SCIRun User Guide

SCIRun 4.3 Documentation

Center for Integrative Biomedical Computing
Scientific Computing & Imaging Institute
University of Utah

SCIRun software download:

http://software.sci.utah.edu

Center for Integrative Biomedical Computing:

http://www.sci.utah.edu/cibc

Supported by:

NIH grant P41- RR12553-07

Author(s):
SCIRun Development Team

Contents

Chapter 1

SCIRun Introduction

SCIRun is a *modular dataflow programming* Problem Solving Environment (PSE). SCIRun has a set of Modules that perform specific functions on a data stream. Each module reads data from its input ports, calculates the data, and sends new data from output ports.

In SCIRun, a module is represented by a rectangular box on the Network Editor canvas. Data flowing between modules is represented by pipes connecting the modules. A group of connected modules is called a Dataflow Network, or net (see figure 1). An infinite number of nets can be created, each solving a separate problem. Chapter 1 of this tutorial demonstrates the use of SCIRun to visualize a tetrahedral mesh.

This chapter demonstrates the construction of a network comprised of three standard modules: ReadField, ShowField, and ViewScene. This chapter also instructs the user on reading Field data from a file, setting rendering properties for the nodes, edges, and faces (the nodes are rendered as blue spheres), and rendering geometry to the screen in an interactive ViewWindow.

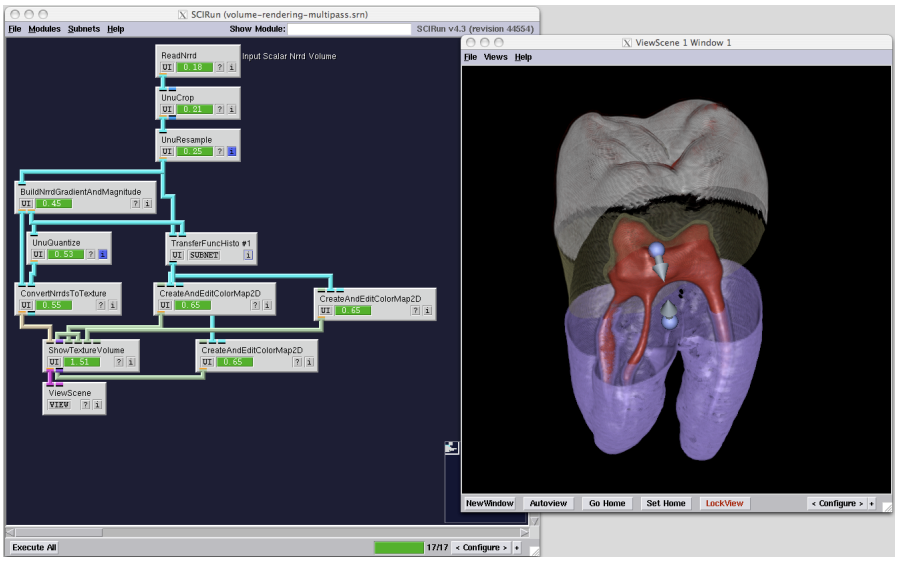

Figure 1.1. SCIRun Dataflow Network

1.1 Starting SCIRun

1.1.1 SCIRun usage

Use the –**help** flag for SCIRun usage and a full list of command line flags.

```
Usage: scirun [args] [net_file] [session_file]
    [-]-d[atadir]            : scirun data directory
    [-]-r[egression]         : regression test a network
    [-]-R[egressionimagedir] : output directory for regression tests
    [-]-s[erver] [PORT]      : start a TCL server on port number PORT
    [-]-e[xecute]            : executes the given network on startup
    [-]-E[xecute]            : executes the given network on startup and
                               quits when done
    [-]-c[onvert]            : converts a .net to a .srn network and exits
    [-]-v[ersion]            : prints out version information
    [-]-h[elp]               : prints usage information
    [-]-p[ort] [PORT]        : start remote services port on port number PORT
    [-]-l[ogfile] file       : add output messages to a logfile
    [-]-t[imeout] N          : kill scirun after N seconds
    [-]-I[mage] file         : Save resulting images in file
    [--nosplash]             : disable the splash screen
    net_file                 : SCIRun Network Input File
    session_file             : PowerApp Session File
```

1.1.2 Preparation for Unix, Mac, Linux, etc.

Preparations must be made before starting SCIRun and its PowerApps. Commands below are typed in a terminal emulation application (e.g. xterm). See SCIRun's initialization file *$HOME/.scirunrc* for additional information. Change directory to SCIRun's build directory and start SCIRun from the command line:

```
cd SCIRun/bin
./scirun [--execute] [network_file]
```

The **scirun** command may take the name of a SCIRun network file (network files have a **.srn** extension) for SCIRun to load. The –**execute** flag tells SCIRun to execute the network after it is loaded. Network files are discussed in a later section.
Do not start SCIRun in the background, i.e. do not type: scirun &.

Mac OS X 10.5 and greater

SCIRun can be started in either the OS X Terminal or the X11 terminal. If the binary package has been installed, the SCIRun application can be launched from Finder.

Mac OS X 10.4

Run SCIRun in the X11 Terminal. SCIRun can also be launched from Finder.

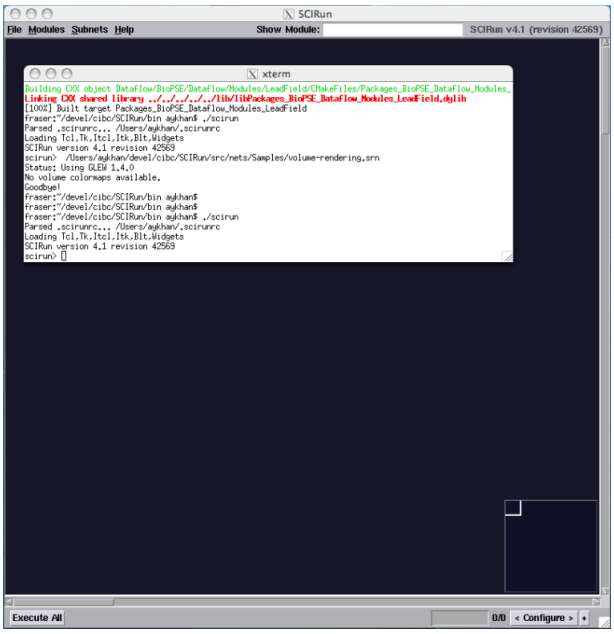

Figure 1.2. SCIRun on OS X 10.4

1.1.3 Preparation for Windows

In Windows, SCIRun can be stared from Explorer, the Start Menu, or launched from a "Windows Command Prompt" (cmd.exe). The installer can also create a desktop shortcut. Alternatively, you can start it up by browsing My Computer or Windows Explorer to the directory that contains scirun.exe and double-clicking on it. The SCIRun initialization file *.scirunrc* can be found in the user applicaton data folder. Command line flags are the same as for Unix platforms.

```
cd "c:\Program Files\SCIRun 4.3\bin"
scirun.exe [--execute] [network_file]
```

If you install the Windows binary version of SCIRun or associate .srn files with SCIRun, you can also double-click on any scirun network file (.srn) and it will run SCIRun with that network file. To associate srn files with SCIRun, do the following:

1. Open a Windows Explorer window, click the Tools menu and select Options
2. Click the File Types tab, and select New
3. In the Dialog that appears, type srn and click OK
4. In the Details for SRN extension frame, click Change and navigate to scirun.exe

1.1.4 Post startup

If the name of a network file is not provided, SCIRun starts with an empty window as shown below.

