AVS USER'S GUIDE

Release 4
May, 1992

Advanced Visual Systems Inc.

Part Number: 320-0011-02, Rev B

NOTICE

Advanced Visual Systems Inc.
300 Fifth Ave.
Waltham, MA 02154

TABLE OF CONTENTS

1 Introduction to AVS

2 Importing Data into AVS

3 Starting AVS

4 Image Viewer Subsystem

5 Geometry Viewer Subsystem

6 Network Editor Subsystem

8 Advanced Network Editor

A AVS on Color X Servers

B Geometry Viewer Script Language

List of Tables

Acknowledgement

The data portayed in the Frontispieces to Chapters 4 and 7 was used courtesy of the UCLA Department of Radiological Sciences.

CHAPTER 1

INTRODUCTION TO AVS

Introduction

The increasing power of supercomputers and graphics systems has made it possible for the scientific and engineering communities to gain new insight into their disciplines. In areas as diverse as fluid dynamics, computer-aided engineering, molecular modeling, and geophysics, researchers are applying these powerful systems to analyze and view their data, using real-time interactive display techniques.

A limiting factor in this growing field has been the existing software tools, which require specialized programming expertise and great expense, both in time and in money. The **Application Visualization System** (AVS) addresses this problem, allowing researchers to apply the hardware power to their problems without requiring programming expertise or a great investment of time.

AVS users can construct their own visualization applications, by combining software components into executable *flow networks*. The components, called *modules*, implement specific functions in the visualization cycle:

- **Filtering** the basic data into a more usable form (more informative, smaller, etc.)
- **Mapping** the filtered data into either: geometric primitives (triangles, lines, spheres, etc.) that when combined together produce a three-dimensional *geometry* representation of the data; or mapping the data into pixels (dots of color) that when combined together produce a two-dimensional picture or *image* representation of the data.
- **Rendering** the 3D geometries or 2D images into pictures on the display screen.

The flow networks are built from a menu of modules by using a direct-manipulation, visual programming interface called the AVS **Network Editor**. With the Network Editor, the user produces an application by selecting a group of modules and drawing connections between them. In many cases, users can construct an entire visualization application, using standard modules and without resorting to traditional procedural programming.

The user views, organizes, and further processes the output of a network through one of the AVS subsystems. The **Geometry Viewer** displays 3D geo-

metric objects. The **Image Viewer** displays 2D images. The **Graph Viewer** creates XY and contour graphs of data.

AVS includes a rich set of modules for construction of networks. AVS also recognizes that it is in the nature of scientific research and visualization to go beyond the bounds of an application. AVS allows users to create their own new modules to meet their specific needs and dynamically load them into AVS networks. The AVS **Module Generator** can be used to automatically generate module code in C or FORTRAN. Users need not have detailed knowledge of the AVS implementation or expertise in disciplines outside their areas of interest.

Modules are "software building blocks" with well-defined interfaces, written either in FORTRAN or in C. The overall structuring of the application is handled on the AVS level; the computational details are handled within modules as FORTRAN or C procedures.

Modules take typed data as inputs and produce typed data as outputs. The basic data types in the system are oriented toward scientific data manipulation and graphic display. These types include:

- 1D, 2D, and 3D grids of numbers with scalar values or vectors of byte, integer, or floating-point values at each grid point. The grids can be regular (*uniform*) or nonregular—where the distance between the grid points is variable (*rectilinear*). It is also possible for the grid to describe a curved or arbitrarily deformed space (*curvilinear*), or an arbitrary list of points in 3D space (*scatter data*). This type of data is called a *field*.
- unstructured cell data
- geometric data
- images
- molecular data

For more information on this subject, see the "Importing Data into AVS" chapter.

In addition to input and output data, modules also have *parameters* that control the module's computation. Once the structure of the application has been established, AVS executes the network, allowing the user to interact with the application by navigating through the network diagram and interacting with various modules through their individual parameters. AVS generates the "control panel" user interface to a module automatically, by associating parameters with either graphical control panels (buttons, sliders, etc) or peripheral input devices (dial boxes, spaceball, etc.).

The remainder of this chapter presents an overview of the AVS approach to the challenge of scientific visualization.

Scientific Visualization Techniques

AVS implements two basic strategies for translating numerical data into color images. In the *pixel-based* method, data points become pixels, more or less directly. In the *geometry-based* method, the numerical data is converted to descriptions of 3D geometric objects. These are, in turn, turned into color images by the machine's low-level graphics software and rendering hardware.

These two strategies are described further in the sections that follow.

Pixel-Based Visualization

The essence of the pixel-based visualization strategy is simple: take a "raw" data value and translate it into a number that represents a color. In AVS, this translation is accomplished with a table lookup, called a *colormap*. You can define, save, and retrieve your own colormaps. AVS includes an interactive Colormap Editor drawing tool for generating colormaps conveniently and quickly.

Colormap Lookup

An AVS colormap is usually a 256-row table; each row specifies a 24-bit "true-color" value (and, optionally, an 8-bit auxiliary field that defines opacity / transparency), as shown in Figure 1-1. A colormap lookup consists of using an input value to select a particular row of the table. The color value in that row is the result of the lookup.

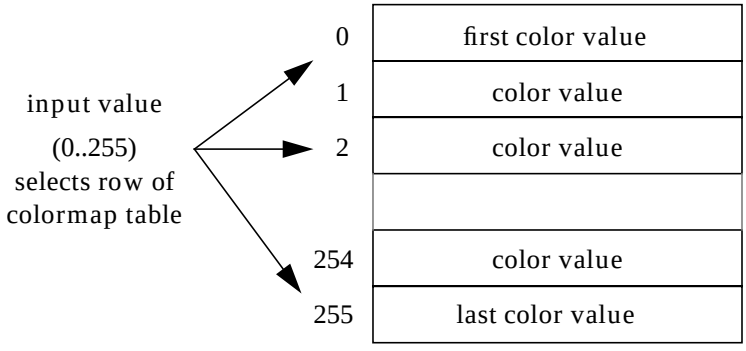


Figure 1-1 AVS Colormap Lookup

By default, AVS colormaps accept *byte* data as input values. Each byte is considered to be an unsigned integer (0..255) which specifies a particular row of the table. However, you can also have AVS automatically scale the colormap to any integer or floating point data range. Where colors are not desired, a gray-shade colormap can be used.

AVS colormaps are independent of the hardware colormaps used by low-level graphics software. All AVS colormaps produce 24-bit "true color" output. If necessary, further translation takes place automatically—for instance, to produce images on a display with only 8 or 12 color planes.

Further Pixel Processing

If multi-dimensional data is converted to pixels, the results must somehow be reduced to 2D before they can be displayed as an image onscreen. AVS provides several ways to perform such reductions:

Slicing
A 2D cross-section can be made through a 3D block of pixels (**orthogonal slicer** module).

Blending
If a 3D block of pixels is passed though a colormap whose auxiliary field contains opacity / transparency data, pixels can be blended along the line of sight. This produces a 2D picture of what appears to be a solid or semi-transparent object in space (**tracer** module).

High-Quality Pixel-Based Visualization

In simple pixel-based visualization, each data point corresponds to a single pixel. When the user "zooms in" on a particular portion of the image, the magnification is performed by pixel replication. (For instance, a single pixel value may be used throughout a 6x6 patch in a zoomed image.)

A variety of techniques can be used to improve image quality: high-order interpolation of data values, antialiasing of pixel values, 3D texture mapping, etc. In addition, 3D graphics techniques such as lighting, shading, and perspective viewing can be used to compute the interpolated pixel values.

Geometry-Based Visualization

AVS's other strategy for turning numbers into pictures brings all the power and flexibility of interactive 3D graphics to the visualization arena. The raw data values (or, more likely, a subset of the values) are mapped into the vertices of geometric objects. The values are used to assign colors to the vertices, using AVS colormaps. Then, the graphics subsystem creates color images from the geometric descriptions.

There are many techniques for creating geometric descriptions, or *geometries*, from raw data. For instance:

- Represent each atom of a molecule as a sphere. Assign color and transparency to the sphere based on the type of atom.
- Given a set of data that specifies the temperature at many points within a volume, use all the points at a given temperature to define an *isosurface* (**isosurface** module).

- Given a set of data that specifies the wind velocity at many points within a volume, use arrows to represent the velocity at each point on an arbitrary plane within the volume (**hedgehog** module).
- Given wind velocity data as above, construct flow lines to represent the motion of an object through the field (**stream lines** module).

The AVS Subsystems

AVS's main menu shows its major subsystems:

Image Viewer
The **Image Viewer** subsystem is a high-level tool for manipulating and viewing images.

Graph Viewer
The **Graph Viewer** subsystem is a tool for creating 2D linear and contour graphs of data.

Geometry Viewer
The **Geometry Viewer** subsystem allows you to view and interact with geometrically-defined objects. The objects must have been created by programs or AVS modules that use AVS's *geom* programming library. You can transform the objects themselves (move, rotate, scale); you can change the viewing parameters (e.g. move the eye point, perspective view, etc.); and you can control the way in which the graphical images are rendered (lighting and shading, Z-buffering, etc.). Multiple objects, such as an isosurface and a slice plane, can be combined into a single scene depicted in a display window.

3D graphics rendering techniques (lighting models, 2D and 3D texture mapping, automatic removal of hidden surfaces, sphere rendering, etc.) rely heavily on the underlying capabilities of an individual platform's graphics subsystem, both hardware and software. Platforms differ in their support of these techniques. AVS attempts to use all of the graphics functionality present on a platform. Where hardware graphics rendering is not available, AVS uses a software renderer.

Network Editor
The **Network Editor** subsystem is a visual programming interface for connecting computational modules together into networks to perform visualization functions. *Modules* and *networks* are discussed in the sections that follow.

Each of these subsystems is described in its own chapter later in this manual. In addition, the main menu includes an **Applications** option:

Applications
Applications produces another menu of choices. AVS comes with two prepared applications, the AVS **Demo** suite, and the **Data Viewer**. Both

applications are primarily useful to the new AVS user learning visualization techniques, terminology, and the AVS interface.

The **Demo** suite provides push-button access to a series of demonstration scripts that illustrate the AVS interface (Geometry Viewer, Image Viewer, Network Editor, and Graph Viewer); scientific visualization techniques; AVS as it can be applied in various fields such as medical imaging and computational fluid dynamics; and AVS modules. The **Demo** suite is described in the *AVS Tutorial Guide*.

The **Data Viewer** is a simplified user interface to AVS's most commonly-used scientific visualization techniques. It provides a pulldown menu interface from which the user selects input, filtering, mapping, and data output techniques. Using the sample datasets in the */usr/avs/data* directory, you can perform significant visualization functions with just a few clicks of the mouse. The Data Viewer is described in the *AVS Applications Guide*

Users can add their own applications to the **Applications** menu (see the "Starting AVS" chapter).

Command Language Interpreter

The Image Viewer, Geometry Viewer, Graph Viewer, and Network Editor can also be driven through a Command Language Interpreter (CLI). You can type CLI commands in response to a prompt and interactively view the results, you can create a *command script* file that executes automatically, and you can write a module that sends CLI commands to the Image Viewer, Geometry Viewer, Graph Viewer, and the Network Editor via the AVS kernel.

The CLI also supports a *journaling* facility in the Network Editor. With journaling switched on, AVS will record most of your actions in the Network Editor into an ASCII file than can be edited to produce a automated demonstration. This technique was used to produce the illustrative scripts stored in the */usr/avs/demo/man_scripts* directory, accessible through the Help facility's **Help Demos** button.

See the "Command Language Interpreter" chapter in the *AVS Developer's Guide*.

AVS Modules

The *module* is the AVS computational unit. Each module accepts data as input and generates other data as output. To create an AVS application, you connect a group of modules into a *network*. The connections represent the flow of data among the modules. Typically, the data originates in one or more disk files, but it can also be supplied by an "external" program, running on the same machine or on another machine in the local network. The data is typically transformed into one or more images by a collection of modules, and finally is

displayed in a window onscreen. Figure 1-2 shows a simple network of modules.

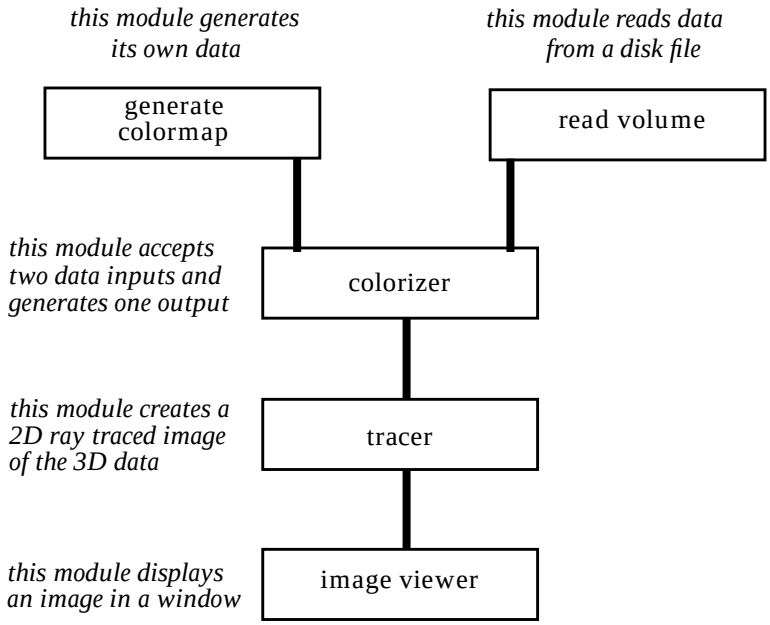


Figure 1-2 Simple Network of AVS Modules

AVS modules can execute locally on the same host system as the AVS program, or modules can execute *remotely*—on another host of the same (homogeneous) or different (heterogeneous) hardware type that runs AVS. An AVS module, compiled, linked, and stored on the remote host, is easily added to any AVS network. This remote module might be a data read and transform process, or a simulation that executes most efficiently on a network compute server.

The remainder of this section discusses the characteristics of individual AVS modules. Networks of modules are discussed in the following section.

Modules: Ports and Parameters

Each AVS module is designed to be a powerful, flexible, easy-to-use processing component. A module is general in its functionality, so that you can use it in a variety of application contexts. Each module does a substantial amount of processing, so that networks need contain only small number of modules to do real, useful work.

You can include a particular module in any number of AVS applications (*networks*); you can even include the same module more than once in a single network.

The key to the modular approach to application building is that each module has a simple, consistent interface, which includes:

- A set of *data inputs* (some optional, some required).
- A set of *input parameters* that control the way the module processes its input data or determines which data to use. One of AVS's most powerful features is that you can change parameter values interactively as a network executes. Input parameters can themselves be made into data input ports and receive values from other modules.
- A set of *data outputs*.

Some modules have no input ports at all. Such modules create their own data, or read data in from a source that is external to the AVS network (e.g. a disk file).

When you use AVS to create a network, each module's interface is represented visually by a *module icon* (Figure 1-3) and a *control panel* (Figure 1-4). The

Figure 1-3 Module's Interface: Icon

module icon is a rectangle, labeled with the module's name. Each data input is represented by an *input port* along the top edge. Each data output is represented by an *output port* along the bottom edge. Each input parameter is represented by a *control* widget (slider, dial, etc.); the controls are assembled in a separate *control panel* window.

