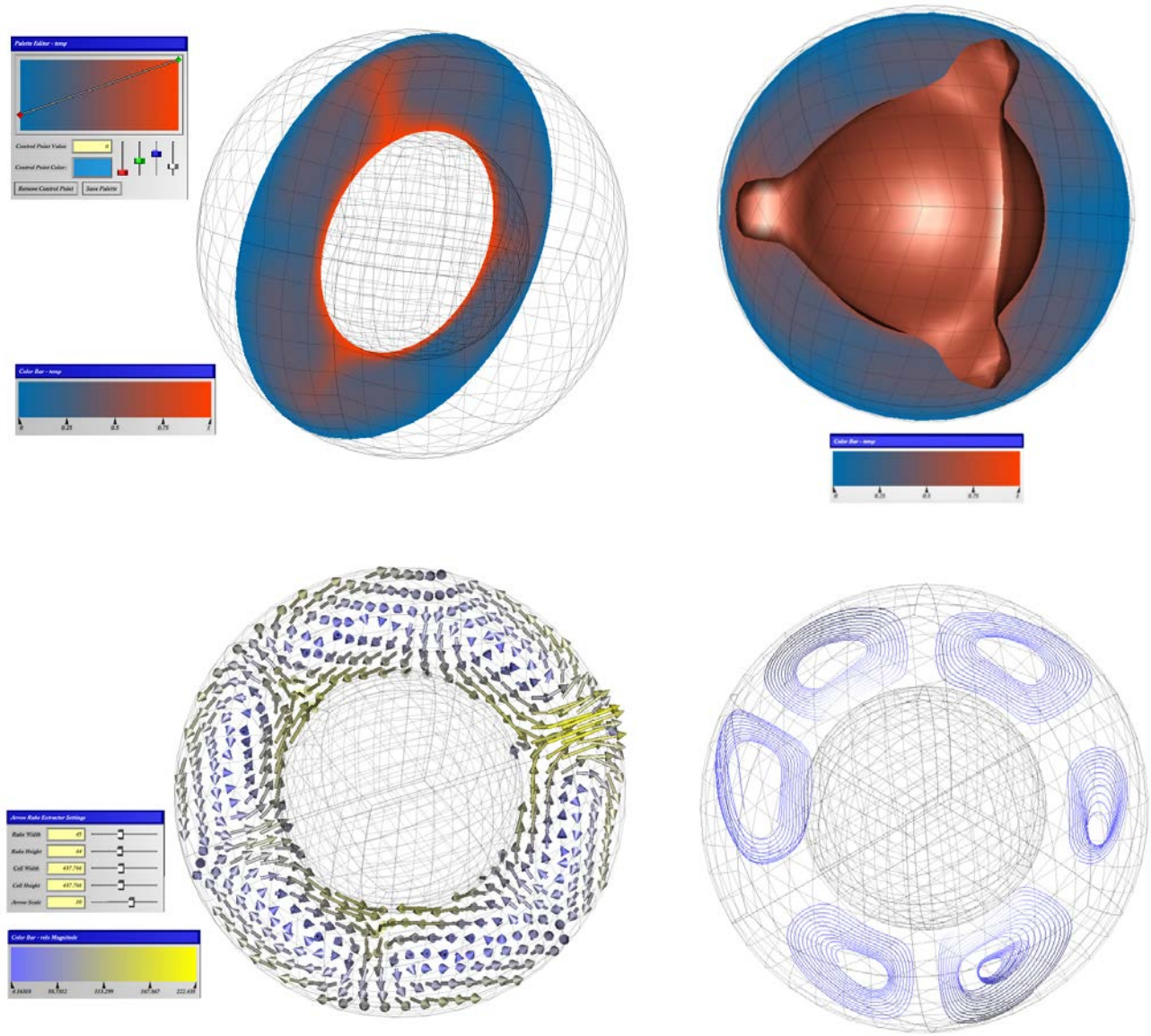


Figure 15: Visualization images for the CitcomS-3.0.1 Cookbook 1 model. *Upper Right*: Color slice of temperature shown with the spherical grid (grid faces), color bar and palette editor. *Upper Left* Color slice and isosurface of temperature. *Lower Right*: Visualization of flow field using the *Arrow Rake*, which creates arrows emanating from a grid of points on a plane of any orientation. *Lower Left*: Streamlines showing convection cells.

# E Appendix: CitcomS-3.0.1 Cookbook 3 Visualization

We ran the Cookbook 3 example regional model from the CitcomS-3.0.1 User Manual. The output from each processor, which were written in the standard ASCII file format, were loaded directly into 3D Visualizer by executing the following command on the command line:

% 3DVisualizer -class CitcomSRegionalASCIIFile pid5129.cfg 200 -log visc -vector velo

Note that the pid5129.cfg file is generated by CitcomS when the model is run and is named by the specific *process identification number (pid)* of the run. Therefore, this file will have a similar name, formatted as pidXXXX.cfg, but with a different number. Also, temperature is in the velocity file, so it is necessary to read in velocity if you want to plot temperature (i.e., do list temp as one of the variables to read in, this will produce an error). The images below were generated using the methods described in Section 8.