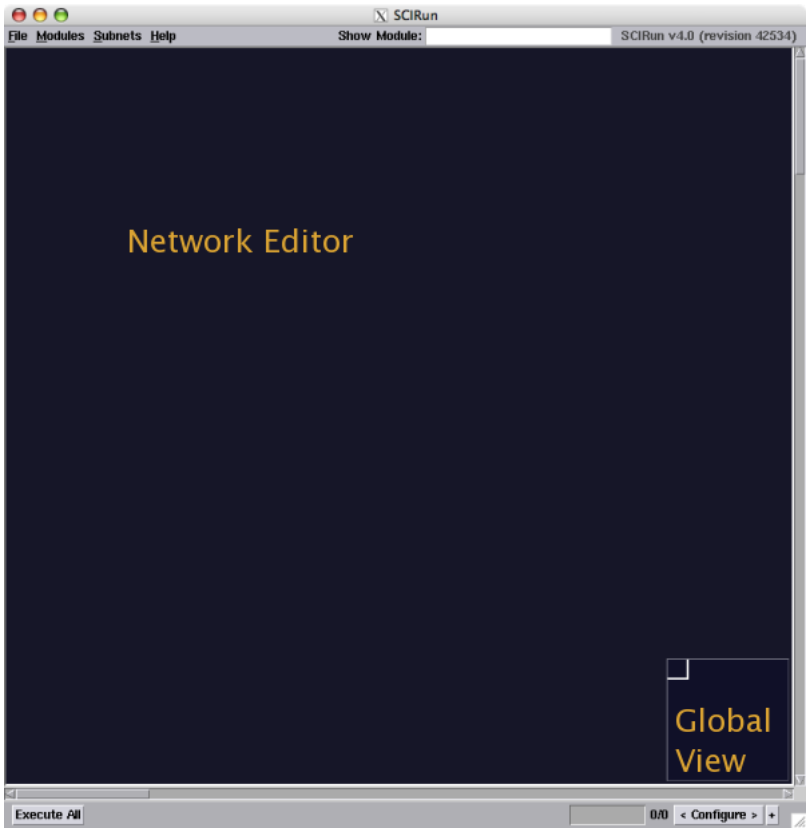

Figure 1.3. SCIRun initial NetworkEditor window

SCIRun may encounter errors during start up. Errors, warnings, and other messages are displayed in SCIRun's log (window?). Errors should be reported to the SCIRun development team through SCI Bugzilla.

Remote Display

Normally, SCIRun's main window is displayed on a console that is part of the computer on which SCIRun runs. It is possible, however, to display SCIRun on a remote console. In the discussion that follows, the term local refers to the machine running SCIRun and the term remote refers to the machine displaying SCIRun.

Advanced GL remote rendering is problematic.

SCIRun is compiled against the GL driver on the local machine, but when displayed remotely, uses the remote machine's GL driver. Many times the local and

remote drivers are not compatible and will result in SCIRun crashing. For this reason, if possible, try not to run SCIRun in this manner.

To display SCIRun remotely, the value of the **DISPLAY** environment variable must be set correctly on the local machine. Also, the local machine must be allowed to send display commands to the remote machine.

Normally, the remote machine makes a connection to the local machine via the ssh command. In this case, ssh sets the value of DISPLAY in such a way that the local machine has permission to send display commands to the remote machine. However, ssh connections result in poor display performance because of encryption activity on the connection.

To increase performance, the value of DISPLAY is set after establishing the ssh connection.

For a sh-style shell:

```
export DISPLAY=remote-machine-name:0.0
```

For a csh-style shell:

```
setenv DISPLAY remote-machine-name:0.0
```

Note that this technique defeats the encryption protection on the connection.

After overriding the value of DISPLAY set by ssh, the local machine will lack permission to send display commands to the remote machine. Use the xhost command on the remote machine to give permission to the local machine:

```
xhost +local-machine-name
```

1.1.5 Anatomy of the Main Window

The SCIRun main window consists of the Menu Bar, the Global View frame, the Message frame, and the Net Edit frame.

Menu Bar: The menu bar is used to load networks, save networks, quit SCIRun, create network modules, and perform other tasks. The menu bar consists of the following menu items:

File: The File menu contains the following items:

Load: Loads a network from a file

Insert: Adds a network to the NetEdit frame without overlap.

Save: Saves a network to a file

Save As...: Saves a network to a new file

Clear Network: Removes all modules and connections from the NetEdit frame

Select All: Selects all modules.

Execute All: Executes all modules.

Create Module Skeleton...: Contains the Create Module Skeleton option which guides a user through the creation of a basic module.

Quit: Quits SCIRun. Pressing key combination Control-q also quits. *Do not press Control-c to exit SCIRun: doing this will drop SCIRun into a debugger.*

Modules: This menu is composed of sub-menus. Each sub-menu corresponds to a category within the SCIRun package database. A category is a group of related modules. Each menu item in a category sub-menu creates a specific module and places it in the NetEdit frame. The NetEdit frame pop-up menu also provides access to the SCIRun and BioPSE (and possibly other) package menus. Activate the NetEdit frame pop-up menu by clicking the right mouse button (Button3).

SCIRun: The SCIRun menu is used to create modules (from the SCIRun package) for use in the Net Edit frame.

BioPSE: The BioPSE menu creates modules (from the BioPSEpackage) for use in the NetEdit frame. It consists of category sub-menus and module menu items.

Other Package Menus: There may be other package menus if other packages have been installed. They also have category sub-menus and module menu items.

Subnets:

Global View Frame: The Global View Frame is located in the lower right corner of the main window. It is used to navigate complex networks.

NetEdit Frame: The NetEdit Frame occupies the main window. It is used to build and execute networks.

Show Module:

Configure Frame: This dockable frame contains three tabs. Toggle expanding the Configure frame by pressing the **Configure** button. Toggle the frame's docking mode by pressing the button to the right of the Configure button.