Data Inputs

A module accepts one or more data sets as input. Each data set must be of a particular AVS data type: *field*, *colormap*, etc. The module doesn't care where its input data comes from, only that the data types are correct.

Table 1-1. Module Input Ports/AVS Data Types

Port Color	Data Type
red	geometry
yellow	colormap
light blue	pixmap
multi-color	field
orange	unstructured cell data
magenta	molecule data type
light purple	integer
dark purple	floating point
green	string
white	user defined data

Each data input is represented on the module icon by a color-coded input port, along the top edge of the icon. The color indicates the type of data that

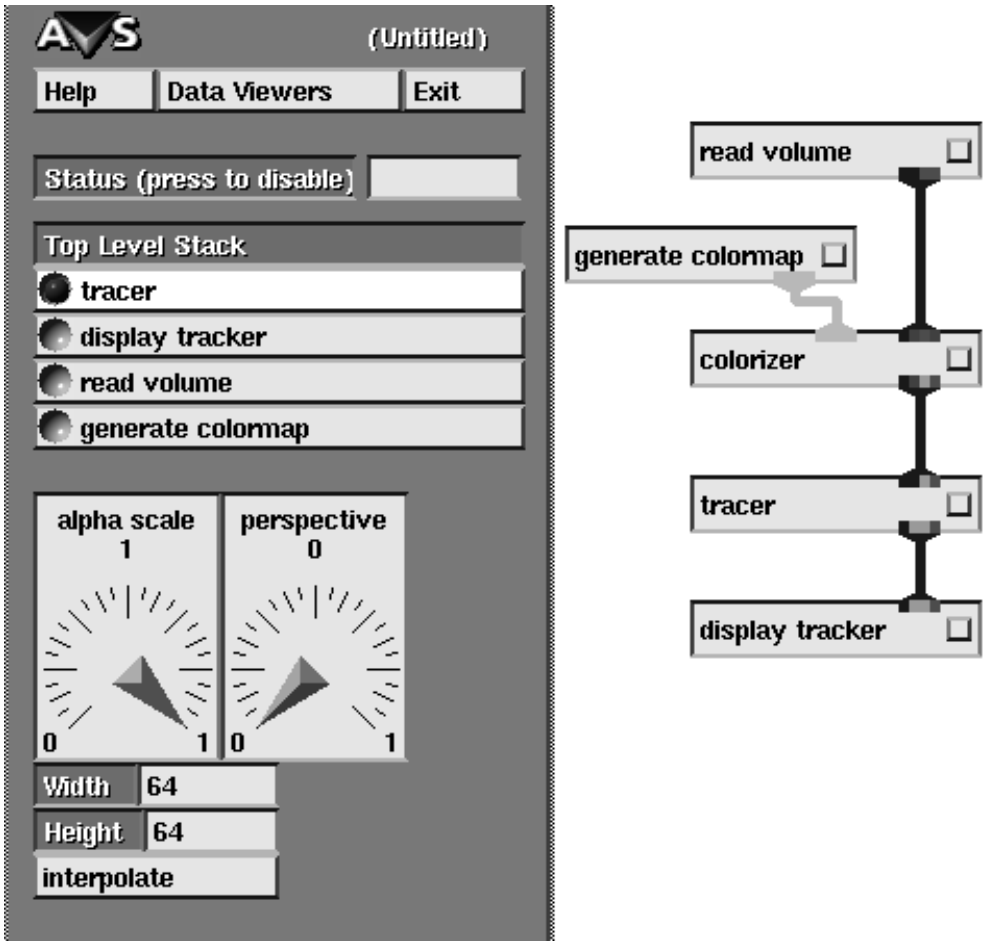


Figure 1-4 Module's Interface: Control Panel

the port accepts (Table 1-1.). AVS checks data types as you interactively build a network, making sure that the connection between two modules is valid. When you begin to establish a module-to-module connection, AVS shows you the valid possibilities.

Input Parameters

A module's **data inputs** determine the *type* of data it processes, while its **input parameters** determine *how* the data is to be processed.

The following examples use the modules shown in Figure 1-2 to illustrate several types of parameters:

- The **read volume** module brings a 3D block of byte values into a network. Its input parameter specifies the file from which the values are to be read.
- The **generate colormap** module creates and outputs a colormap that defines the map used to transform data values into color values. Its input

parameter is implemented as an interactive "colormap editor", with which you specify the 256-entry colormap.

- The **tracer** module reduces a 3D block of partially-transparent color values to a 2D image using a ray tracing algorithm. Its input parameters control the size of the output image, the opacity, the interpolation method used, and the global perspective.
- The **image viewer** module is the AVS Image Viewer. It displays the output image. It includes facilities for composing scenes of multiple images, of saving images to disk, for performing image processing techniques upon the image output of other modules, and for creating flipbook animations.

Parameters are the "control knobs" for a module. By "adjusting the knobs", you can control the way in which a module processes its data—change the angle of a cross-section plane or a rotation, change a coloring scheme, change the way values are sampled from a large data set, enlarge an image to examine some detail, etc.

Each of a module's parameters is represented by an onscreen *control widget*. Figure 1-5. Figure 1-6, and Figure 1-7 present examples of control widgets.

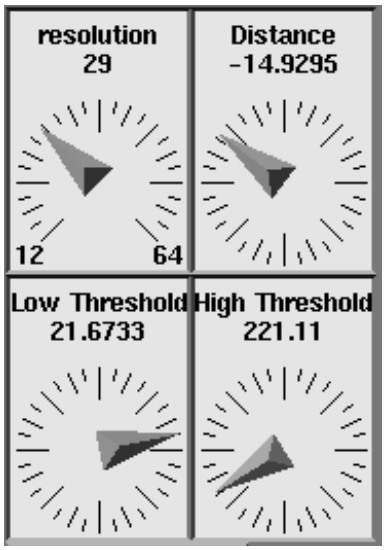


Figure 1-5 Module Control Widgets: Dial Widgets

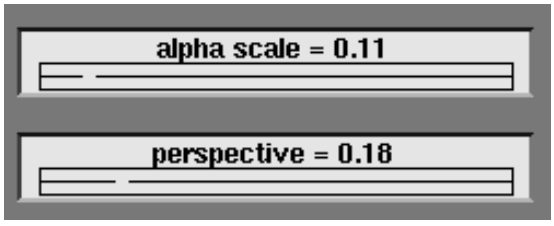


Figure 1-6 Module Control Widgets: Slider Widgwts

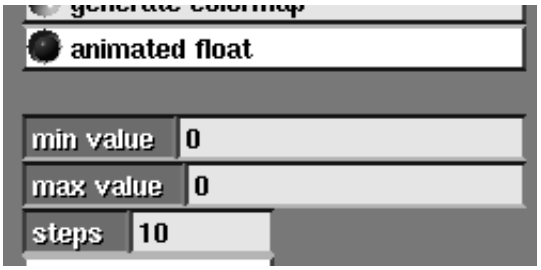


Figure 1-7 Module Control Widgets: Typein Widgets

AVS includes the following types of control widgets:

- **Dials** and **sliders** can be used to indicate integers or floating point values.
- **Typeins** allow you to specify a character string: title, label, filename, etc. Typeins can also be used to specify numeric values: integers or floating-point numbers.
- **Toggles** implement on/off switches for various parameters.
- **Radio buttons** (also called **choices**) implement sets of mutually exclusive choices.
- **File browsers** allow you to specify a file to be read or written.

AVS also provides a set of Data Input modules that will produce each of the standard parameter data types (integer, floating point, string, etc.) and send them to other modules' input parameter ports.

Data Outputs

Data outputs for modules are analogous to data inputs. Each data output is represented on the module icon by a color-coded output port, along the bottom edge of the icon. The color-coding is the same as for input ports.

Subroutine Modules and Coroutine Modules

There are two types of AVS modules, which differ in the way they fit into networks. A short explanation follows; for a more complete discussion, see the *AVS Developer's Guide*.

- *Subroutine* modules are essentially passive, like subroutines in a standard program. When you execute a network, each subroutine module initializes itself (a system process is created). But the module does not perform any work (the process *sleeps*) until the AVS Flow Executive signals it. In addition to "waking up" the module, the Flow Executive passes its input data to it. When the module finishes computing, it passes the output data back to the Flow Executive, then returns to its dormant state. Execution is synchronous; one module is active at a time.
- *Coroutine* modules are active, not passive. Rather than being like a subroutine, a coroutine is an autonomous, cooperative process that can continually execute, passing data to the Flow Executive on its own initiative, instead of doing so only when it is signalled. Coroutine modules typically

implement computational simulations, such as repeatedly releasing particles to flow through a vector field.

Standard Modules and Module Libraries

The AVS product includes a large number of general-purpose modules. This means that, often without any programming, you can begin to visualize your data sets.

The modules can be grouped into *module libraries*, each of which contains a set of modules designed to be used together. During an AVS session, you can switch back and forth among module libraries easily. You can also rearrange the libraries or create new ones, using an interactive module library editing facility.

User-Written Modules

One of the most important aspects of the AVS system is its extensibility. Many installations have already developed computer programs to process the raw data. AVS makes it easy to turn such user-supplied programs into AVS modules. Once this is accomplished, the user-written module can be combined with any other modules—AVS-supplied or user-written—to implement visualization applications.

The AVS **Module Generator** is an interactive interface that will generate skeletal module source code in C or FORTRAN for subroutine or coroutine modules. Based upon user input to a series of menu panels, it automatically creates the AVS library calls that will produce the interface to the module. It uses "reserved areas" to show where user-supplied code is required. The module writer can use the Module Generator to create modules, automatically produce Makefiles, compile and debug module code. The Module Generator is described in the *AVS Applications Guide.*

AVS Networks

Modules are the computational units in AVS. **Networks** are used to configure modules together into a visualization application. To view in a particular manner, you select the modules that perform the appropriate computations and combine them into a network. You can save the network on disk, then repeatedly use it to visualize the same data, or any other data set of the same form. After using AVS for some time, you will most likely maintain a group of networks that, collectively, satisfy most of your visualization needs.

You create networks using the AVS Network Editor subsystem. The mouse-driven interface allows you to interactively construct network diagrams, like those illustrated above. To select a module, you drag its icon from a *Palette* into a *Workspace* (Figure 1-8). To make and break connections between modules, you click-and-drag the mouse.

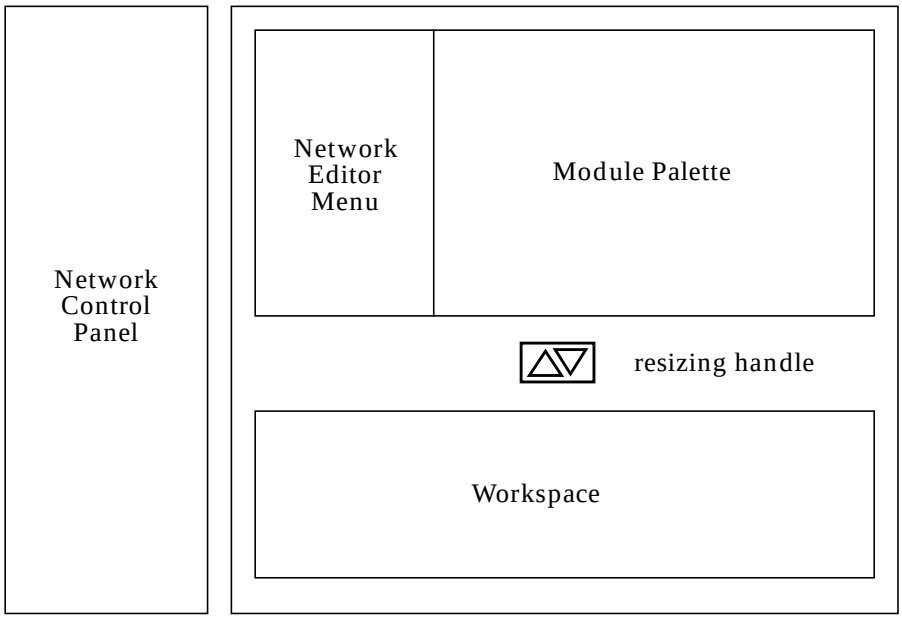


Figure 1-8 AVS Network Editor Windows

At any time, you can save a network in a disk file, for later retrieval. Only the network structure and the current settings of the input parameters are saved—the data to be visualized is not part of the network, but is loaded when the network executes.

Data Flow in an AVS Network

Figure 1-9 repeats Figure 1-2. This time, the figure emphasizes the top-to-bottom way data flows through a typical AVS network.

The data-flow diagram reflects the scientific visualization process, which begins with data and ends with onscreen images. Networks that use data stored on disk begin with a "read data" module. (There are several such modules, to accommodate the variety of AVS data types.) These modules allow you to specify the name of a file containing the raw data. By selecting different files, you can use the same network to visualize different data sets.

The network illustrated above has a simple structure and performs a (relatively) simple task—reading a single data set and constructing a single image. More complex networks can use multiple data sets, creating independent images or composite images. A network can consist of any number of independent sub-networks. Figure 1-10 illustrates a more sophisticated network.

Modules in a network can execute in parallel when there are multiple processors available (including remote modules on remote hosts), the modules have no input dependencies, are in different processes, and the user explicitly enables this feature with the *-parallel* command line option.

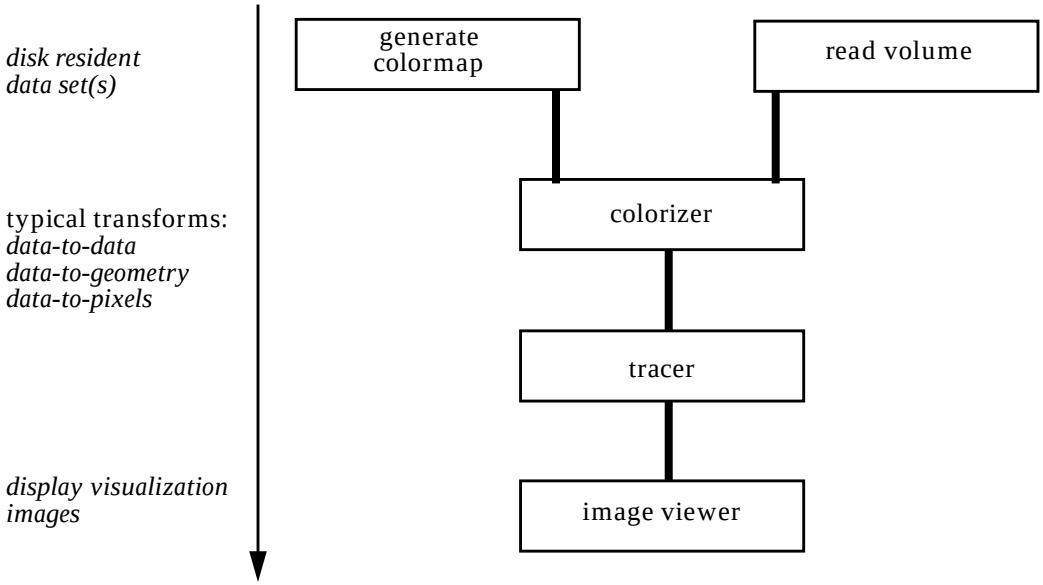


Figure 1-9 Data Flow in a Network

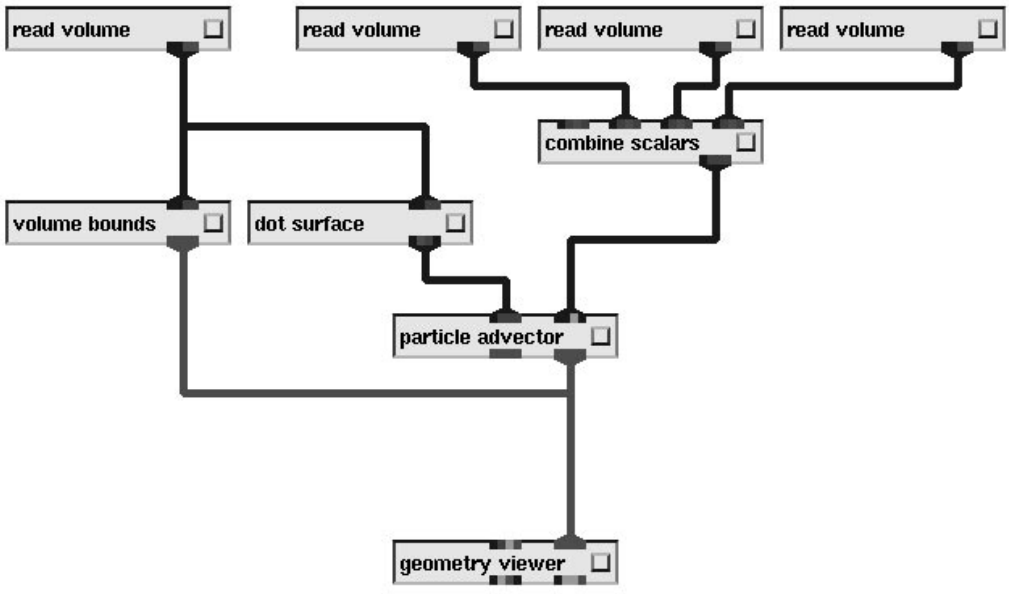


Figure 1-10 Complex Network Structure

Where a certain set of modules and connections are being used repeatedly, modules can be grouped together to form *macro* modules. A macro module appears in a network as a single module icon that the user interacts with in

the same manner as an individual module. Macro modules can be nested within other macro modules to form hierarchies of arbitrary depth.