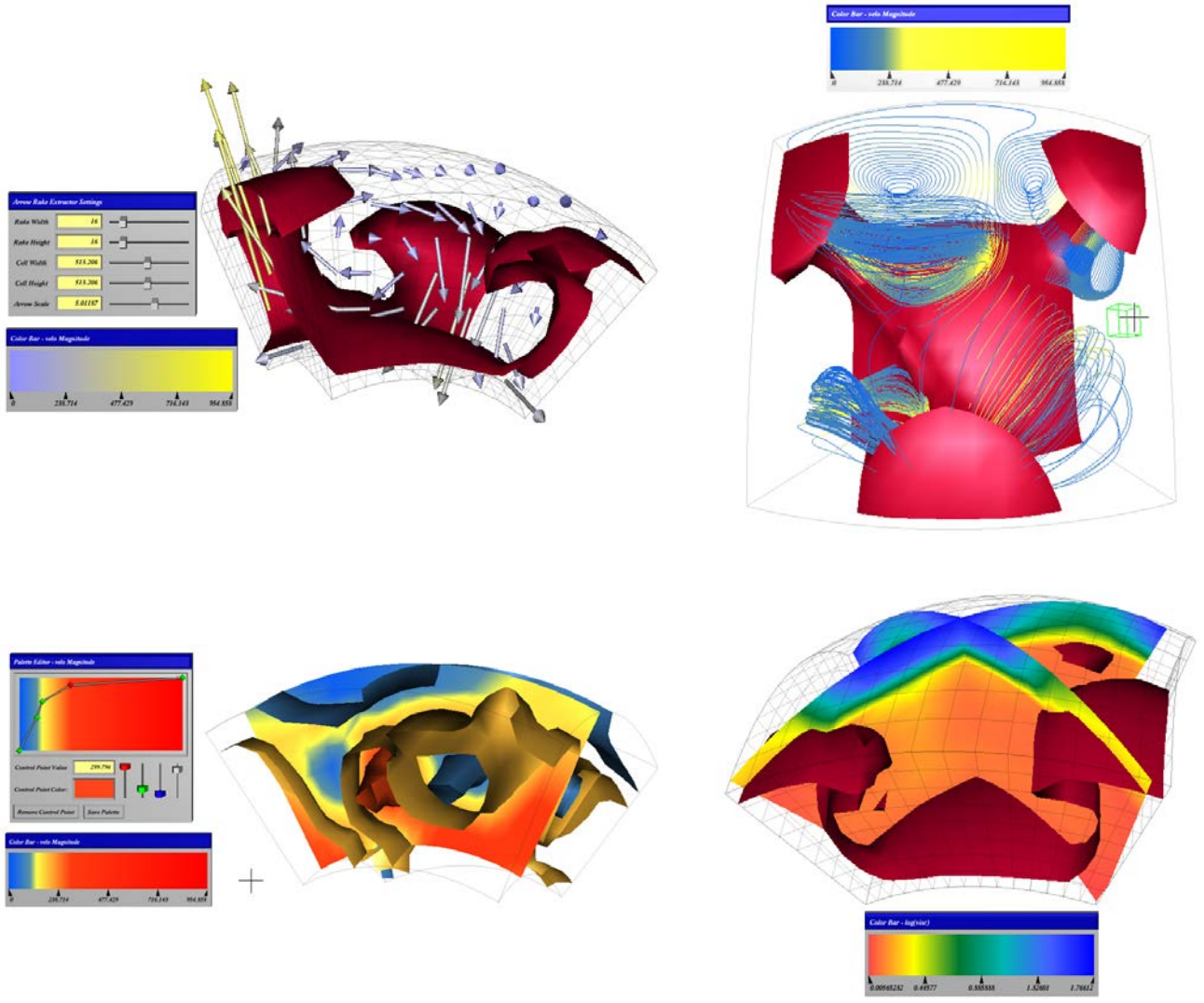


Figure 16: Visualization images for the CitcomS-3.0.1 Cookbook 3 model. *Upper Right*: Isosurface of temperature for hot upwelling material and an arrow rake showing the circulating velocity field. Note colorbar is for velocity magnitude and arrows are colored by velocity magnitude. *Upper Left*: Isosurface of temperature with several streamlines. *Lower Right*: Isosurfaces and color slice of velocity magnitude. *Lower Left*: Color slice of $\log_{10}$ viscosity and isosurface of temperature.