Figure 1.4. SCIRun NetworkEditor configure options

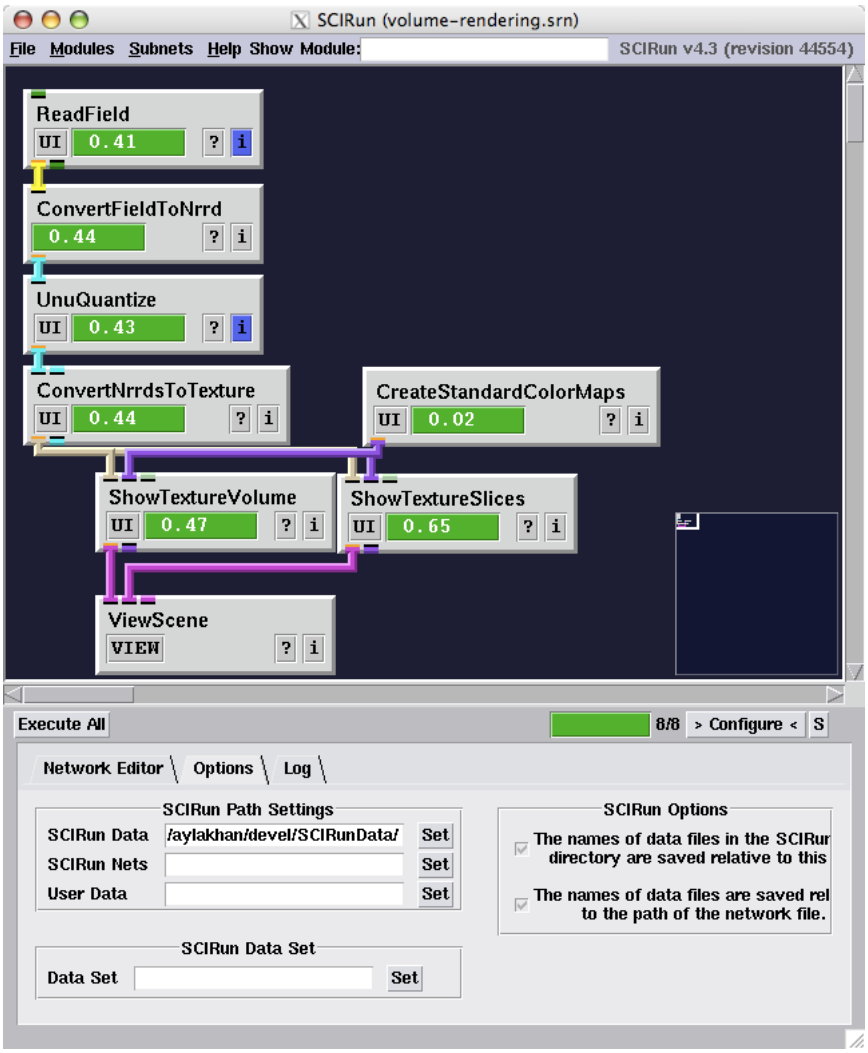

Figure 1.5. SCIRun configure options

Network Editor:

Log: Messages during program startup are displayed in the Message frame. The Message frame reports errors, warnings, and important information. Errors on startup may mean SCIRun has been installed incorrectly or has been installed from a buggy distribution; please report these errors.

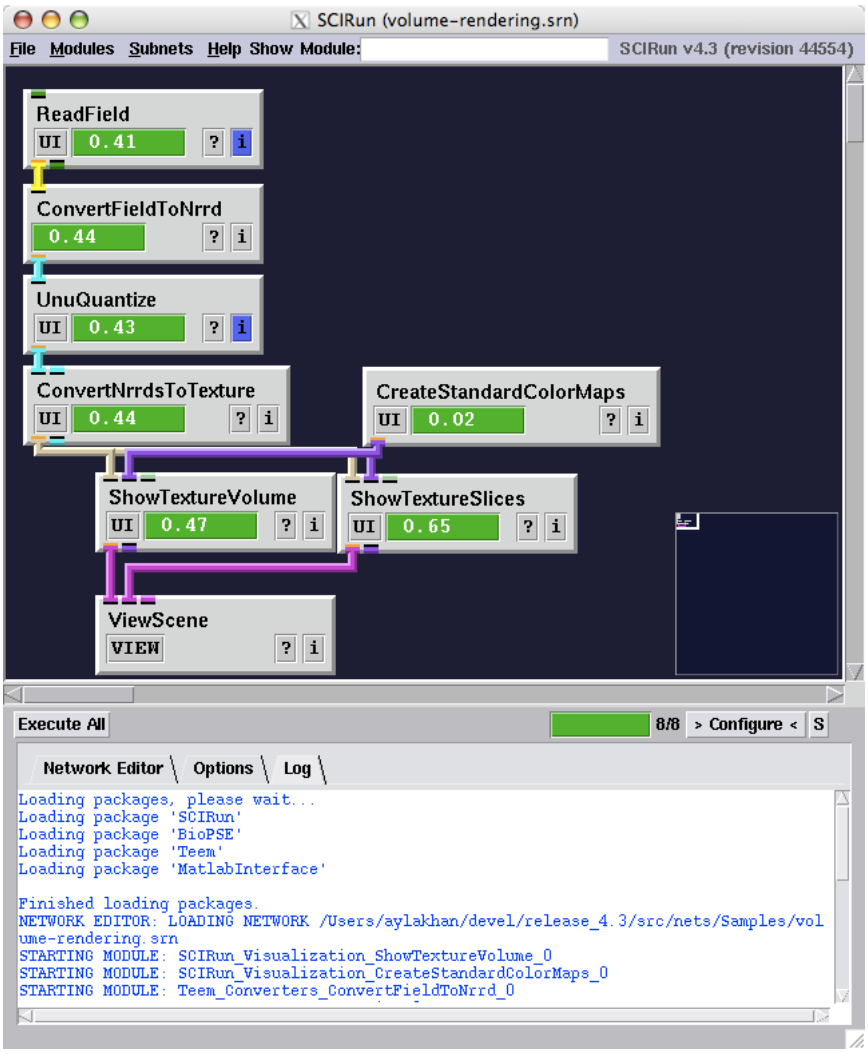

Figure 1.6. SCIRun log messages

1.1.6 The Terminal Window

After starting, SCIRun runs a shell-like application in the terminal window called the SCIRun shell. The SCIRun shell displays a prompt:

```
myhost:bin user$ ./scirun
SCIRun version 4.3 revision 44554
scirun>
```

This program is a modified Tool Command Language (Tcl) shell program. It is possible to type Tcl'ish SCIRun commands at the prompt. A later section describes use of the SCIRun shell.

SCIRun's Environment

SCIRun responds to a number of variables to alter its default behavior. They can be in two places: the .scirunrc file, or in environment variables.

When started by a user the first time, SCIRun creates a default version of .scirunrc in the user's home directory (for Unix-based environments) or in the directory that you compiled or installed scirun.exe in (for windows). Users may modify their default .scirunrc file.

SCIRun then processes the .scirunrc file on each subsequent startup. The .scirunrc file contains assignments to environment variables that affect SCIRun's behavior. Lines beginning with '#' are ignored.

Users may also set SCIRun related environment variables in their shell. Values of variables set in the shell override values set in .scirunrc.

See the content of .scirunrc for a complete list of SCIRun related environment variables. The following is a partial list of variables understood by SCIRun.