Networks may contain only one kind of "cycle": a module's output data can subsequently be fed back into the immediately previous module as input, using the *upstream data* mechanism. The two modules must agree on the data structure they are communicating with. Upstream data ports and connections are usually invisible.

Network Control Panel

A network's data-flow diagram does not show one very important aspect of network execution: the settings of the module's input parameters. As you construct a network, the control widgets that represent the parameters (and allow you to control their values) are automatically assembled in the *Network Control Panel* window along the left edge of the screen. See Figure 1-4.

By default, the control widgets are collected into pages, one page for each module. You can redesign the layout of control widgets, however, to create simpler and more convenient user interfaces to your networks. This allows developers of networks to "package" their work so that even the most sophisticated visualization tasks can be performed easily and reliably by users. This facility is described in the "Layout Editor" section of the "Advanced Network Editor" chapter.

You can also extend the Network Control Panel to include additional physical input devices such as a dialbox and a spaceball. Certain types of input parameters can be associated with a dialbox or the spaceball, instead of with an on-screen control widget.

AVS Display Windows

AVS creates its visualization images in *display windows* on the screen. (There is also a provision for saving black and white and color versions of images in PostScript files for printing, storage, or transfer to another site.) Each display window is an X Window System window. This integration of AVS with X means that you can move, resize, iconify, and otherwise manipulate display windows using an X window manager. AVS also provides some window-oriented functions, such as *zoom* and *unzoom* for pixel-based windows. You can integrate display windows into the control panels of the visualization networks you build, creating predictable and space-efficient user interfaces.

An Example

The following series of figures illustrate using the Network Editor's visual programming interface to construct a series of visualization networks.

All of the networks use the same sample dataset, available online with the AVS release in the file */usr/avs/data/field/hydrogen.fld*. *hydrogen.fld* is a 64x64x64 uniform field, where each data value is a byte quantity from 0 to 255 that represents the probability of an electron occurring around the nucleus of a hydrogen atom.

Note: This same dataset also exists as a **volume**-format file in */usr/avs/data/volume/hydrogen.dat*. We use its field representation because it has more general applicability as an example.

With this single dataset, we illustrate:

- Three geometry-based visualization techniques. The mapper modules involved (**isosurface**, **arbitrary slicer**, and **volume bounds**) produce 3D geometry output representations of the input data. These geometries are assembled together and viewed through the AVS Geometry Viewer in its **geometry viewer** module form.
- Two image-based visualization techniques. The mapper modules (**orthogonal slicer** and **tracer**) produce 2D colorized image output representations of the input data. These images are assembled together and viewed through the AVS Image Viewer in its **image viewer** module form.
- Two graph-based visualization techniques. In these techniques, the output of the **orthogonal slicer** and **generate histogram** modules are viewed as contour and linear plots with the AVS Graph Viewer in its **graph viewer** module form.

There are over 100 visualization modules supplied with AVS. The samples given below show only the most basic network structures applied to the simplest of input datasets. For more examples of visualization networks:

- See the *AVS Module Reference Manual*. Each module has one or more example networks that show how it is used in conjunction with other modules.
- The **Demo** suite under the **Applications** menu presents a selection of demonstraton scripts. These scripts automatically construct illustrative networks, then run the networks with a variety of parameter settings. When the network completes, it stays in the Network Editor where you can manipulate the modules' parameter widgets yourself and watch their effects.

Mapper Modules: Geometries

Figure 1-11 is a simple network. In this network, the **read field** module reads the */usr/avs/data/field/hydrogen.fld* dataset into the network.

arbitrary slicer receives this data as input and creates a 2D slice plane through the 64x64x64 volume of data.

The slice plane would be a featureless gray plane, except that **arbitrary slicer** uses the colormap it receives from the **generate colormap** module to paint the

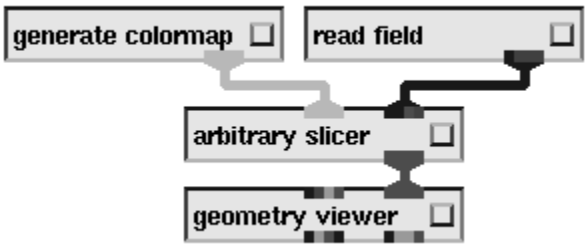


Figure 1-11 One-Mapper Geometry Network

numeric values intersected by the 2D slice plane different colors. In the default colormap, small values map to various shades of blue, mid-range values to greens, yellows, and oranges; while larger values map to reds. (The reds and blues appear nearly the same shade of gray in this black and white representation.) The color image would show the two circular medium-gray areas as red, indicating a high probability of an electron occurrence; the peripheral dark gray areas as blue indicating low probability, while the areas in between have fine distinctions of intermediate colors indicating intermediate probability of an electron.

The mapping between numbers and colors is entirely arbitrary. You control the mapping with the **generate colormap** module's Colormap Editor widget, as discussed in the "Network Editor" chapter.

arbitrary slicer sends the geometric slice plane to the Geometry Viewer represented by the **geometry viewer** module for display and manipulation. This produces the output window. The user has first used the Geometry Viewer's **Normalize** button to center the slice plane in the view window, then used the middle mouse button to rotate the slice plane in space.

Compositing Mapping Techniques

Figure 1-13 shows a slightly more complex network. The network is identical to the previous one, except that two additional geometry mapper modules, **isosurface** and **volume bounds**, have been added in parallel to the existing **arbitrary slicer** module. They receive the same hydrogen dataset as input from **read field**.

isosurface creates a 3D contour through a volume of data. You supply its **level** parameter widget with a numeric value and **isosurface** produces a geometric surface that passes through all of the data values in the volume that equal the level value.

volume bounds is a utility mapper module that simply creates a geometry object that is a box around the dataset's limits or *extents* in space. It shows where the data ends, how it is shaped, and permits you to orient yourself in space. *hydrogen.fld* is a uniform, cubical field, hence the cubical volume bounds. Curvilinear data produce much more interesting volume bounds.

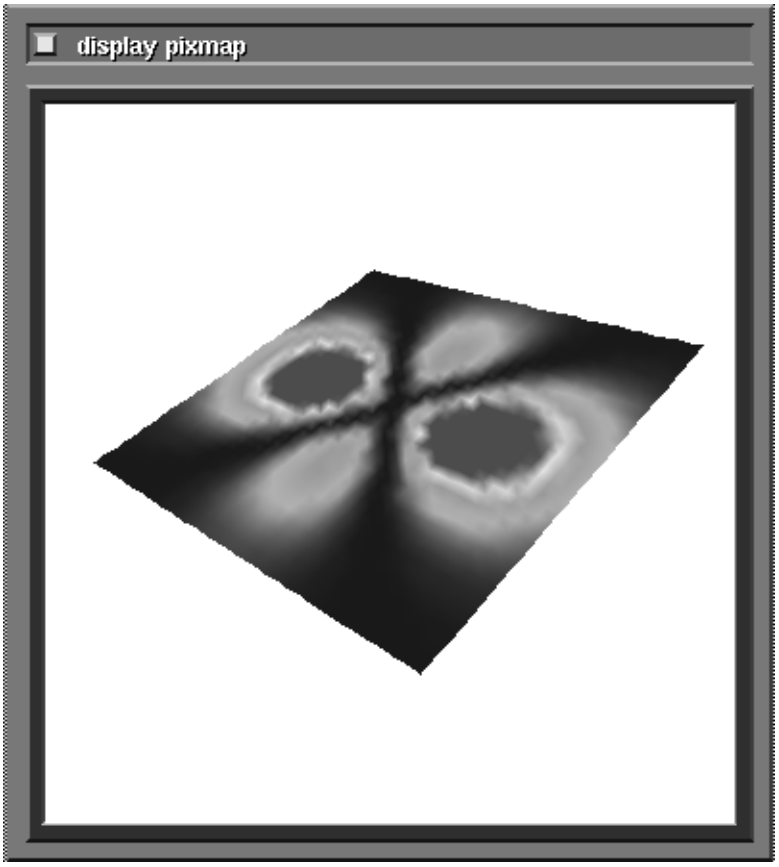


Figure 1-12 Arbitrary Slice Plane Viewed with the Geometry Viewer

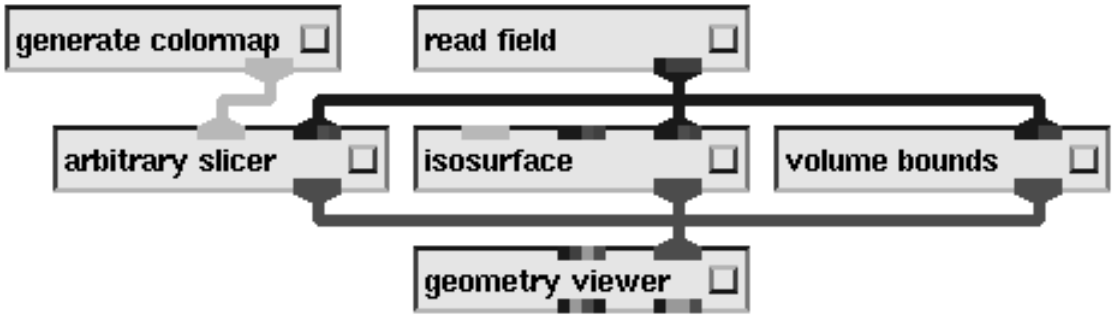


Figure 1-13 Three Geometry Mappers Modules in Parallel

All three of the geometries output by the mappers enter the same **geometry viewer** module. Hence, they are composited together into one Geometry Viewer scene window (Figure 1-14). The **isosurface** shows vividly the nature of the distribution of data values throughout the volume of the dataset. One could have surmised this by moving the **arbitrary slicer**'s plane around the volume and piecing together an impression of the "doughnut and spheres" structure out of its 2D portrayals, but **isosurface** captures the structure more graphically.

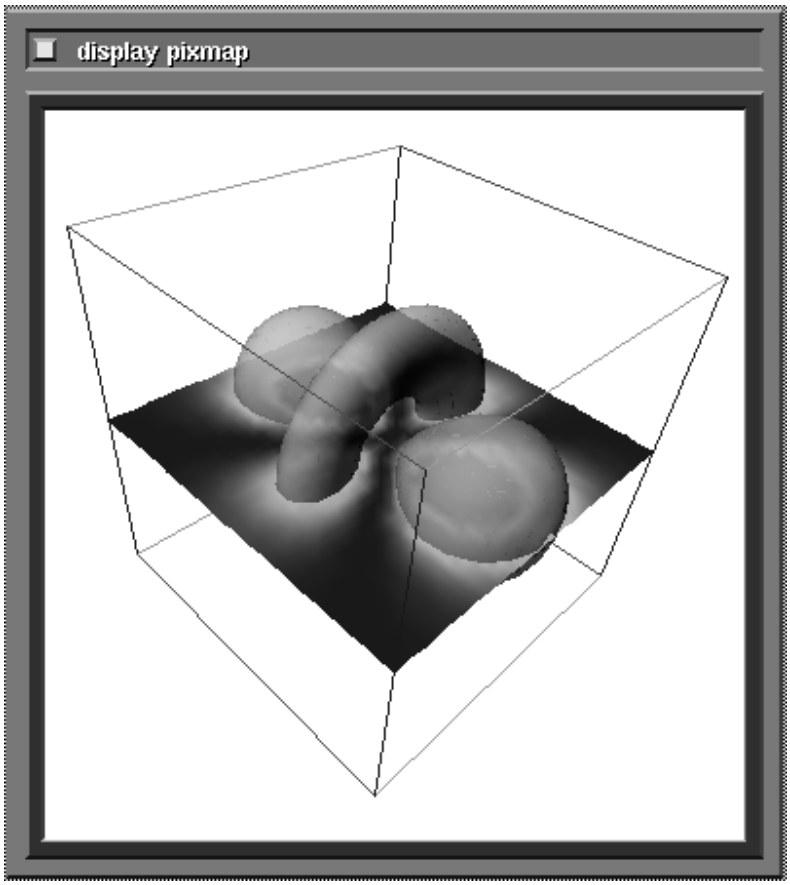


Figure 1-14 Three Geometry Objects in One View

An **isosurface** is usually opaque. Here the user has used the Geometry Viewer's transparency controls under its **Edit Property** button to make the surface semi-transparent, letting the colors of the **arbitrary slicer**'s slice plane show through.

Mapper Modules that Produce Images

We now construct networks using mapper modules that produce 2D images to represent data rather than geometries. Figure 1-15 shows a simple imaging techniques network. As with the geometry network, the first module **read field** inputs the data to the network.