# F Appendix: CitcomCU-1.0.2 Input 1 Test Case Visualization

We ran the "input1" test case that comes with the CitcomCU-1.0.2 release. The binary output files from CitcomCU-1.0.2 (using the Output_ucd.c output file) for each processor were loaded directly into 3D Visualizer by executing the following command on the command line:

% 3DVisualizer -class CitcomCUSphericalRawFile caseA 200 temp -log visc -vector velo.

where caseA is the modelname used in the CitcomCU input file. The images below were generated using the methods described in Section 8.

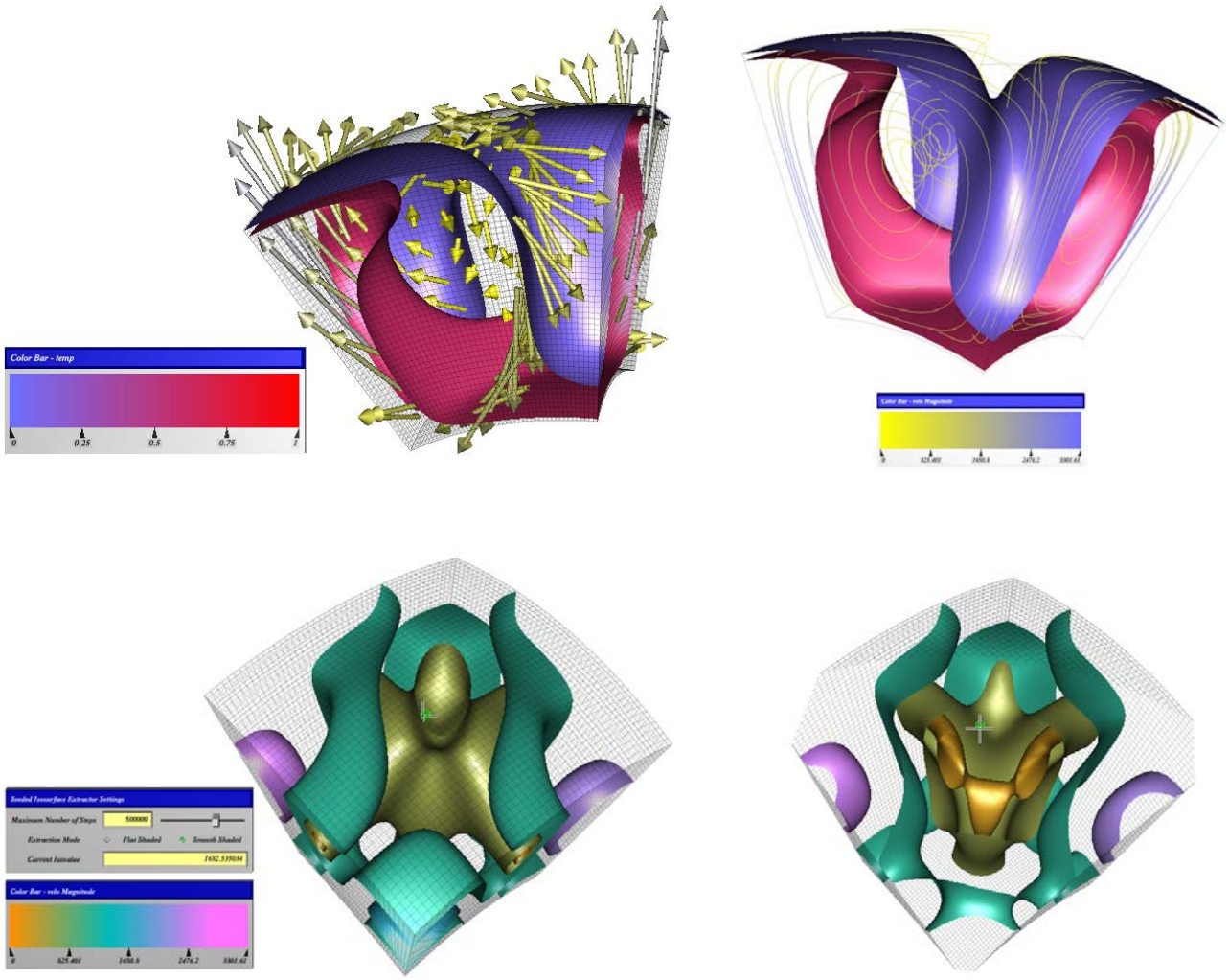



Figure 17: Visualization images for the CitcomCU-1.02 input 1 test case. *Upper right*: Isosurfaces of hot upwelling and cold downwelling material and two arrow rakes showing the circulating velocity field that feeds the upwellings and leaves the downwellings. Note colorbar is for temperature. *Upper Left*: Same isosurfaces as in part A, but shown with streamlines. Note colorbar is for velocity magnitude and streamlines are colored by velocity magnitude. *Lower Left*: Isosurface of velocity magnitude showing fast moving material in the corners, and a near-zero region in the center. *Lower Right*: Same velocity magnitude isosurfaces viewed with a cutting plane exposing the structure at the center of the box..