The Initialization File

The contents of **.scirunrc** look typically like:

```
#
#  For more information, please see: http://software.sci.utah.edu
#
#  The MIT License
#
#  Copyright (c) 2009 Scientific Computing and Imaging Institute,
#  University of Utah.
#
#
#  Permission is hereby granted, free of charge, to any person obtaining a
#  copy of this software and associated documentation files (the "Software"),
#  to deal in the Software without restriction, including without limitation
#  the rights to use, copy, modify, merge, publish, distribute, sublicense,
#  and/or sell copies of the Software, and to permit persons to whom the
#  Software is furnished to do so, subject to the following conditions:
#
#  The above copyright notice and this permission notice shall be included
#  in all copies or substantial portions of the Software.
#
#  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
#  OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
#  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
#  THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
#  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
#  FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
```

```
#  DEALINGS IN THE SOFTWARE.
#

#
# Sample .scirunrc file.
#
# This file is copied from the SCIRun src tree file:
#   ./SCIRun/src/scirunrc
#
# Lines beginning with '#' are commented out.
#
# All variables in this file have been commented out or set
# to their default values.
#
# This file contains a list of environment variables that can be either set
# in your shell or here. If you set these variables in your shell, they will
# override the values in this file.
#
# All variable declarations must use the syntax:
#    Environment_Variable_Name = Value
#
# Boolean variable declarations use this syntax:
#    Environment_Variable_Name = <true,1,on,yes|false,0,off,no>
#

#####################################
### ENVIRONMENT VARIABLES SECTION ###
#####################################

# REMEMBER, if these variables are set in your shell environment, they will
# override the valuse in this file.

### SCIRUN_ACCEPT_LICENSE
# Records whether license was accepted
# SCIRUN_ACCEPT_LICENSE = true

### SCIRUN_LOAD_PACKAGE = [comma-seperated list]
#   The format of SCIRUN_LOAD_PACKAGE is a comma seperated list of
#   package names.  This list specifies the package libraries to load
#   when SCIRun is first brought up.
# SCIRUN_LOAD_PACKAGE = BioPSE,Uintah,Teem


### SCIRUN_SHOW_HIDDEN_MODULES = [boolean]
```

```
#   Show modules that have been set to be hidden. These hidden modules include
#   obsolete or unfinished modules. Use extra care when using these modules
#   as they may be deleted in a future version or may cause scirun to crash
# SCIRUN_SHOW_HIDDEN_MODULES = true

### SCIRUN_DATA = [path]
#   Directory location that 'Reader' modules will default to.
# SCIRUN_DATA = /usr/sci/data/SCIRunData

### SCIRUN_DATASET = [name]
#   Name of data set in SCIRUN_DATA to default to.
# SCIRUN_DATASET = sphere


### SCIRUN_NET_SUBSTITUTE_DATADIR = [boolean]
#   If true, module filename variables will be automatically substituted for
#   'SCIRUN_DATADIR' and 'SCIRUN_DATASET' in saved .net files.
# SCIRUN_NET_SUBSTITUTE_DATADIR = false

### SCIRUN_NET_RELATIVE_FILENAMES = [boolean]
#   If true, SCIRun will save all filenames relative to the location of the
#   network file in the .srn files.
# SCIRUN_NET_RELATIVE_FILENAMES = true


### SCIRUN_MYDATA_DIR = [path]
#   Directory, or list of directories separated by colons,
#   that data 'Reader' and 'Writer' modules will list as
#   optional paths in the directories drop down menu.
# SCIRUN_MYDATA_DIR = $HOME/data:/scratch/data

### SCIRUN_NET_DIR = [path]
#   Directory, or list of directories separated by colons,
#   that the network editor will list as
#   optional paths in the directories drop down menu.
# SCIRUN_NET_DIR = [path]

### SCIRUN_TMP_DIR = [path]
#   Directory for writting temporary files - must be world read/writeable
# SCIRUN_TMP_DIR = /tmp

### SCIRUN_SERV_TMP_DIR = [path]
#   Directory for writting temporary files - must be world read/writeable
#   This is one is used by the external application interface. Both scirun
```

```
#   and scirunremote use this one to exchange files.
# SCIRUN_SERV_TMP_DIR = /tmp

### SCIRUN_CONFIRM_OVERWRITE = [boolean]
#   If false, SCIRun writer will not prompt the user before overwriting a file
# SCIRUN_CONFIRM_OVERWRITE = true


### SCIRUN_INSERT_NET_COPYRIGHT = [boolean]
#   If true, adds SCI copyright to any SCI .net files that are created.
# SCIRUN_INSERT_NET_COPYRIGHT = false

### SCIRUN_NOSPLASH = [boolean]
#   If true, SCIRun will not display the splash screen at startup.
# SCIRUN_NOSPLASH = false


### SCIRUN_HIDE_PROGRESS = [boolean]
#   If true, SCIRun will not display progress bars during startup and loading
# SCIRUN_HIDE_PROGRESS = false


### SCIRUN_FAST_QUIT = [boolean]
#   If true, SCIRun will not ask the user to save a network before quitting
# SCIRUN_FAST_QUIT = false

### SCI_REGRESSION_TESTING = [boolean]
#   If True, regression testing mode will turn on.  Execution starts after a
#   network is loaded and SCIRun will exit immediately after creating an image.
# SCI_REGRESSION_TESTING = false


### SCIRUN_REGRESSION_TESTING_TIMEOUT = [integer]
#   If set, and if SCI_REGRESSION_TESTING is True, SCIRun will automatically
#   kill itself after specified seconds.
# SCI_REGRESSION_TESTING_TIMEOUT = 300


### THREAD_NO_CATCH_SIGNALS = [boolean]
#   Turns off the SCI Signal handlers.
# THREAD_NO_CATCH_SIGNALS = true


### SCIRUN_DRAWARRAYS_DISABLE = [boolean]
```

```
#   Turns off use of glDrawArrays.  This should be set to true if you wish
#   to avoid drawing artifacts on _ATI_ Radeon cards under Linux or to fix
#   cross-platform remote display problems.
# SCIRUN_DRAWARRAYS_DISABLE = false


### SCIRUN_MPEG_LICENSE_ACCEPT = [boolean]
#   License information describing the mpeg_encode software can be found
#   in SCIRun's Thirdparty directory, in the mpeg_encode/README file:
#   This software is freely distributed.  That means, you may use it
#   for any non-commercial purpose.  However, patents are held by
#   several companies on various aspects of the MPEG video standard.
#   Companies or individuals who want to develop commercial products
#   that include this code must acquire licenses from these companies.
#   For information on licensing, see Appendix F in the standard.
#   For more information, please see the mpeg_encode README file.
#   If you are allowed to use the MPEG functionality based on the above
#   license, you may
#   enable MPEG movie recording in SCIRun (accessible via the SCIRun
#   Viewer's "File->Record Movie" menu) by setting the value of
#   SCIRUN_MPEG_LICENSE_ACCEPT to "true".
# SCIRUN_MPEG_LICENSE_ACCEPT = false

### SCIRUN_USE_DEFAULT_SETTINGS = [boolean]
#   Force the sizes for ShowField to calculate defaults when loading a new
#   Field.  Also Forces the Viewer to autoview whenever a new netork is
#   loaded.
#SCIRUN_USE_DEFAULT_SETTINGS = true

### DISABLE_MATLAB_PROCESS
# Disable forking of SCIRun on start of SCIRun to support Matlab running in
# separate process
# DISABLE_MATLAB_PROCESS = true

############################
# GUI PREFERENCES SECTION ##
############################

### SCIRUN_GUI_UseGuiFetch = [boolean]
#   Allows the user to tell a module to fetch the GUI from wherever it
#   happens to be on the screen and bring it to the mouse, and then
#   return it to its previous location.
SCIRUN_GUI_UseGuiFetch = on
```

```
### SCIRUN_GUI_MoveGuiToMouse = [boolean]
#   Makes GUIs appear near the mouse on GUI creation.
SCIRUN_GUI_MoveGuiToMouse = on


### SCIRUN_STRAIGHT_CONNECTIONS = [boolean]
#   Use straight lines to represent connections between modules in the
#   network editor.
# SCIRUN_STRAIGHT_CONNECTIONS = off

### SCIRUN_LARGE_MODULES = [boolean]
#   Use a large module layout instead of a smaller one
# SCIRUN_LARGE_MODULES = off


#################
# UNDOCUMENTED ##
#################
# LOGNAME
# USER
# SCI_DEBUG
```