The **orthogonal slicer** module also produces a 2D slice through a 3D volume. It differs from **arbitrary slicer** above in two regards. First, the slice plane must be orthogonal to the X, Y, or Z axis instead of arbitrarily placed within the volume. Second (and more significant) **orthogonal slice** takes a 3D field volume of data and outputs another *2D field*. **arbitrary slicer** outputs a geometry which can have only one destination—the **geometry viewer** module. By outputting another field, **orthogonal slicer**'s output can be sent to other data fil-

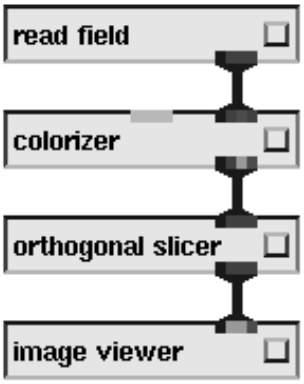


Figure 1-15 Simple Imaging Technique Network

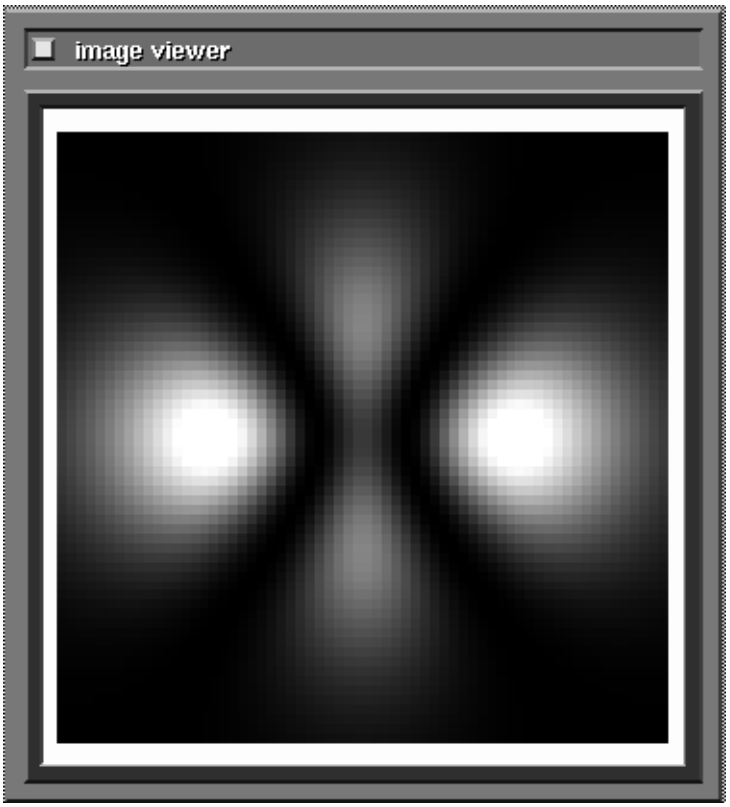


Figure 1-16 Orthogonal Slice Seen with the Image Viewer

tering and mapper modules for further processing. It is a flexible data subsetting tool.

A close look reveals that **orthogonal slicer** has no colormap input port as **arbitrary slicer** has. To gain the same effect, the **colorizer** module is interposed to color the data values according to a colormap. **colorizer** effectively changes a field of numeric probability data into a field of color values, which **orthogonal slicer** subsets.

The subsequent figure (Figure 1-16) shows the 2D output displayed in the Image Viewer (**image viewer** module). The orthogonal slice plane is the middle Z plane of the volume data.

Combining Imaging Techniques

Figure 1-17 shows a more complex image technique network. The new mapper module is **tracer**. It produces a ray-traced image rendering of volume data. By shooting hypothetical parallel rays of light into the volume, then coming up with a pixel value that represents how much that ray of light would have been diminished in brightness and changed in color as it passed through the volume, it creates an image that appears to be a solid, perhaps hazy and semi-transparent 3D object in space.

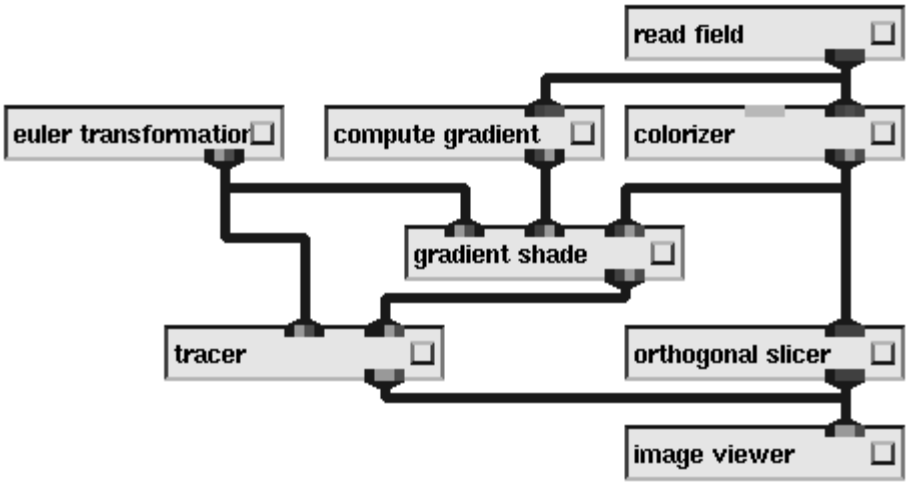


Figure 1-17 Tracing, Gradient Shading and Slicing Combined

To heighten the three-dimensional impression, the user has added the **compute gradient/gradient shade** modules to the network that pre-process the hydrogen volume data. These modules calculate how rapidly data is changing within the field (**compute gradient**), and create pseudo-shadows to indicate that change (**gradient shade**). Darker shadows indicate data changing more rapidly. When these pseudo-shadows are overlaid upon the **tracer** image, it creates an even more startling impression of a real, solid object.

The **euler transformation** module is feeding parameter data to both **tracer** and **gradient shade**. This "parameter-as-data" module allows the user to orient the **tracer** module's volume in space; it acts as a kind of "this is where the camera is" module. **gradient shade** also gets the transformation so that its pseudo-shadow overlays are at the same orientation.

Figure 1-18 shows the two images as they appear in the Image Viewer. The user has used the Image Viewer to scale, position, and label the two images.

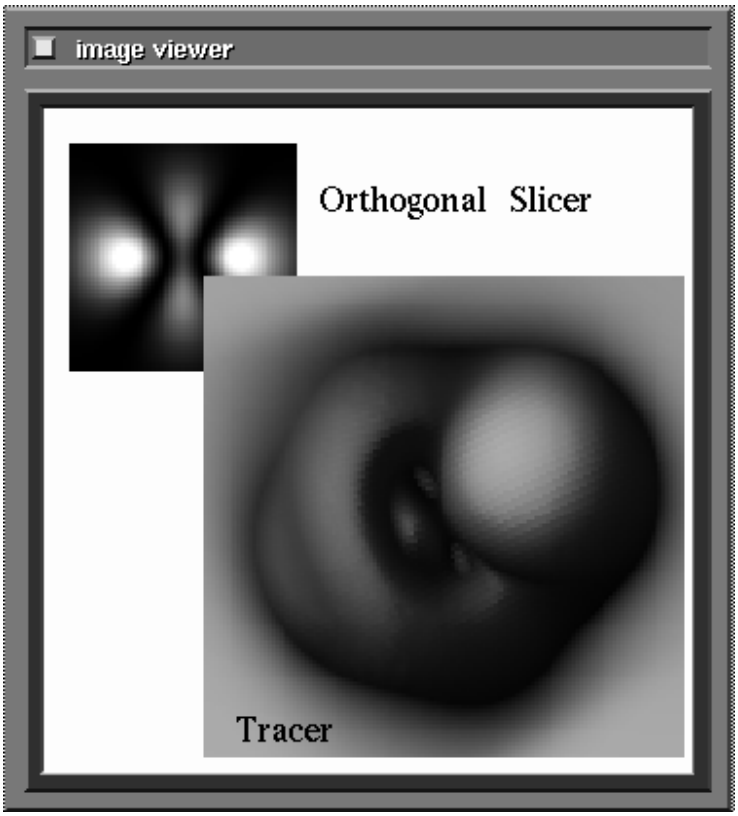


Figure 1-18 Tracer and Orthogonal Slicer Images in the Image Viewer

Producing Graphs

Figure 1-19 shows a simple network to produce a graph. **read field** again reads the hydrogen atom data. **orthogonal slicer** appears in order to section out a 2D slice of the 3D field, which it feeds to the Graph Viewer's (**graph viewer** module) center input port. This port is used to make the Graph Viewer create a contour plot of the data in the 2D slice. The contour plot that results from this network is shown in the lower left corner of Figure 1-23. The contour is, again, made from the middle Z plane of the hydrogen data.

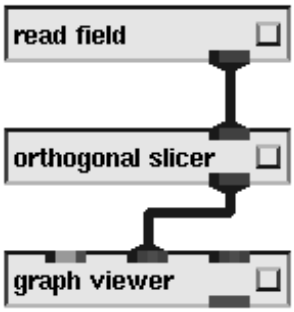


Figure 1-19 Simple Network to Produce a Graph

The next network (Figure 1-20) shows the **generate histogram** module taking the hydrogen data and producing a 1D field that represents the distribution of data within the dataset. It sends this data to a different **graph viewer** module to plot as Y values against X values, in bar plot format (Figure 1-21). You could use **generate histogram**'s minimum and maximum parameter widgets settings to plot the distribution of different subsets of the data values, effectively "zooming in" on different data ranges.

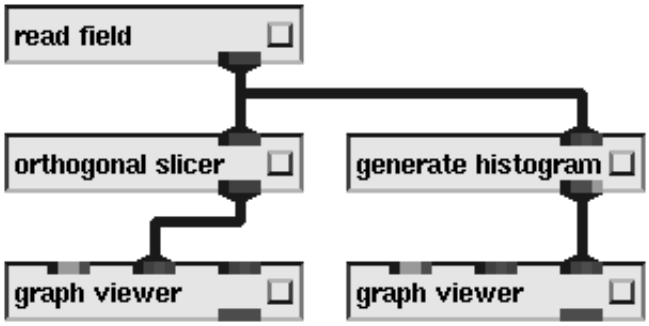


Figure 1-20 Plotting Distribution with Generate Histogram

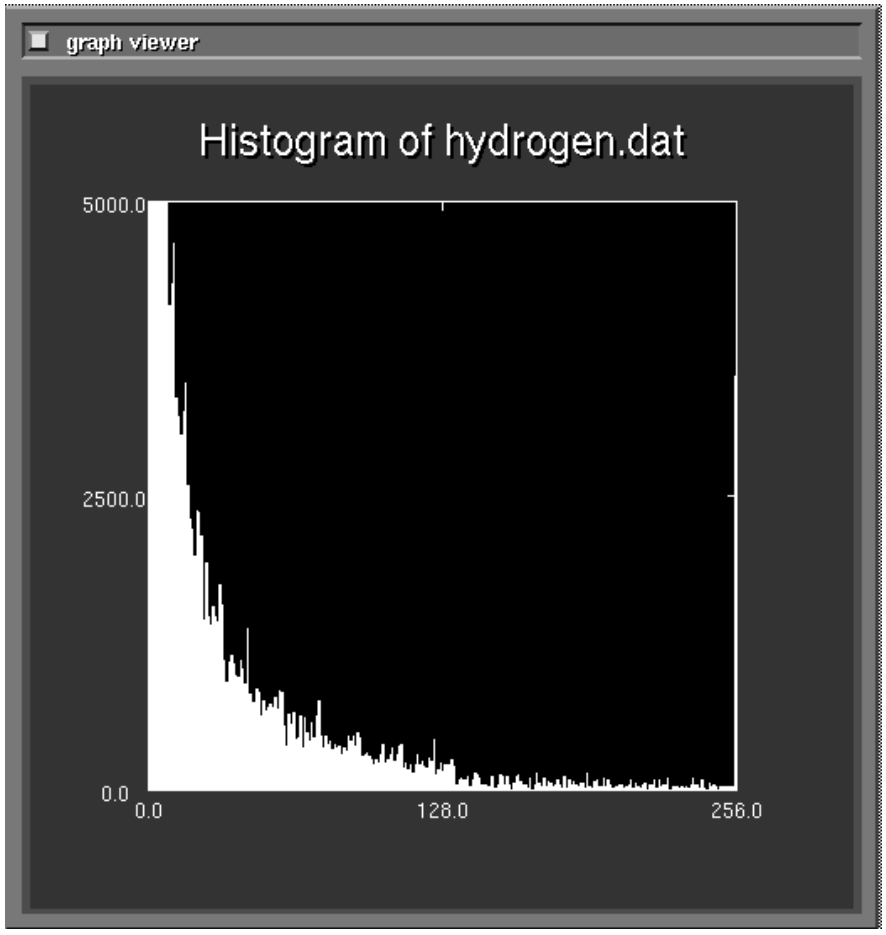


Figure 1-21 As a Bar Plot in Graph Viewer

Techniques Combined

The next two figures (Figure 1-22, Figure 1-23) show a single network performing all of the visualization techniques shown so far. The next figure shows the single, multi-branched network; and the following figure the display windows it produces together on the screen. Again, all are representations of the single 3D field dataset *hydrogen.fld*.

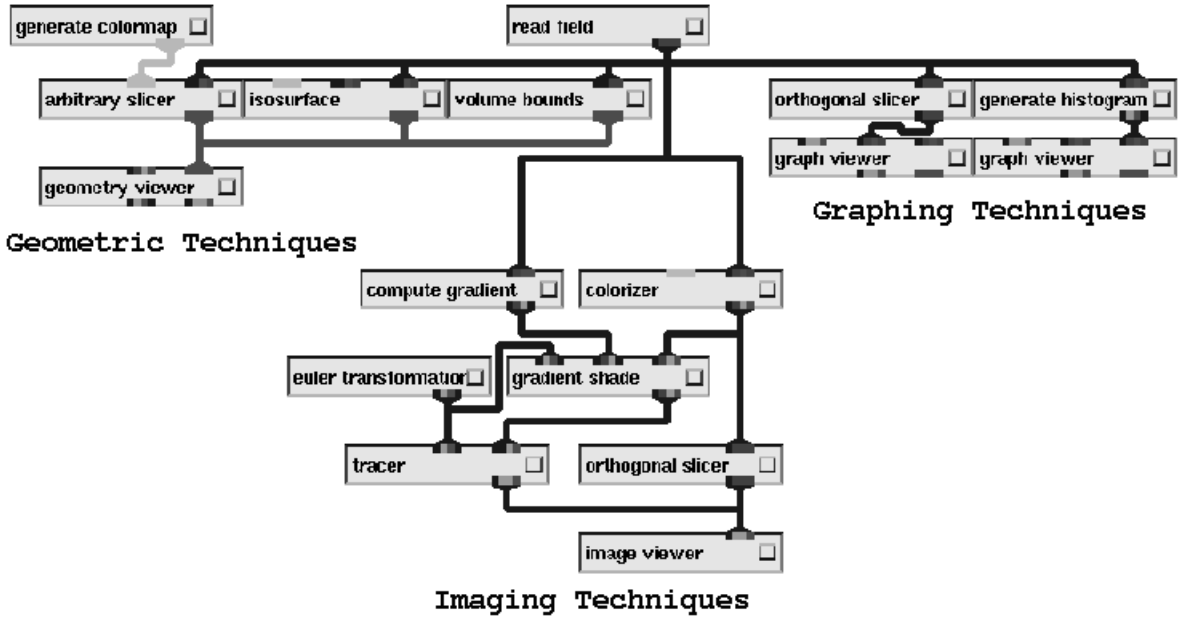


Figure 1-22 Techniques Combined in a Single Network

Curvilinear, Vector Data

AVS also supports curvilinear, floating point vector data as well as the uniform scalar byte data represented by *hydrogen.fld*. Figure 1-24 uses modules to produce geometric renderings that visualize the */usr/avs/data/field/bluntfin.fld* dataset. (This is actually two PLOT3D-format datasets in */usr/avs/data/plot3d* read in using an AVS field header.) The magnitude of the vectors (**extract vector/vector mag**) are plotted as spheres (**bubbleviz/scatter dots**) whose color reflects the size of the vector; **stream lines** are passed through the data; and the curvilinear shape of the data is reflected by **volume bounds** (Figure 1-25). The **color range** module is inserted after the **generate colormap** module in order to scale the colormap to the narrow data range (floating point values all near zero), rather than the default 0 to 256 byte value range.

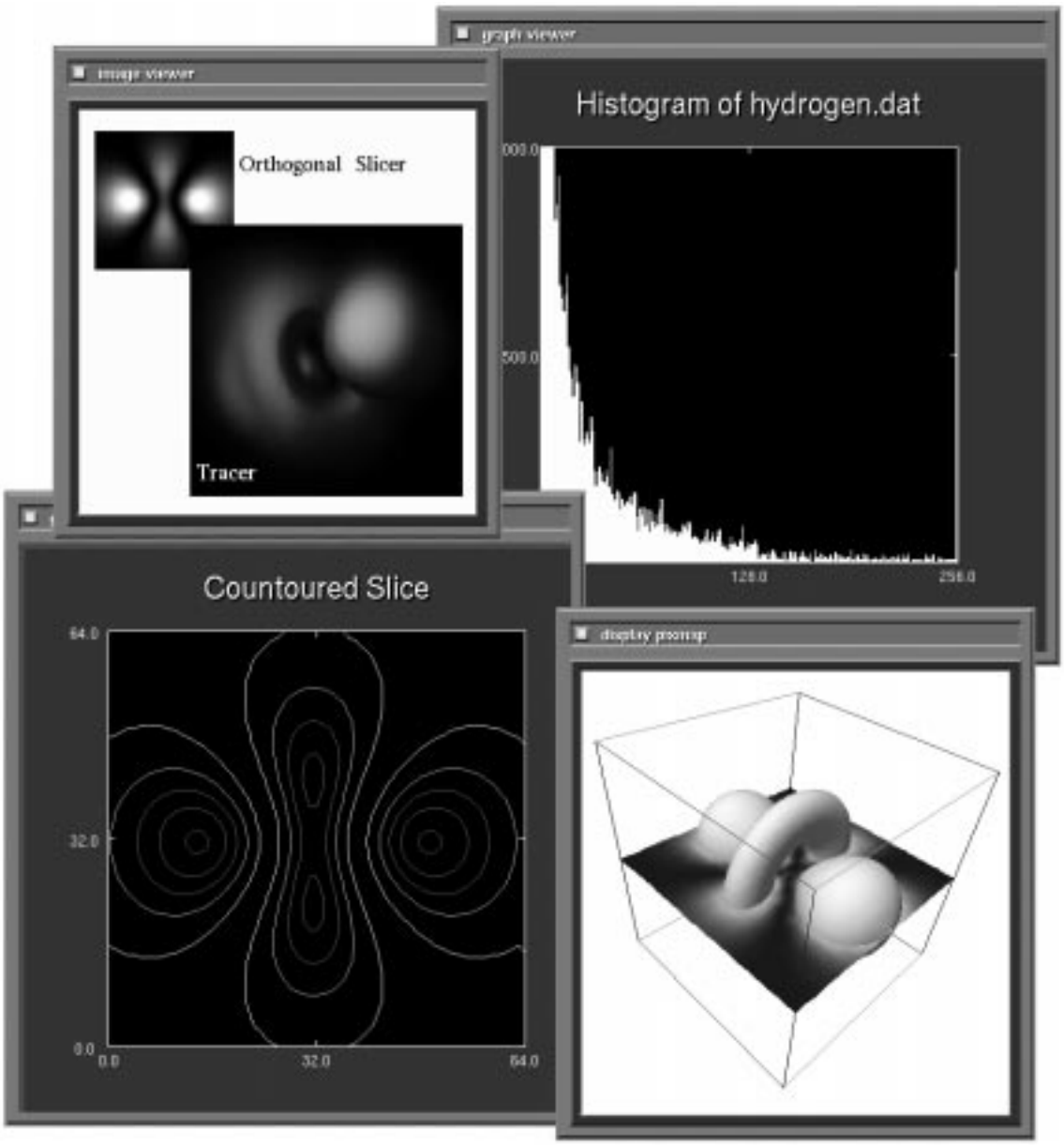


Figure 1-23 Image/Geometry/Graph Viewers Display Windows

Note on Platforms

AVS runs on a variety of vendor hardware platforms. Where the platform has a native graphics subsystem, AVS uses the graphics software library and hardware rendering capabiities present on the platform to produce its 3D renderings of objects in the Geometry Viewer.

The AVS Geometry Viewer also contains a *software renderer*. The software renderer implements its own graphics model (lighting, shading, texture mapping, clipping planes, etc.) in software, rendering the resulting Geometry Viewer scenes into an X Window System image file which is then displayed

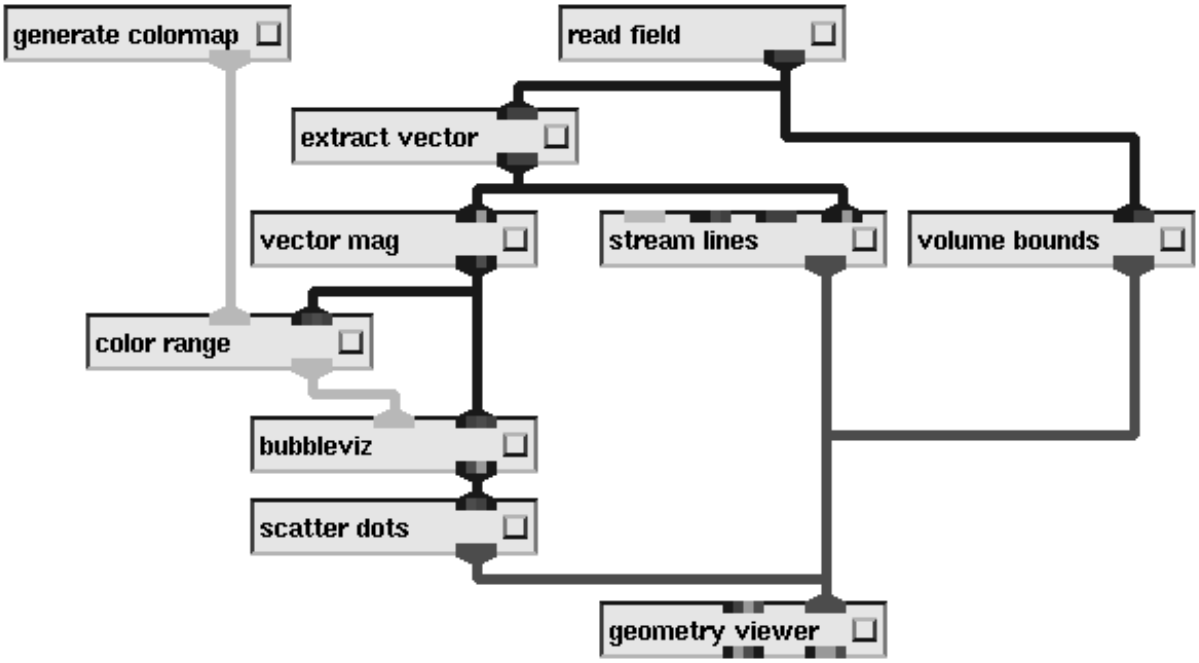


Figure 1-24 Network to Visualize Vectors

on the screen. Although generally slower than a hardware renderer, the software renderer is useful in several circumstances:

- You can use it to run AVS as a remote X Window System client on another machine in your network. Your own workstation or "X terminal" need only meet a few minimum requirements. See the "AVS on Color X Servers" appendix for more information on this feature.
- If the hardware renderer on your platform does not support a particular rendering feature such as transparency or texture-mapping, you can switch to the software renderer to obtain the rendering technique.

See the release notes that accompany AVS on your platform for specific information on rendering options.

AVS Documentation

The remainder of this *AVS User's Guide* discusses:

- Importing Data into AVS and AVS Data Types
- Starting AVS
- Image Viewer Subsystem
- Geometry Viewer Subsystem

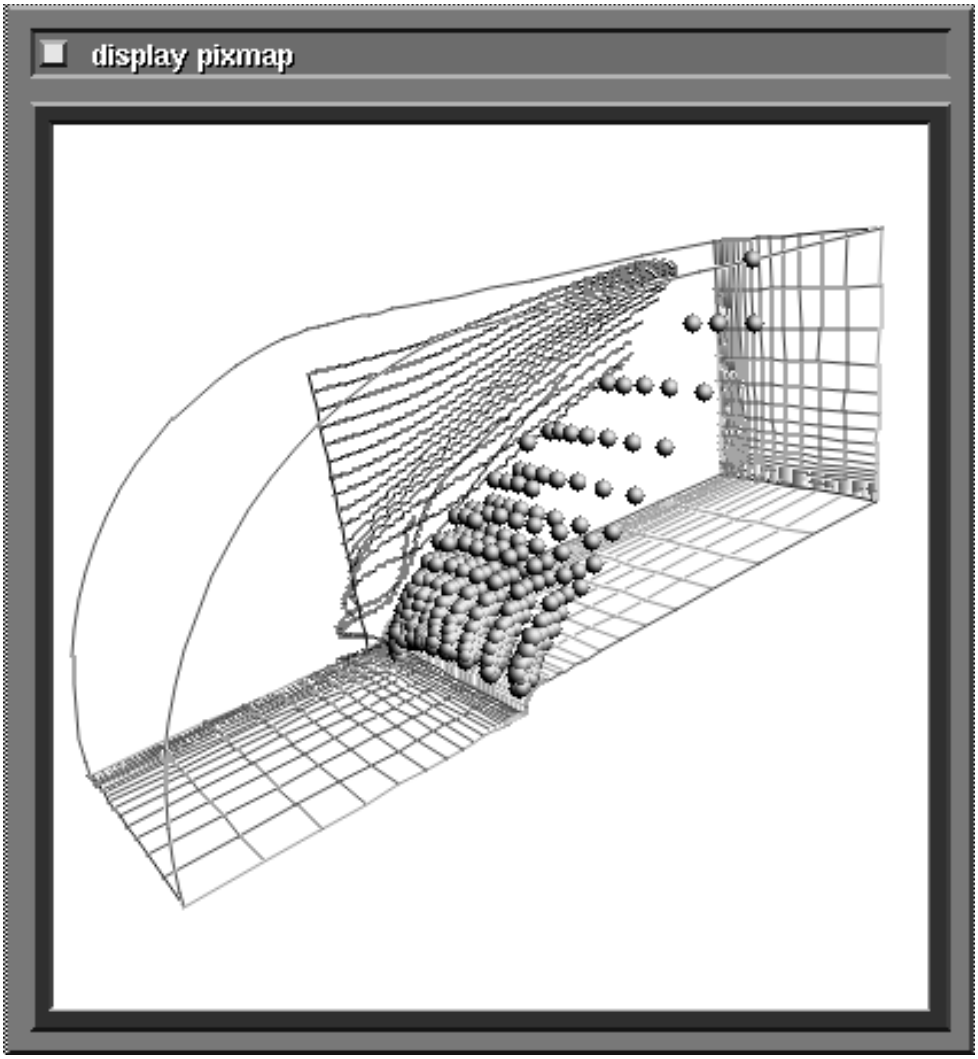


Figure 1-25 Stream Lines/Scatter Dots in the Geometry Viewer

- Network Editor Subsystem
- Graph Viewer Subsystem
- Advanced Network Editor Techniques

The appendicies cover using AVS from a color "X terminal," and some supplemental information on the Geometry Viewer's data storage script language.

The *AVS Module Reference Manual* describes each of the 100+ AVS scientific visualization modules in "man page" format. Each module's inputs, outputs, and parameters are listed together with a description of how the module works and when you would use it. Most show one or more sample networks that you can construct for yourself with the Network Editor. Each of these module man pages is also available online through the Network Editor.

The *AVS Tutorial Guide* gives a step-by-step introduction to using three AVS subsystems: the Demo suite, the Geometry Viewer, and the Graph Viewer.

The *AVS Developer's Guide* describes how to write an AVS module. Its "AVS Data Types" chapter and various appendicies contain the most detailed descriptions of the actual internal format of AVS data structures, and the programming libraries that manipulate them. The *Developer's Guide* also describes the Command Language Interpreter.

The *AVS Applications Guide* describes how to use three AVS facilities: the **Module Generator,** an interface that generates skeletal AVS module code in C and FORTRAN; the **AVS Data Interchange Application** (ADIA), a facility for importing external-format data files into AVS field file format; and the **Data Viewer**, a facility for rapidly constructing networks using a pulldown menu interface.

Interactive help is available, including sample CLI scripts that illustrate network and module use. See the "Using On-line Help" section of the "Starting AVS" chapter.

CHAPTER 2

IMPORTING DATA INTO AVS

Introduction

The process of scientific visualization is, in very simple terms, one of transforming data into pictures. AVS contains a comprehensive set of modules for transforming data and for generating and manipulating 2D images and 3D geometric scenes. These modules and viewers are discussed in detail in the remainder of this *User's Guide* and in the *AVS Module Reference Manual*. This chapter presents various techniques for importing data into AVS.

Unfortunately, scientific data is computed or acquired in a plethora of different formats. Perhaps at some point in the future a standard format will be adopted for representing and storing scientific data, but today, no such standard exists. In order to use AVS effectively with *your data*, you will have to spend some time translating the data into a format that AVS can understand.

This chapter is designed to help you achieve this goal. The first section, "AVS Data Types," discusses the basic types of data used by AVS, and will guide you in selecting a data type which closely matches your own data. The next section, "Data Import Strategies," presents a series of choices available to you for importing your data into AVS. The last five sections, "Field Data," "Geometric Data," "Unstructured Cell Data," "Colormap Data," and "Molecule Data Type" present a description of each major AVS data type, AVS modules for reading these data types, and external file formats for storing AVS-compatible data sets.

Once you have imported your data into AVS, the power of interactive scientific visualization will be at your fingertips—allowing you to experience the same benefits of insight and discovery that thousands of AVS users have experienced.

AVS Data Types

This section briefly describes each of the basic AVS data types and gives a few examples of how each data type is used. To help guide you in selecting which data type is appropriate for your application, a quick reference table is presented at the end of this section indicating the appropriate data type for various scientific and engineering applications.

AVS has two main types of data: **primitive data** and **aggregate data**.

Primitive Data

Primitive data includes simple scalar types (byte, integer, single-precision floating-point, and double-precision floating point) and text strings. Primitive data types are the basic building blocks of aggregate data types, and are also used by AVS to represent parameters.

byte

A single byte, or 8 bits. On all machines, a byte is used to represent an unsigned integer in the range 0..255.

Bytes are typically used by aggregate data types to represent pixel or voxel intensities. If the intensities are discrete and within the ranges specified above, byte data significantly reduces the amount of memory required to store a scientific data set when compared to integers or floating-point values.

integer

A single machine word. On 32-bit machine architectures, integers are four bytes and can be used to represent signed integer values in the range -2**31 to 2**31 (-2147483648..2147483647). On 64-bit architectures, these ranges are much larger.

Integers are used by aggregate data when discrete values are appropriate, but greater than the range afforded by bytes (0..255). For example, integers can be used to represent voxel intensities. Integers can also be used as parameters. Examples include data replication (zoom=2), downsizing (downsize=4), or an 2D orthogonal slice plane setting within a 3D volume of data (k=21).

single-precision floating point

A single machine word used to represent a floating-point quantity in *single* format. On 32-bit machine architectures, single-precision floating point values are four bytes.