1.2 SCIRun Data Variables

For convenience, the SCIRUN_DATA environment variable (and optionally SCIRUN_DATASET) can be set before starting SCIRun. Alternatively, use the –**datadir** flag when starting SCIRun.

SCIRUN_DATA: Points to the location of the SCIRunData directory on the system. Note: it is assumed the SCIRunData directory has already been downloaded (this is a separate download from the SCIRun build).

SCIRUN_DATASET: Indicates which dataset to load. For example, in SCIRunData, SCIRUN_DATASET can be aneurysm, sphere, utahtorso etc.

SCIRUN_DATAFILE:

Setting environment variables

Environment variables may be set as shown above in the SCIRUN_DATA section in Unix environments.

For windows environments, follow these directions: Click the Start button, select Control Panel, and then System. In the Advanced tab, click the Environment Variables button. Select New and enter the name and value of the environment variable. If it already exists, click on it and select Edit to modify it.

Chapter 2

SCIRun Networks

This section describes how to create, save, load, execute, and edit networks.

The following conventions are used when describing mouse and keyboard actions performed by the user:

Button1: Left mouse button

Button2: Middle mouse button

Button3: Right mouse button

Press: Press and hold a key or mouse button

Click: Press and release a mouse button

Type: Press and release a key

ButtonN-Drag: Press ButtonN, then drag the mouse.

Ctrl-ButtonN: Press the Control key, then press or click ButtonN

Ctrl-Key: Press and hold the control key, then click Key

2.1 Anatomy of a Module

A module is a single-purpose unit that functions within a dataflow environment. All modules are similarly represented by a graphic within the Network Editor frame. The graphical front end is the same for all modules and consists of the following elements:

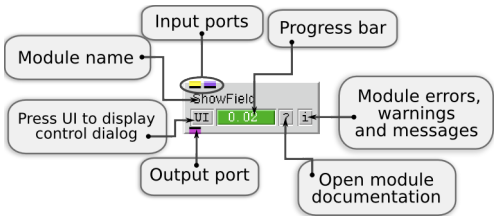

Figure 2.1. Anatomy of a SCIRun module

2.1.1 Input Ports

Modules have zero or more input ports for receiving data located at the top of the module. Modules have zero or more output ports for sending data located at the bottom of the module. Every module has at least one input or one output port. Each port corresponds to a data type, and each data type has a unique color. The table below maps port colors to data types. Input ports connect to other modules' output ports. Output ports connect to other modules' input ports. Connections can only be made between ports of the same type. Some modules have a dynamic input port. When a connection is made to a dynamic input port, a new instance of the port will be created.

Data Type	Port Color
Bundles	Orange
Color Maps	Purple
Field	Yellow
Field Array	Brown
Geometric Objects	Magenta
ITK Datatype	Pink
Matrices	Blue
Nrrd Data	Cyan
Strings	Green

2.1.2 Module UI

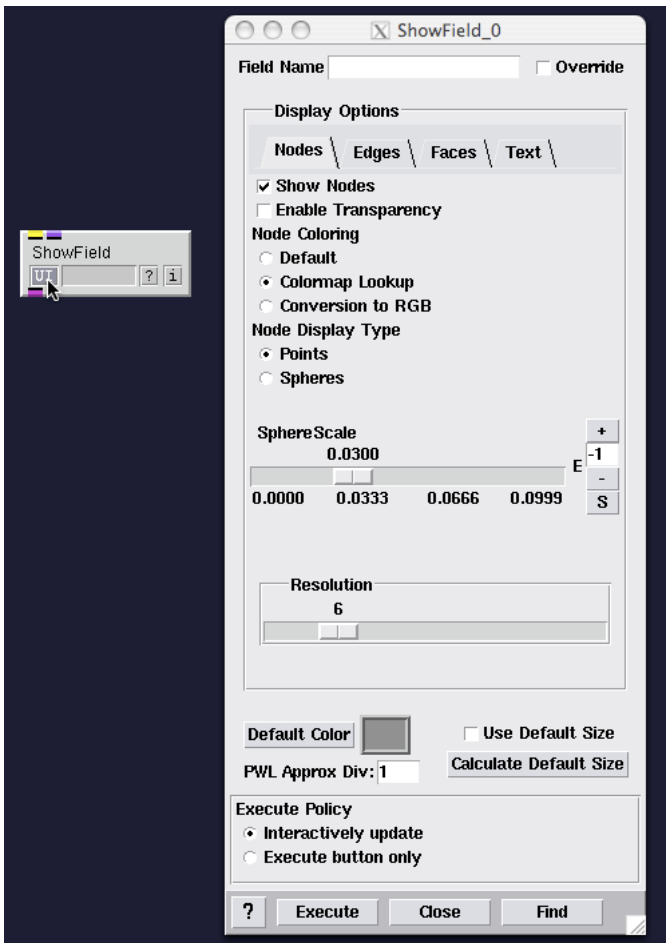

Figure 2.2. ShowField module UI

Pressing the UI button displays the module's control dialog. Some modules have no dialog, some have simple dialogs, and some have complex dialogs that allow elaborate control over the module. Figure 2.1.2 shows the control dialog for Show Field module.

2.2 Anatomy of a Connection

A connection lets data flow from one module's output port to (usually) another module's input port. Sometimes a connection will connect a module's output port to its input port.

Connections are colored grey if they are disabled.

Pressing Button3 while the mouse pointer is over a connection activates a connection's pop-up menu. The content of the pop-up menu changes depending on the state of the connection.

2.3 Creating a Module

To create a module, select its name from one of the package (e.g., SCIRun) menus' category sub-menus. Package menus are accessed from the main window's menu bar or from the NetEdit frame's pop-up menu. The NetEdit frame's pop-up menu is activated by pressing Button3 (right mouse button) while the mouse pointer is in the NetEdit frame (but not over a module or connection). The pop-up menu contains a list of category sub-menus from the SCIRun package and other installed packages. Each category sub-menu provides access to modules within the category.

After creating a module, its gui is placed in the NetEdit frame.

2.4 Setting Module Properties

To change a module's properties, click its UI button. This displays the module's control dialog. Use the dialog to change the module's properties. Each module's reference documentation explains the use of its control dialog. A module's reference documentation can be displayed by clicking a module's **?** button. The figure below shows the control dialog for the Show Field module.

2.5 Selecting Modules

Some operations (moving, destroying, disabling, and enabling) act on a group of selected modules (see Moving Module(s), Destroying Module(s), and Disabling Modules). Groups of selected modules can be created two ways:

1. Perform Button1-Drag while the pointer is in the NetEdit frame, but not over a module or a connection. This starts a selection box, allowing the user to select multiple modules. Modules overlapping the selection box are selected. Release Button1 to complete the selection. Additional groups of selected modules are created by performing Ctrl-Button1-Drag

2. Select the first module by clicking Button1 while the pointer is over a module. Then add modules to the group by clicking Ctrl-Button1 on additional modules. All previously selected modules remain selected.

Selected modules are colored differently from unselected modules.

2.6 Destroying Modules

To delete a module, choose menu item Destroy from a module's pop-up menu.

Multiple modules can be deleted at one time. Select one or more modules, then choose menu item Destroy Selected from a module's pop-up menu.

2.7 Moving Modules

Modules can be moved in the NetEdit frame. To move a module, perform Button1-Drag while the pointer is over a module, and move the module to its new location.

Multiple modules can be moved at one time. Select one or more modules, then perform Button1-Drag while the pointer is over any one of the modules in the selected group.

Two modules may be moved by dragging one of the connections between them. To move two modules, perform Button1-Drag when the pointer is over a connection between two modules.

The NetEdit frame will scroll when a module is moved to the frame's edge.

2.8 Disabling and Enabling Modules

Modules may be temporarily disabled. Disabled modules do not execute and data does not flow through them.

To disable modules, select one or more modules, then choose menu item Disable from a module's pop-up menu.

Disabling a module disables all incoming and outgoing connections to the module and prevents the module from executing during execution of the network. Disabled modules are drawn in a darker shade of grey.

To enable disabled modules, choose Enable from a disabled module's pop-up menu. Enabling one module enables all selected modules.

See also Disabling/Enabling Connections.

2.9 Editing and Displaying Module Notes

Module notes allow the user to document the purpose of each module by attaching notes to each module in the NetEdit window. Notes can be displayed or hidden.

To create or edit module notes, choose menu item Notes from a module's pop-up menu. The note editor dialog can also be activated by clicking Button1 on notes displayed in the NetEdit window. Enter new notes or edit existing notes in the note editor dialog.

Notes may be hidden, displayed in tooltips, or displayed in the NetEdit frame. The note editor provides the following note display options: Default, Tooltip, Top, Left, Right, or Bottom. Choose a display mode by clicking Button1 on the appropriate button.

Choose Tooltip to display notes as tooltips. Notes are displayed only when the mouse pointer hovers over a module.

Choose one of Default, Top, Left, Right, or Bottom to display notes in the NetEdit to the right, top, left, right, or bottom of the module respectively.

Choose None to hide module notes. Displayed notes can be hidden by clicking Button2 when the pointer is on the notes. Hiding notes does not delete notes.

Click button Text Color to change the color of notes displayed in the NetEdit frame.

Click button Cancel to abort note editing.

Click button Clear to erase notes.

Click button Done to accept edited notes.

2.10 Viewing a Module's Log

Each module supports a message log. The module writes error messages or other types of messages to its log.

To display a module's log, choose Show Log from a module's pop-up menu or click Button1 on a module's message indicator.

2.11 Creating a Connection

To connect the output (input) port of one module to the input (output) port of another module, use Button2.

To make a connection, position the mouse pointer over a module's input (output) port. Then perform Button2-Drag and drag the mouse pointer toward another module's output (input) port.

When Button2 is pressed, the program shows all valid connections as black lines. It also draws one red colored connection, which is the connection made if the drag is stopped by releasing Button2.

Make the connection by releasing Button2 when the pointer is over the desired destination port, or when the red colored connection is the desired connection. The connection is drawn using the color corresponding to the connection's data type.

Users can connect a module's output port to the input ports of one or more modules by repeating the procedure just described.

2.12 Editing and Displaying Connection Notes

Connection notes allow the user to document the purpose of each connection by attaching notes to each connection in the NetEdit window. Notes can be displayed or hidden.

To create or edit connection notes, choose menu item Notes from a connection's pop-up menu. Enter new notes or edit existing notes in the note editor dialog. The note editor dialog can also be activated by clicking Button1 on notes displayed in the NetEdit window.

Notes may be hidden, displayed in tooltips, or displayed in the NetEdit window. The note editor provides the following note display options: Default, Tooltip and Top. Choose a display mode by clicking Button1 on the appropriate button.

Choose ToolTip to display notes as tooltips–notes are displayed only when the mouse pointer hovers over a connection.

Choose Default or Top to display notes on top of the connection.

Choose None to hide connection notes. Displayed notes can also be hidden by clicking Button2 when the pointer is over the notes. Hiding notes does not delete notes.

Click button Text Color to change the color of notes displayed in the NetEdit frame.

Click button Cancel to abort note editing.

Click button Clear to erase notes.

Click button Done to accept edited notes.

2.13 Highlighting Related Connections

To highlight the tree of connections affecting a connection, perform Ctrl-Button1 anywhere on a connection.

To highlight the tree of connections downstream an output port, perform Ctrl-Button1 on the output port.

To highlight the tree of connections upstream an input port, perform Ctrl-Button1 on the input port.

2.14 Disabling a Connection

To disable a connection, click Button3 on the connection to bring up the connection menu and select Disable. The connection appears grey.

Disabling a connection prevents data from flowing through, as if it were not connected.

2.15 Deleting a Connection

To delete a connection, choose menu item Delete from a connection's pop-up menu or click Ctrl-Button2 while the pointer is on a connection.

2.16 Undoing/Redoing a Connection

Type Ctrl-z to undo the last connection creation or deletion. Undo can be repeated.

Type Ctrl-y to redo the last undone connection creation or deletion.

2.17 Executing a Network

"Network Execution" means one or more modules must be executed in a coordinated fashion. SCIRun's scheduler manages the coordinated execution of modules.

2.17.1 The Basics

The scheduler is invoked when an event triggers a module's execution. The scheduler creates a list of all modules that must execute in coordination with the triggered module. Modules upstream (directly or indirectly) from the triggered module are put on the execution list if they have not previously executed. All modules downstream from the triggered module are put on the execution list. Once the scheduler determines which modules must be executed, it executes them (in parallel where possible).