Single-precision floating-point data values are used in aggregate data to represent continuously varying phenomena, e.g. temperature or velocity. Single-precision floating-point parameters are used for continuously varying values. Examples include the diffuse component of a gradient shader, or the x value of a bilinear interpolator.

double-precision floating point

Two machine words used to represent a floating-point quantity in *double* format. On 32-bit machine architectures, double-precision floating point values are two words, or eight bytes.

Double-precision floating-point data is used in cases where the accuracy of single-precision is not great enough to contain the significant components of the input data values.

text strings

An array of bytes, each of which represents a single character.

Text string parameters are typically used to represent file names. Aggregate data uses text strings for labels on elements of data; for example, in a multi-dimensional fluid flow data set, a text string might be used as a label for each of the vector components: density, x-momentum, y-momentum, etc.

Aggregate Data

Aggregate data is used by AVS to represent the major data types for scientific and engineering data visualization, including field data, geometric data, unstructured cell data, and colormap data. AVS has other aggregate data types, including pixel maps, and user-defined data. These two data types are generally not required by a novice or intermediate-level AVS user to import data into AVS, and are therefore not described in this chapter. For a full description of AVS data types, consult the *AVS Developer's Guide*.

Field Data

Field data is an *n*-dimensional array with an *m*-dimensional vector of values at each array location (where *n* and *m* are any integers). The physical location of each data element (if it exists) is either implied or specified explicitly with 3D coordinates.

Field data is typically used to represent two-dimensional images (e.g. satellite images or x-rays) and three-dimensional volumes (e.g. MRI scans, CAT scans, numerical simulation data, fluid flow data). It can also be used to represent one-dimensional data (e.g. x-y plots, or a series of points in space).

Geometric Data

Geometric data consists of three-dimensional objects that are constructed out of one or more of the following primitives: polyhedra, polygons, meshes, spheres, and polytriangles.

Geometric data is used to represent any three-dimensional shape, including real-world objects (e.g. rotor blades in jet turbines, a brake assembly in an automobile, or a film advance mechanism in a camera), objects represented as geometries (e.g. ball and stick models of molecules), and objects created by visualization techniques (e.g. an arbitrary slice plane through a volume, or an isosurface).

Unstructured Cell Data

Unstructured cell data (UCD) consists of a geometric model built of individual cells. The cells, including 1D, 2D, and 3D shapes, are defined by nodes, or vertices. Data, either scalar or vector, can be assigned to the entire model, each individual cell, and each vertex of each cell.

Unstructured cell data is used in finite element problems, including structural analysis and certain areas of computational fluid dynamics.

Molecule Data Type

The molecule data type (MDT) consists of ten defined CHEM objects that can be combined hierarchically in linked lists to represent the structure of molecules. At the root of the hierarchy is the CHEMmolecule. A CHEMmolecule is a data structure containing the molecule's name, and a unit in angstroms of bohrs. Chained off of the CHEMmolecule is an arbitrarily long list of CHEMatoms (data for component atoms), CHEMchemunits (data for defining molecule substructures), CHEMquantums (data for classical quantum chemistry), and user-defined data.

The molecule data type can be used to model chemical data for problems in classical, substructure, and quantum chemistry.

The molecule data type and the *libchem* library that manipulates it is described fully in its own document, the *AVS Chemistry Developer's Guide.* It is discussed only briefly in this *User's Guide.*

Colormap Data

A colormap is an arbitrary-sized one-dimensional array of 4D vectors. Vector components include normalized floating-point values for hue, saturation, brightness, and opacity.

Colormaps are used to specify colors of data values that are displayed on the screen. For example, an image composed of 8-bit pixels can be pseudo-colored by a colormap that ranges continuously in hue from blue to red. A pixel intensity of 0 is colored blue when it is displayed and a pixel intensity of 255 is colored red. All intermediate pixel intensities are colored with a hue that is in the color spectrum between blue and red.

AVS Data Type Reference Table

The following table presents a few common scientific applications and the AVS data types frequently used to represent various types of data. If you do

not find your application in this table, try to locate one that has similar characteristics.

Table 2-1. AVS Data Type/Application Cross Reference Table

Application	AVS Data	Examples
CFD	*field*	Simulation grids
		Finite difference data
	ucd	Finite element model and data
	geometry	Structures
MCAE	*ucd*	Finite element model and data
	geometry	Structures
	field	Finite difference data
Molecular Chemistry	*molecule data type*	Ball and Stick models
	geometry	Microscopic and x-ray imaging
	field	Scatter data
Quantum Chemistry	*molecule data type*	Molecular orbitals
	field	Molecular interactions
		Subatommic particles
Seismic	*field*	3D volumes - sampled data
		3D volumes - simulated data
	geometry	Structures
Strategic Imaging	*field*	2D images
		Digital terrain maps
		Multispectral images
	geometry	Structures
Medical Imaging	*field*	2D images - CAT / MRI / PET scans
		x-rays
		3D volumes - CAT / MRI / PET scans
	geometry	Anatomical structures
Meteorology	*field*	2D images
		3D volumes - sampled data
		3D volumes - simulated data

Data Import Strategies

There are basically two choices a user has when importing data into AVS: *(1)* use an existing AVS module, or *(2)* write a customized new module.

You must first determine if an existing AVS module can satisfy your data input requirements. To do so, you must understand each of the AVS data types, the existing modules for reading AVS data, and the format of external AVS-compatible data. The following sections of this chapter will allow you to make this assessment.

As will be shown, many "data importing" modules are provided with AVS. These modules may be already-compiled utilities, or they may exist as source code examples that you can compile.

It is possible that you will be unable to find a module suited to your data format. However, it is likely that one exists elsewhere. Contact your sales office to see if a "user-contributed" module exists for your data format—many users of AVS could have data similar to your own and might already have created a module to import it.

If an existing AVS module is not available for importing your data, you will have to write a new module, tailored to match your specific data format. One of the primary goals of AVS is to minimize the amount of programming necessary if, and when, a new module must be developed.

Once it has been determined that programming is necessary, several options are available:

1. Write a shell-level "translator" that converts your data format into the appropriate AVS data format.
2. Modify your application to produce data in AVS format.
3. Write an AVS module that converts your data format into the appropriate AVS data format.
4. Modify your application as an AVS module that can be incorporated into AVS networks.

If programming is required, the most effective use of AVS will generally involve writing a module. Once written, the module can become part of many different visualization networks, and can easily be modified and enhanced to satisfy additional requirements. AVS has example module source programs in Fortran and C for each of its internal data formats, located in the */usr/avs/examples* directory. You can use these samples as a guide to writing your own modules.

There are many additional benefits to incorporating a data-producing application into an AVS module. Once an application has been transformed into an AVS module, the user can gain interactive control over various parameters (e.g. starting conditions, boundary conditions, constraints). In effect, the process of visualization can be tightly integrated with the simulation or data acquisition process.

The remainder of this chapter is devoted to helping you determine if there is an existing solution to the problem of importing your data into AVS. The emphasis here is on the "no-programming required" approach. However, pointers are given to existing sample source for data importing modules. It is strongly suggested that the user consult the *AVS Developer's Guide* before programming AVS modules.

Field Data

A **field** is a generalization of the familiar array structure. Whereas each element of an array has a single data value (e.g. byte or integer), each element of an AVS field can have a list of data values. Thus, a field can be described as an *n*-dimensional array with an *m*-dimensional vector of values at each array location (where *n* and *m* are any integers). Moreover, the field can include coordinate data, so that each field element is mapped to a real-world location.

The AVS field datatype has the following components:

ndim
The number of computational dimensions in the field. For an image, **ndim** = 2. For a volume, **ndim** = 3.

dim1
dim2
dim3
...
The dimension size of each axis (the array bound for each dimension of the computational array). The number of **dim***x* entries is based on the value of **ndim**.

nspace
The dimensionality of the physical space that corresponds to the computational space (number of physical coordinates per field element).

veclen
The number of data values for each field element. All the data values must be of the same primitive type (e.g. **integer**), so that the collection of values is conceptually a **veclen**-dimensional vector. If **veclen**=1, the single data value is, effectively, a scalar. Thus, the term **scalar field** is often used to describe such a field.

data_type
The primitive data type of all the data values: must be one of **byte, integer, single,** or **double**.

field_type
The field type, one of **uniform, rectilinear,** or **irregular**.

min_ext and **max_ext**
The minimum and maximum coordinate value that any member data point occupies in space, for each axis in the data.

min_data and **max_data**
The minimum and maximum data value in the field.

labels
A title for each of the individual elements in a vector of values.

units
A string that describes the unit of measurement for each vector element.

field_data
The data values for the field.

coordinate_data
The computational-to-physical space mapping coordinate data values.

Other references to the AVS field format can be found in the **read field** page of the *AVS Module Reference Manual*, and in the "AVS Data Types" chapter of the *AVS Developer's Guide*.

AVS field data is classified into three basic categories, according to the specification of computational-to-physical space mapping. As mentioned above, the three types of fields are **uniform**, **rectilinear**, and **irregular**.

Uniform Fields

A uniform field has no computational-to-physical space mapping *between* data elements. The field implicitly takes its mapping from the organization of the computational array of field elements. However, AVS allows an entire uniform field to be mapped into physical space by specifying physical coordinates for the **bounds** of the field dataset. Example

Consider the following 2D integer-valued array (using a FORTRAN-style notation):

DATA(*I, J*) *I*=1,2 *J*=1,5

DATA(1,1) = 12
DATA(2,1) = 17

DATA(1,2) = 4
DATA(2,2) = 0

DATA(1,3) = 10
DATA(2,3) = -5

DATA(1,4) = 16
DATA(2,4) = 16

DATA(1,5) = 16
DATA(2,5) = 8

This array describes a 2D **computational** space, with *I* and *J* dimensions. The size of the *I* dimension is 2; the size of the *J* dimension is 5. The data is of type *integer*.

Since there is only one data value for each field element, this is said to be a **scalar field**. The following notation might be used to indicate the values of a **vector field**:

DATA(2,3) = (2.51, 1.09, 5.73)

The field is still 2-dimensional, but the data value is said to be a *3-vector*. Such a data value might be used to represent a velocity vector, or to represent a temperature-pressure-humidity measurement at each location in space.

In the absence of any additional information, there is a natural mapping between the computational space and a 2D **physical** space, the X-Y coordinate plane (see Figure 2-1.) .

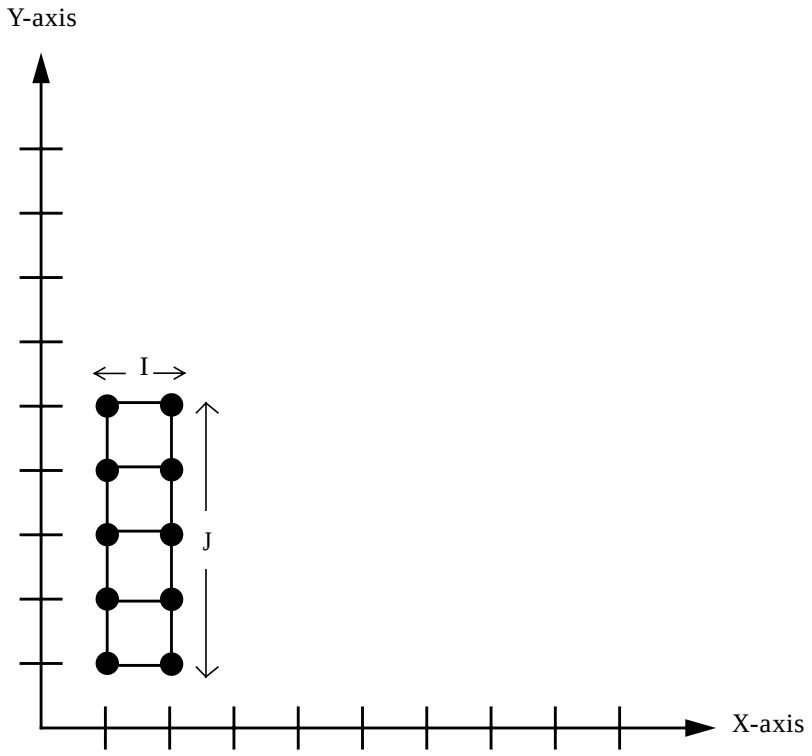


Figure 2-1 Uniform 2D Field

The physical space is a uniformly-spaced lattice. Accordingly, a field with no coordinate data for each of the data values is said to have the field type **uniform**.

Rectilinear Fields

In a rectilinear field data set, each array index in each dimension of the computational space is mapped to a physical coordinate. This produces a physical

space whose axes are orthogonal, but the spacing among the elements is not necessarily equal. Example

An explicit mapping between the computational and physical spaces can be established by specifying coordinate data for all of the points along *each axis*:

X-coordinates: 0, 3, 6, 9, 12
Y-coordinates: 20*log(1), 20*log(2), 20*log(3), 20*log(4), 20*log(5)

From the preceding example, array element DATA(1,3) would be mapped to physical location (0, 20*log(3)) according to this scheme. This mapping from computational space to the X-Y plane can be pictured as in Figure 2-2.

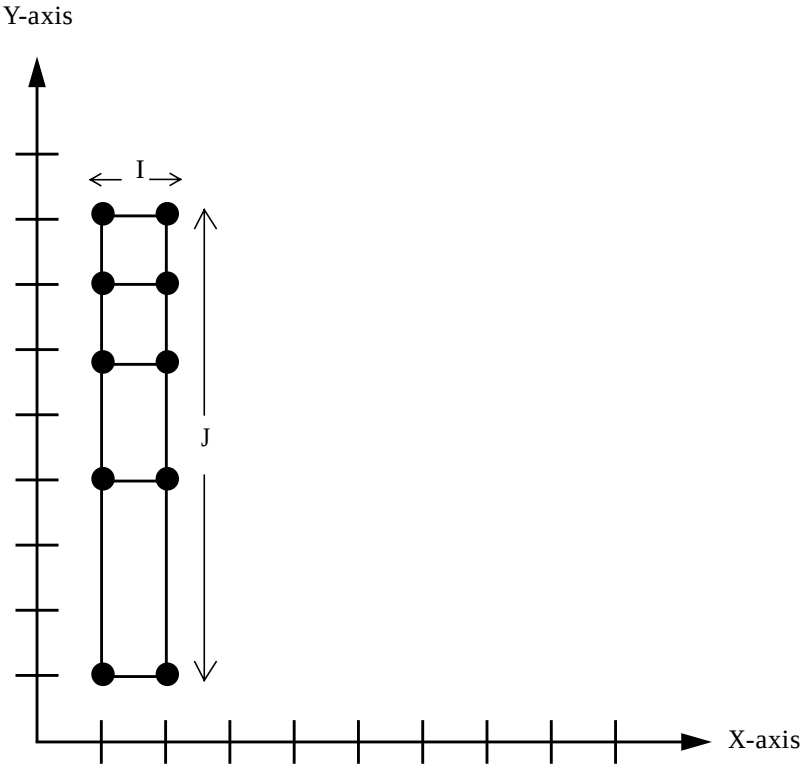


Figure 2-2 Rectilinear 2D Field

Note that in a **rectilinear** field, lines connecting the lattice points are always mutually orthogonal—all the angles are right angles.

Irregular Fields

For irregular fields, there is no restriction on the correspondence between computational space and physical space. Each element in the computational space is assigned its own physical coordinates. Example

The first example is extended once again, this time to include a coordinate pair for each of the data elements:

DATA(1,1) -> (1.0, 1.0)
DATA(2,1) -> (7.0, 1.0)

DATA(1,2) -> (3.0, 3.0)
DATA(2,2) -> (6.0, 2.5)

DATA(1,3) -> (4.0, 4.5)
DATA(2,3) -> (5.5, 4.5)

DATA(1,4) -> (3.5, 6.0)
DATA(2,4) -> (4.5, 5.5)

DATA(1,5) -> (3.5, 7.5)
DATA(2,5) -> (6.0, 8.0)

This mapping from computational space to the X-Y plane can be pictured as in Figure 2-3.

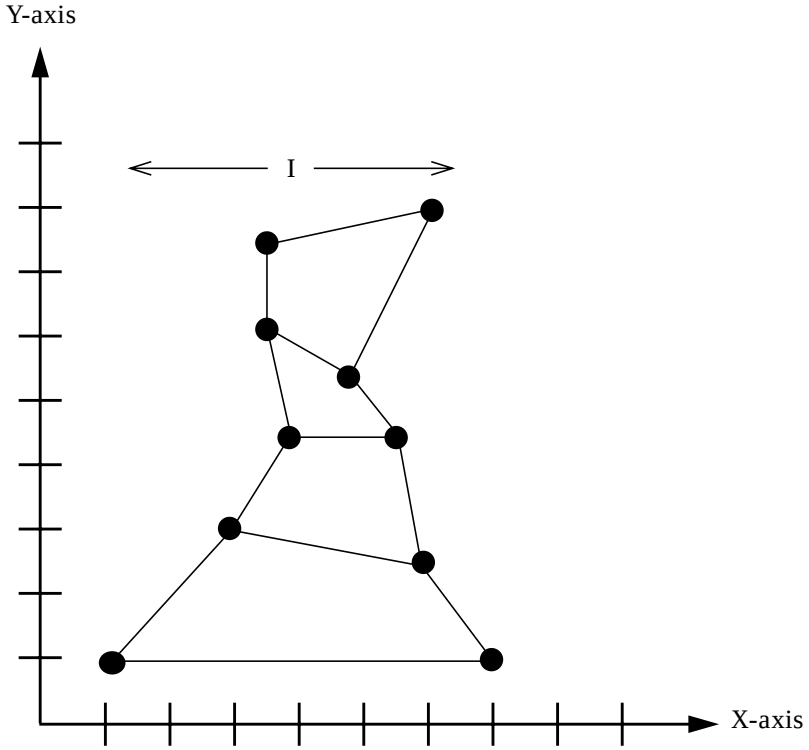


Figure 2-3 Irregular or Curvilinear 2D Field

Note that there is nothing in this scheme that restricts the physical space to having the same number of dimensions as the computational space. For example, the field element DATA(2,3) could be mapped to the physical point (4.5, 5.5, -8.1) in 3D space. This kind of mapping can be used to "wrap" a plane

(computational space) around a sphere (physical space), or to warp a flat plane into a 3D manifold.

For additional examples, see the "AVS Data Types" chapter in the *AVS Developer's Guide*.

AVS Data Interchange Application: ADIA

The **AVS Data Interchange Application** (**ADIA**) is an interactive tool for importing a wide variety of binary and ASCII data formats into an AVS field. ADIA is similar in functionality to the **read field** module described below, with these additional features:

- It can read 16-bit "halfword" data. Many medical imaging applications produce 12-bit data in two 8-byte halfwords. (The data will be represented in the AVS field as 32-bit integers.)
- It supports variables and expressions, making it possible to define skips and spaces necessary to read the input data as a function, rather than as an absolute number.
- It is an interactive facility contained within AVS—one does not need to use a text editor to externally edit an ASCII header.
- It can read data format information, such as the dimensions of a dataset, from the input dataset and use these values as part of its variables and expressions.
- It creates ASCII data forms that define how to read a particular input dataset format. These data forms can be reused and exchanged.

ADIA is not documented in this manual. Instead, it is fully described in the *AVS Applications Guide* manual.

AVS Module: read field

The **read field** module has two input modes, "native field input" and "data-parsing input". In its first input mode, it reads an AVS **field** data structure from a disk file into a network. The format of an AVS **field file** is discussed below in the section "AVS Field File Format".

In its second input mode, it converts data stored in ASCII, FORTRAN unformatted, or pure binary data files into AVS field format. **read field** can thus be used to import *many* datasets into AVS.

Native Field Input

read field can read files in the native AVS field file format into an AVS network. An AVS field file (suffix *.fld*) has the following components:

- An ASCII header that describes the field

- Two separator characters that divide the ASCII header from the data and coordinate information
- A binary area containing the data and coordinate information

The **write field** module creates files in this format.

ASCII Header

The ASCII header contains a series of text lines, each of which is either a comment or a *TOKEN=VALUE* pair. For example, the following header defines a field of type "field 2D 4-vector byte", which is the AVS image format:

```
# AVS field file
#
ndim=2          # number of computational dimensions
dim1=512
dim2=480
nspace=2        # number of physical dimensions
veclen=4
data=byte
field=uniform
```

The first two lines are comments, indicated by the **#** character. Note that the first line of the header *must* begin as follows:

```
# AVS
```

In this example, comments also occur at the end of the 3rd and 6th lines. Any characters following (and including) **#** in a header line are ignored. Comments are not required.

Separator Characters

The ASCII header must be followed by two formfeed characters (i.e. **Ctrl-L**, octal 14, decimal 12, hex 0C), in order to separate it from the binary area. This scheme allows you use the **more**(1) shell command to examine the header. When **more** stops at the formfeeds, press **q** to quit. This avoids the problem of the binary data garbling the screen.

Binary Area

The binary area contains both **data values** and **coordinate mapping** data. This section describes the components of the binary are by looking at how the size of this area is calculated.

The size of the **data values** section of the binary area is calculated by taking the product of:

- All the dimensions of the field: *(dim1 * dim2 * ... dimx)*,
- The number of data values per field element: *(veclen)*, and
- The byte size of the primitive data type: *(data_size)*.

Thus, the size (in bytes) of the data values section is:

(*dim1 * dim2 * ... dimx * veclen * data_size*)

In the block of data values:

- All the data values for a field element are stored together.
- The first array index varies most quickly (*FORTRAN-style*).

The size of the **coordinate mapping** data section of the binary area depends on the field type: uniform, rectilinear, or irregular.

- For **uniform** fields, coordinate mapping data contains two values for each physical dimension (n-space) of the data, one associated with the minimum extent of the data, and the other associated with the maximum extent.

  The minimum and maximum extent values in the coordinate binary area are copies of the **min_ext** and **max_ext** values in the field data structure, *except* when the field has been cropped, downsized, or interpolated. Then the field data structure contains the original field's **min_ext** and **max_ext** values, while the coordinate section of the binary area contains the minimum and maximum extent of the subsetted data. Mapper modules can use this additional extent information to properly locate their geometric representation of the subsetted data in world coordinate space. The extents in the coordinate binary area are stored in this order: minimum x, maximum x, minimum y, maximum y, minimum z...etc.

  The size of the coordinate mapping data section for a uniform field is simply:

  (2 * nspace) * sizeof(float)

  NOTE: *sizeof(float)* refers to the size, in bytes, of a single-precision floating-point number. On 32-bit architectures, this will be 4.
- For **rectilinear** fields, there is one coordinate for each array index in each dimension of computational space. Thus, the size of the coordinates area is:

  (*dim1 + dim2 ... + dimx) * sizeof(float)*

  All of the X-coordinates are stored together, at the beginning of the coordinates area. Following these are all the Y-coordinates, and so on.
- For **irregular** fields, each field element is mapped to a point in *nspace*-dimensional physical space. Thus, the size of the coordinates area is:

  (*dim1 * dim2 ... * dimx) * nspace * sizeof(float)*

  As with rectilinear fields, all of the X-coordinates are stored together, at the beginning of the coordinates area. Following these are all the Y-coordinates, and so on.

Example 1

The following ASCII header describes a volume (3D uniform field) with a single byte of data for each field element. This format might be used to represent CAT scan data, where all of the 2D slices are equally spaced:

```
# AVS field file
ndim=3           # number of dimensions in the field
dim1=64          # dimension of axis 1
dim2=64          # dimension of axis 2
dim3=64          # dimension of axis 3
nspace=3         # number of physical coordinates per point
veclen=1         # number of components at each point
data=byte        # data type (byte, integer, float, double)
field=uniform    # field type (uniform, rectilinear, irregular)
```

In the binary area, the data area occupies this amount of space:

(64 * 64 * 64) * 1 * 1 = 262,144 bytes

The coordinates area occupies (2 * 4) * 3 bytes. The total binary area occupies 262,168 bytes.

Example 2

The following ASCII header describes a volume (3D uniform field) whose data for each field element is a 3D vector of single-precision values. This format might be used to represent a volume of data from a sampling device where the samples are not equally spaced.

```
# AVS field file
ndim=3           # number of dimensions in the field
dim1=27          # dimension of axis 1
dim2=25          # dimension of axis 2
dim3=32          # dimension of axis 3
nspace=3         # number of physical coordinates per point
veclen=3         # number of components at each point
data=float       # data type (byte, integer, float, double)
field=uniform    # field type (uniform, rectilinear, irregular)
```

In the binary area, the data area occupies this amount of space:

(27 * 25 * 32) * 4 * 3 = 259,200 bytes

The coordinates area occupies (2 * 4) * 3 bytes. The total binary area occupies 259,224 bytes.

Example 3

The following ASCII header describes an irregular volume (3D irregular field) with one single-precision value for each field element. The binary area includes an (X,Y,Z) coordinate triple for each field element, indicating the corresponding point in physical space. This format might be used to represent velocity data from a simulation of fluid flow around a curved object (e.g. an airplane wing). Note that the data type is specified as **xdr_float**. This means that the data file is written in Sun's external data representation (XDR) format. This allows for transporation of field files between machines of dissimilation data storage architectures ("big-endian" vs "little-endian"). When using **read field**, if **Auto** is selected on the control panel, **read field** will look at the data specification. If it is simply **float**, (or **integer**, or **double**), it will assume

the file is written in the system's native format. If it is **xdr_integer**, **xdr_float**, or **xdr_double** it will translate the XDR format into its native format. If **Portable** is selected rather than **Auto**, it will always assume the file is written in XDR format.

```
# AVS field file
ndim=3           # number of dimensions in the field
dim1=40          # dimension of axis 1
dim2=32          # dimension of axis 2
dim3=32          # dimension of axis 3
nspace=3         # number of physical coordinates per point
veclen=1         # number of components at each point
data=xdr_float   # data type (byte, integer, float, double)
field=irregular  # field type (uniform, rectilinear, irregular)
```

In the binary area, the data area occupies this amount of space:

(40 * 32 * 32) * 4 * 1 = 163,840 bytes

The coordinates area occupies this amount of space:

(40 * 32 * 32) * 4 * 3 = 491,520 bytes

Data-Parsing Input

In its second input mode, **read field** can convert a certain class of data stored in ASCII, FORTRAN unformatted, or pure binary data files into AVS field format. To import data into AVS, you must create an ASCII description file that defines the structure of the AVS field to make. The first part of this description file is identical in format and meaning to the ASCII header file described above.

The second part of this file contains commands that specify which files contain the data or coordinate information, its data type (ASCII, FORTRAN unformatted, or binary), and simple parsing instructions. **read field** can read a file that is parseable by this general scheme:

skip *n* lines or bytes
move over an **offset** of *m* columns on this line (ASCII only)
read the value
do until *# of values needed*
{
  take *p* **stride**(s) to the next value
  read the value
}

Note that **read field** follows parsing instructions you supply; it does not "figure out" the format of a data file.

The ASCII description file, data, and coordinate information for rectilinear and irregular data can all be read from different files. If the resulting AVS field contains a vector of data values at each point, each vector element can also be read from a separate file.

The ASCII description file must have a *.fld* file suffix or the **read field** file browser will not display the file.

read field data parsing capability is meant to be used only once, in order to convert data to AVS field format. The parsing activity makes **read field** run more slowly than when it reads a file that is already in AVS field format. Once you have read your data using **read field**'s data-parsing mode, you should use the **write field** module to store it permanently on disk in AVS field file format.

ASCII Description File

As the example below shows, the ASCII description file contains a series of text lines that define the AVS field to construct. Each line is either:

- A comment
- A required line in the form *token=value*
- An optional line in the form *token=value*
- A **variable** or **coord** parsing specification

The following ASCII description file imports three-dimensional curvilinear data with a vector of values at each point into an AVS field of type "field 3D 3-vector irregular float". This type of data often occurs in computational fluid dynamics applications. The data and coordinate information are in separate files, both of which were written as straight binary data. Both files happen to have a serial organization. In the data file, all of vector element 1's values appear, then all of vector element 2's, then all of vector element 3's values. In the X, Y, Z coordinate file, all the X coordinate values appear, then all the Y's, then all the Z's.

Each line's meaning is explained in detail below.

```
# AVS field file     the string "# AVS" must be the first five
#                    characters in the file.
#
#                    When a '#' character appears in a line,
#                    the rest of the line is a comment
#
ndim=3                  # REQUIRED - the number of dimensions
dim1=40                 # REQUIRED - dimension of axis 1
dim2=32                 # REQUIRED - dimension of axis 2
dim3=32                 # REQUIRED - dimension of axis 3
nspace=3                # REQUIRED - coordinates per point
veclen=3                # REQUIRED - components at each point
data=float              # REQUIRED - data type
field=irregular         # REQUIRED - field type
min_ext=-1.0 -1.0 -1.0  # OPTIONAL - coordinate space extent
max_ext=1.0 1.0 1.0     # OPTIONAL - coordinate space extent
label=x-velocity        # OPTIONAL - label for variable 1
label=y-velocity        # OPTIONAL - label for variable 2
label=z-velocity        # OPTIONAL - label for variable 3
unit=miles-per-second   # OPTIONAL - unit label for variable 1
unit=miles-per-second   # OPTIONAL - unit label for variable 2
```

```
unit=miles-per-second   # OPTIONAL - unit label for variable 3
min_val=-2.1 -0.3 -3.7  # OPTIONAL - minimum data values
max_val=5.79 3.54 1.50  # OPTIONAL - maximum data values
#
# For each coordinate X, Y, and Z: data reading instructions
#
coord 1 file=/jetdata/wing.bin filetype=binary skip=12
coord 2 file=/jetdata/wing.bin filetype=binary skip=163852
coord 3 file=/jetdata/wing.bin filetype=binary skip=327692
#
# For each value in the vector: data reading instructions
#
variable 1 file=/jetdata/wdata.bin filetype=binary skip=28
variable 2 file=/jetdata/wdata.bin filetype=binary skip=163868
variable 3 file=/jetdata/wdata.bin filetype=binary skip=327708
```

Any characters following (and including) **#** in a header line are ignored.

NOTE: The first five characters in the ASCII description file *must* be "# AVS" or **read field** will not recognize the file as valid.

The example above shows all of the required *TOKEN=VALUE* token names: an ASCII description file that is missing one or more of these lines causes **read field** to generate an error. Required *TOKEN=VALUE* pairs are stored in the AVS field that **read field** produces as output.