Network execution is mostly transparent. That is, events that trigger module execution usually generate automatically. Sometimes, however, the user must manually generate a triggering event by choosing the Execute item from a module's pop-up menu.

2.17.2 Details

Each module executes in its own thread and blocks (waits) until its upstream modules can supply it with data. After a module completes its computation, it sends the results to its downstream modules. This completes a module's execution cycle. The module will not have another chance to receive data from its input ports and send data to its output ports until some event puts the module back on the scheduler's execution list.

This behavior prevents modules from computing in an iterative fashion (sending intermediate results to their downstream modules), because downstream modules cannot receive results until they are in their execution cycle. Downstream modules would need to be executed each time the upstream module posts an intermediate result.

2.17.3 Intermediate Results

Some modules are designed to be used in an iterative fashion. They use a method called send_intermediate to send the results of each iteration. When this method is used, the scheduler (re)executes downstream modules each time the upstream module posts its next result. Downstream modules are able to receive the results of each iteration as soon as the upstream module sends them.

Modules SolveMatrix and MatrixSelectVector (from the SCIRun package and Math category) are examples of modules that compute iteratively using the send_intermediate method.

2.17.4 Feedback Loops

Some modules are designed to be used in an iterative fashion. They use a method called send_intermediate to send the results of each iteration. When this method is used, the scheduler (re)executes downstream modules each time the upstream module posts its next result. Downstream modules are able to receive the results of each iteration as soon as the upstream module sends them.

Modules SolveLinearSystem and GetColumnOrRowFromMatrix (from the SCIRun package and Math category) are examples of modules that compute iteratively using the send_intermediate method.

2.18 Documenting a Network

It is useful to document the function of a network. A network's note pad is used for this purpose. To edit the network's note pad, select the Network Properties item from the main window's File menu. This displays the network's note pad editor. The editor allows the user to write notes on the purpose and use of the network.

2.19 Saving a Network

SCIRun can save networks to files. Network files have an extension of **.srn** (in the past they also had .net, .sr and .uin extensions).

To save a network, select item Save from the main window's File menu. A file browser dialog prompts for the name and location of the network file. If changes are made to an existing network, Save saves changes made to an existing net file. Save an existing network under a new name using the Save As... menu item. A file browser dialog prompts for the new name of the network file. Subsequent uses of Save saves changes to the newly created file.

The position and size of open Viewer windows are saved to the network file.

A backup network will be saved in the form #filename# each time a module executes (or an Execute All is done). If no filename has been specified, SCIRun will save the current network to #MyNetwork.svn# to the current directory.

Network files are XML files. These files can be edited, however, reasons for doing so are beyond the scope of this guide.

2.20 Loading a Network

To load a network file, select the Load item from the main window's File menu. A file browser dialog prompts for the name and location of the network file. The loaded network replaces an existing network in the NetEdit frame.

2.21 Inserting a Network

To avoid merging networks, select the Insert item from the main window's File menu. This option allows the user to place one SCIRun network next to another, avoiding overlap. A file browser dialog prompts the user for the name and location of the network file.

The new network is inserted into the upper left corner of the NetEdit frame. If a network of modules already exists in the NetEdit frame, the Insert command places a new network to the immediate right of the existing network.

2.22 Clearing a Network

To remove all modules and connections from the NetEdit frame, select the Clear item from the main window's File menu. A text box appears, confirming whether the user wants to proceed with or cancel the clearing operation.

2.23 Navigating a Network

A complex network may not be entirely visible in the NetEdit frame. Use the NetEdit frame's scroll bars or the network view tool to view complex networks.

The Global View Frame shows the entire "network world". The small rectangular region (outlined in black) in the Global View Frame is the network view tool and a window on the network world. The position of the view tool determines the part of the network visible in the NetEdit frame. To view other parts of the network, press Button1 while the pointer is anywhere in the Global View Frame: this moves the tool to the location of the pointer - then drag the tool to the new location.

2.24 Creating a Sub-Network

A sub-network is a group of modules that are treated as a single module. In a network, you may use a sub-network as you would a module.

To create a sub-network, select one or more modules, then choose item Create Sub-Network from any selected module's pop-up menu. Selected modules will be replaced by a sub-network graphic and the sub-network editor will activate. Sub-networks may be created in sub-networks. The section Editing a Sub-Network explains the use of the sub-network editor.

A sub-network editor button replaces the CPU time and Progress Bar in the sub-network graphic. A sub-network's editor is activated by pressing the sub-network editor button.

Pressing button UI activates control dialogs of all modules in a sub-network. Control dialogs can be activated individually in the sub-network editor.

In a network, a sub-network behaves as does a module. A sub-network has input and/or output ports (orginating from modules within the sub-network) that can be connected to modules outside of the sub-network. Each sub-network has a pop-up menu and a set of editable notes (see Editing and Displaying Module Notes). A sub-network does not have a log. Logs of individual modules in the sub-network can be viewed in the sub-network editor window (see Viewing a Module's Log).

2.25 Expanding a Sub-Network

Expanding a network reverses the action of menu item Create Sub-Network–all modules in a sub-network are returned to the network containing the sub-network and the sub-network is destroyed. This action cannot be undone.

To expand a sub-network choose menu item Expand Sub-Network from the sub-network's pop-up menu.

2.26 Editing a Sub-Network

The sub-network editor is activated by pressing a sub-network's editor button. The sub-network editor works similarly to the NetEdit window. All network editing features of the NetEdit window are available in the sub-network editor–modules can be created, connections can be made, etc.

Sub-networks have input and/or output ports. These are created in the sub-network editor. Sub-network input ports are created by making a connection between the input port of a module (in the sub-network) and the top edge of the sub-network editor window. Sub-network output ports are created by making a connection between the output port of a module (in the sub-network) and the bottom edge of the sub-network editor window.

A sub-network can be renamed by entering its new name into the Name text entry widget.

2.27 Saving a Sub-network

Sub-networks can be saved for reuse in networks and sub-networks.

Sub-networks are saved in directory SCIRun/Subnets, in the user's home directory.

To save a sub-network, press the sub-network's editor button, which activates the sub-network's editor, then press the editor's Save Template button.

Saved sub-networks are listed as menu items in the Sub-network package menu (which is available in the NetEdit window's File menu and popup menu).

Loading a Sub-Network gives instructions for loading sub-networks into networks and sub-networks.

2.28 Loading a Sub-Network

Saved sub-networks are loaded into networks and sub-networks by selecting their names from the Sub-network package menu. Package menus are accessed from the main window's menu bar or from the NetEdit frame's pop-up menu. Unlike other package menus, the Sub-network menu has no category sub-menus.

A sub-network may be loaded one or more times in a network. Each time a sub-network is loaded a new instance of it is created in the network.

2.29 The SCIRun Shell

After starting, SCIRun runs a shell-like application in the terminal window. This shell displays the prompt `scirun>` in the terminal window.This program is a Tool Command Language (Tcl) shell program extended with SCIRun specific commands.

It is possible to type Tcl SCIRun commands at the prompt.

Chapter 3

SCIRun Objects

3.1 Mesh

SCIRun meshes are classified as unstructured, structured, or regular. A mesh consists of nodes and implicit or explicit node connectivity information.

Node locations and connectivities for unstructured meshes are specified explicitly. Node locations are specified explicitly, and connectivities are known implicitly for structured meshes. Node locations and connectivities are known implicitly for regular meshes.

A regular mesh is more constrained than structured and unstructured meshes. Likewise, a structured mesh is more constrained that a unstructured mesh. A more constrained mesh can be trivially converted to a less constrained mesh by explicitly enumerating implicit properties. For instance, a LatVol mesh is converted to a StructHexVol by enumerating a LatVol mesh's node coordinates (connectivities are still implied). Also, a LatVol mesh is converted to a HexVol mesh by enumerating a LatVol's node locations and connectivities. Module ToStructured performs this type of conversion.

It is not meaningful to convert a less constrained mesh to a more constrained mesh. It is possible, however, to sample an unstructured field at regular intervals to create a regular field. For example, a HexVolField is interpolated onto a LatVol mesh using modules SampleLattice and DirectInterpolate.

3.1.1 Unstructured Meshes

CurveMesh

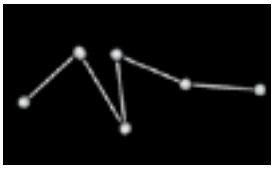

Figure 3.1. Curve Mesh

A curve mesh is a segmented curve.

HexVolMesh

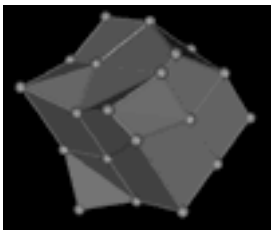

Figure 3.2. HexVol Mesh

A hex volume mesh is a subdivision of space into hexagonal elements.

PointCloudMesh

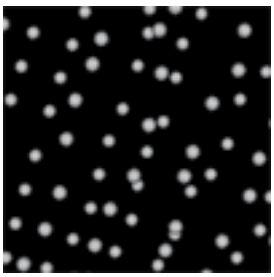

Figure 3.3. Point Cloud Mesh

A point cloud mesh is a set of unconnected points.

PrismVolMesh

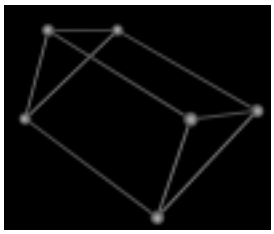

Figure 3.4. PrismVol Mesh

A Prism Volume mesh is a subdivision of space into prism elements. Prism elements have five faces, two trangular faces connected by three quadrilateral faces.

QuadSurfMesh

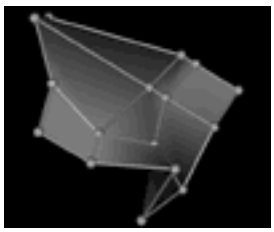

Figure 3.5. QuadSurf Mesh

A surface made of connected quadrilaterals.

TetVolMesh

Figure 3.6. TetVol Mesh

A tet volume mesh is a subdivision of space into tetrahedral elements.

TriSurfMesh

Figure 3.7. TriSurf Mesh

A tri surface mesh is a surface made of connected triangles.

3.1.2 Structured Meshes

StructCurveMesh

StructHexVolMesh

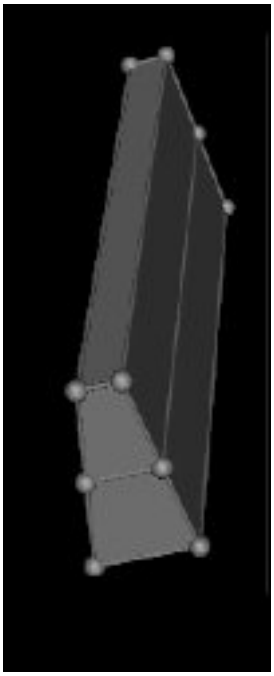

Figure 3.8. StructHexVol Mesh

A subdivision of space into structured hexagonal elements.

StructQuadSurfMesh

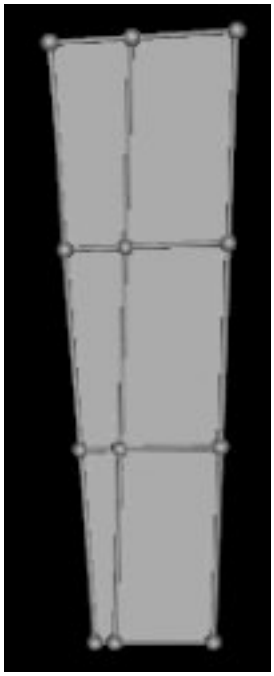

Figure 3.9. StructQuadSurf Mesh

A surface made of connected quadrilaterals on a structured grid.

3.1.3 Regular Meshes

ImageMesh

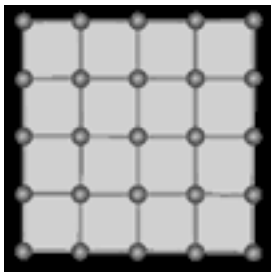

Figure 3.10. Image Mesh

An Image mesh is a regular 2D grid. Note that an Image mesh is not used for image processing.

LatVolMesh

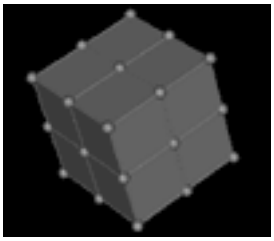

Figure 3.11. LatVol Mesh

A lattice volume mesh is a regular 3D grid.

ScanlineMesh

Figure 3.12. Scanline Mesh

A scanline mesh is a regularly segmented straight line (a regular 1D grid).

3.2 Field

A Field contains a geometric mesh, and a collection of data values mapped on to the mesh. Data can be stored at the nodes, edges, faces, and/or cells of the mesh.

3.3 Storing types in SCIRun Field

The dimensionality of the mesh type determines the available storage locations. For example, a TriSurfMesh has nodes, edges, and planar faces, but not cells, which are assumed to be three-dimensional elements. As a result, a TriSurf cannot store data in cells, but can store data in edges or faces.

Data types can be tensor, vector, double precision, floating point, integer, short integer, char, unsigned integer, unsigned short integer, unsigned char...

3.4 Matrix

SCIRun supports the DenseMatrix, DenseColMajMatrix, SparseMatrix, SparseRowMatrix, ColumnMatrix subtypes.

ColumnMatrix An Mx1 mathend000# matrix using M mathend000# storage units.

DenseMatrix An MxN mathend000# matrix using MxN mathend000# storage units.

SparseRowMatrix A MxN mathend000# matrix where most elements are zero and no storage is allocated for zero valued elements.

3.5 ColorMap

The SCIRun ColorMap type is a mapping of color values to data values.