Optional *TOKEN=VALUE* pairs are stored in the output AVS field as well, if they are provided. **min_ext** and **max_ext** are stored in the output AVS field even if they are not specified, as **read field** calculates them if they are not provided.

The **variable** and **coord** lines are not stored in the output AVS field. They are only instructions to **read field**.

With the exception of filenames, ASCII description file specifications are *not* case-sensitive. You can surround the **=** character with any amount of white space (including none at all). For example, "dim2 = 32", "DIM 2 =32", and "Dim2=32" are all equivalent.

Below is a complete description of all of the tokens recognized by the **read field** parser:

ndim = *value* (required)
The number of computational dimensions in the field. For an image, **ndim** = 2. For a volume, **ndim** = 3.

dim1 = *value* (required)
dim2 = *value* (required, depending on total number of dimensions)
dim3 = *value* (required, depending on total number of dimensions)
...
The dimension size of each axis (the array bound for each dimension of the computational array). The number of **dim***x* entries must match the value of **ndim**. For instance, if you specify a 3D computational space field

(**ndim**=3), you must specify the length of the X dimension (**dim1**), the length of the Y dimension (**dim2**), and the length of the Z dimension (**dim3**). Note that counting is 1-based, not 0-based. If you have scatter data (**ndim=1**) in 3D coordinate space (**nspace=3**), there will be only **dim1=***value.*

nspace = *value* (required)
The dimensionality of the physical space that corresponds to the computational space (number of physical coordinates per field element).

In many cases, the values of **nspace** and **ndim** are the same — the physical and computational spaces have the same dimensionality. But you might embed a 2D computational field in 3D physical space to define a manifold; or you might embed a 1D computational field in 3D physical space to define an arbitrary set of points (a "scatter").

veclen = *value* (required)
The number of data values for each field element. All the data values must be of the same primitive type (e.g. **integer**), so that the collection of values is conceptually a **veclen**-dimensional vector. If **veclen**=1, the single data value is, effectively, a scalar. Thus, the term *scalar field* is often used to describe such a field.

data = byte (one of the four options is required)
data = integer
data = float
data = double
The primitive data type of all the data values. **xdr_integer, sdr_float,** and **xdr_double** may also be specified. If **Auto** is selected on input, **read_field** will examine the **data=** field. If the specification is unqualified, it assumes the file is written in the native format for the platform ("big-endian" vs "little-endian"). If it is **xdr_*value***, it will read it as Sun's external data format (XDR) and translate it into the native format. If **Portable** is selected, it assumes XDR format no matter what is placed here.

field = uniform (one of the three options is required)
field = rectilinear
field = irregular
The field type. A **uniform** field has no computational-to-physical space mapping. The field implicitly takes its mapping from the organization of the computational array of field elements.

For a **rectilinear** field, each array index in each dimension of the computational space is mapped to a physical coordinate. This produces a physical space whose axes are orthogonal, but the spacing among elements is not necessarily equal.

For an **irregular** field, there is no restriction on the correspondence between computational space and physical space. Each element in the computational space is assigned its own physical coordinates.

min_ext = *x-value [y-value] [z-value]...* (optional)
max_ext = *x-value [y-value] [z-value]...* (optional)
The minimum and maximum coordinate value that any member data point occupies in space, for each axis in the data. If you do not supply this value, **read field** calculates it and stores it in the output AVS field data structure. This value can be used by modules downstream to, for example, size the **volume bounds** drawn around the data in the Geometry Viewer or put minimum and maximum values on coordinate parameter manipulator dials (**probe**). Values can be separated by blanks and/or commas.

If you do not know the extents, don't guess—let **read field** calculate them. Most downstream modules use whatever values are supplied, without checking their validity. If the wrong numbers are specified, incorrect results will be computed.

label = *string1 [string2] [string3]...* (optional)
Allows you to title the individual elements in a vector of values. These labels are stored in the output AVS field data structure. Subsequent modules that work on the individual vector elements (for example, **extract scalar**) will label their parameter widgets with the strings provided here instead of the default "Channel 0, Channel 1...", etc. You can either use one **label** line as shown here, or separate label lines as shown in the example above. In either case, the labels are applied to the elements of the vector in the order encountered. You can also label single scalar values, though downstream modules may ignore such a label. Any alphanumeric string is acceptable. Strings can be separated by blanks and/or commas.

unit = *string1 [string2] [string3]...* (optional)
Allows you to specify a string that describes the unit of measurement for each vector element. You can either use one *unit* line as shown here, or separate unit lines as shown in the example above. In either case, the unit specifications are applied to the elements of the vector in the order encountered. You can also specify the unit for a single scalar value, though downstream modules may ignore it. Any alphanumeric string is acceptable. Strings can be separated by blanks and/or commas.

min_val = *value [value] [value]...* (optional)
max_val = *value [value] [value]...* (optional)
For each data element in a scalar or vector field, allows you to specify the minimum and maximum data values. These values are stored in the output AVS field data structure. This is used by subsequent modules that need to normalize the data. Values can be separated by blanks and/or commas.

read field does not calculate these values if you do not supply them (unlike **min_ext** and **max_ext**). If you do not know these values, don't guess—just leave these optional lines out. In this case, the **write field** module can, at your instruction, compute these values when it creates an AVS field file. Most downstream modules use whatever values are supplied, without checking their validity. If the wrong numbers are specified, incorrect results will be computed.

variable *n* **file=***filespec* **filetype=***type* **skip=***n* **offset=***m* **stride=***p*
coord *n* **file=***filespec* **filetype=***type* **skip=***n* **offset=***m* **stride=***p*

Both of these specifications must be on a *single* line. There is no support for continuation characters.

variable specifies where to find *data* information, its type, and how to read it.

coord specifies where to find *coordinate* information, its type, and how to read it. It is used when the data is **rectilinear** or **irregular**.

The individual parameters are interpreted as follows:

- *n* — An integer value that specifies which element of a data vector or which coordinate (1 for x, 2 for y, 3 for z, etc.) the subsequent read instructions apply to. **n** does not default to 1 and must be specified.
- **file** = *filespec* — The name of the file containing the data or coordinates. The *filespec* can be an absolute full pathname to a file, or it can be a *filespec* relative to the directory that contains the field ASCII header. For example, an absolute pathname might be */home/myuserid/experiment/data*. In a relative pathname specification, if the ASCII file of field parsing instructions exists in the file */home/myuserid/experiment/readit.fld* and the data and coordinates file are in the subdirectory */home/myuserid/experiment/data*, you can name these files as *xdata/xyzs* and *data/values*. The advantage of this second approach is that you can move the directories containing your data around without having to change the contents of the ASCII parsing instruction file.
- **filetype = ascii** — **ascii** means that the data or coordinate information is in an ASCII file. In ASCII files, float data can be specified in either real (0.1) or scientific notation (1.00000e-01) format interchangeably.
- **filetype = unformatted** — the file is written in FORTRAN unformatted format. (FORTRAN unformatted data is binary data with additional words written at the beginning and end of each data block stating the number of bytes or words in the data block.) In general, **read field** can read unformatted files where all variables of one type (for example, all the X coordinates) were output as one "record" in a single write statement. This is usually the case.
- **filetype = binary** — the file is written in straight binary format, such as that produced by Unix output routines, *write* and *fwrite*.

  Note the warning on binary compatibility among different hardware platforms on the **read field** man page.

  In each case, **read field** will use the data type specified in the earlier **data={byte,float,integer,double}** statement when it interprets the file.
- **skip** = *n* — For **ascii** files, **skip** specifies the number of *lines* to skip over before starting to read the data. Lines are demarked by newline characters.

  For **binary** or **unformatted** files, **skip** specifies the number of *bytes* to skip over before starting to read the data.

  There are two motivations for **skip**. First, data files often include header information irrelevant to the AVS field data type. Second, if the file contains, for example, all X data values, then all Y data values,

skip provides a way to space across the irrelevant data to the correct starting point.

skip can only be used once at the start of the file. There is no way to **skip**, read, **stride**, then **skip** again.

You must simply know what value to use for **skip** based on your knowledge of the software that produced the original data file, the number of data elements, and the type (byte, float, double, integer, etc.)

skip defaults to 0.

- **offset** = *m* — **offset** is only relevant to ASCII files; it is ignored for binary and unformatted files. **offset** specifies the number of columns to space over before starting to read the first datum. (The **stride** specification determines how subsequent data are read.) Hence, to read the fourth column of numbers in an ASCII file, use **offset=3**.

  In ASCII files, columns must be separated by one or more blank characters. Commas, semicolons, TAB characters, etc., are *not* recognized as delimiters. If necessary, edit ASCII files to meet this restriction.

  offset defaults to 0 (the first column, no columns spaced over).

- **stride** = *p* — **stride** assumes you are "standing on" the data value just read. **stride** specifies how many "strides" must be taken to get to the next data value. In ASCII files, **stride** means stride forward *p* delimited items. In binary and unformatted files, **stride** means stride forward *p * the size of the data type* (byte, float, double, integer). In a file where the data or coordinate values are sequential, one after the other, the **stride** would be 1. Note that this presumes homogeneous data in binary and unformatted files — double-precision values could not be intermixed with single precision values.

  stride defaults to 1.

  The stride value will be repeatedly used until the number of data items indicated by the product of the dimensions (e.g. dim1 * dim2 * dim3) have been read.

Example 1

Here are some **skip**, **offset**, and **stride** examples for ASCII data. "A's" are vector component 1; "B's" are vector component 2.

ASCII file organization 1:

```
X       Y       Z       A       B
1       1       1       A1      B1
2       2       2       A2      B2
3       3       3       A3      B3
4       4       4       A4      B4
5       5       5       A5      B5
```

To read A: skip=1, offset=3, stride=5 To read B: skip=1, offset=4, stride=5

ASCII file organization 2:

```
A1      A2      A3      A4      A5
A6      A7      A8      A9      A10
A11     A12     A13     A14     A15
B1      B2      B3      B4      B5
B6      B7      B8      B9      B10
B11     B12     B13     B14     B15
```

To read A: skip=0, offset=0, stride=1 To read B: skip=3, offset=0, stride=1

ASCII file organization 3:

```
A1      B1      A2      B2      A3      B3
A4      B4      A5      B5      A6      B6
A7      B7      A8      B8      A9      B9
A10     B10     A11     B11     A12     B12
```

To read A: skip=0, offset=0, stride=2 To read B: skip=0, offset=1, stride=2

ASCII file organization 4:

```
TEMP1=A1 TEMP2=A2 TEMP3=A3 TEMP4=A4
TEMP5=A5 TEMP6=A6 TEMP7=A7 TEMP8=A8
PRESS=B1 PRESS=B2 PRESS=B3 PRESS=B4
PRESS=B5 PRESS=B6 PRESS=B7 PRESS=B8
```

read field cannot read this file until the data labels and equal signs are edited out.

Example 2

You have some 3-dimensional, curvilinear data that projects the amount and location of wood that will be eaten after five years by a colony of termites that has entered a 14th century Scandanavian grain silo structure at a particular spot in its base. The data is in one ASCII file, *decay.dat*, as a long sequential, numbered list of 1250 consumed-wood values that looks like this:

```
1,1002.707;
2,1443.971;
3,1307.069;
4,1240.354;
5,1778.715;
    ...
```

The coordinates that correspond to the data values are in a separate ASCII file, *loc.dat*, that looks like this:

```
LOC,1,0,0.2500000,0.0000000e+00,1.105255,0.0000000e+00;
LOC,2,0,0.2500000,0.0000000e+00,1.000000,0.0000000e+00;
LOC,3,0,0.5000000,0.0000000e+00,1.552552,0.0000000e+00;
LOC,4,0,0.5000000,0.0000000e+00,1.442042,0.0000000e+00;
LOC,5,0,0.5000000,0.0000000e+00,1.331531,0.0000000e+00;
    ...
```

In the data file, the second column represents the data. In the coordinate file, the fourth through sixth columns are the x, y, and z coordinates, respectively.

First, to read this data, you must use a text editor to globally edit out the commas and semi-colons, changing them to spaces. The files now look like:

```
1 1002.707
2 1443.971
  ...

LOC 1 0 0.2500000 0.0000000e+00 1.105255 0.0000000e+00
LOC 2 0 0.2500000 0.0000000e+00 1.000000 0.0000000e+00
  ...
```

The following ASCII description file, *decay.fld*, would import the data into AVS field format:

```
# AVS Field File
#
# Termite Decay after Five Years
#
ndim=3           # number of dimensions in the field
dim1=25          # dimension of axis 1
dim2 =10         # dimension of axis 2
dim3 =5          # dimension of axis 3
nspace=3         # number of physical coordinates
veclen=1         # number of elements at each point
data=float       # data type (byte, integer, float, double)
field=irregular  # field type (uniform, rectilinear, irregular)
coord 1 file=/Termite/loc.dat filetype=ascii offset=3 stride=7
coord 2 file=/Termite/loc.dat filetype=ascii offset=4 stride=7
coord 3 file=/Termite/loc.dat filetype=ascii offset=5 stride=7
variable 1 file=/Temite/decay.dat filetype=ascii offset=1 stride=2
```

Example 3

The following ASCII description file specifies how to convert the volume data in the file */usr/avs/data/volume/hydrogen.dat* into an AVS field. *hydrogen.dat* is a series of binary byte values that represent the probability of finding an electron at various locations around a hydrogen nucleus. The first three bytes in the file give the X, Y, and Z dimensions of the data—however, this information is not part of the actual data and must be skipped over. You could examine these three bytes and determine what to use for the dimensions in the ASCII description file. Thereafter, it is just a matter of reading successive bytes. **offset** is not used because this is not an ASCII file. **stride** is allowed to default to 1.

```
# AVS field file
ndim=3             # number of dimensions in the field
dim1=64            # dimension of axis 1
dim2=64            # dimension of axis 2
dim3=64            # dimension of axis 3
nspace=3           # number of physical coordinates per point
veclen=1           # number of components at each point
data=byte          # data type (byte, integer, float, double)
field=uniform      # field type (uniform, rectilinear, irregular)
variable 1 file=/usr/avs/data/volume/hydrogen.dat filetype=binary skip=3
```

Example 4

This ASCII description file specifies how to use **read field** to convert the image data in */usr/avs/data/image/mandrill.x* into an AVS field. The first two words in *mandrill.x* are 32-bit integers that specify the horizontal and vertical dimensions of the image. This information must be skipped over — you must supply it in the ASCII description file. Thereafter, *mandrill.x* is a succession of 32-bit straight binary words, one word per pixel. However, in AVS, each of these words is considered to be a vector of 4 bytes. The first byte is the "alpha" (or "transparency") value for the pixel, and the second through fourth bytes are the red, green, and blue values for each pixel. Thus, this whole file is treated as a series of binary bytes.

```
# AVS field file
#
ndim = 2                        # number of dimensions in the field
nspace=2                        # number of physical coordinates
dim1=500                        # dimension of axis 1
dim2=480                        # dimension of axis 2
veclen=4                        # number of components at each point
data=byte                       # data type (byte, integer, float, double)
field=uniform                   # field type (uniform, rectilinear, irregular)
label = alpha, red, green, blue     # labels the vector elements
variable 1 file=/usr/avs/data/image/mandrill.x filetype=binary skip=8 stride=4
variable 2 file=/usr/avs/data/image/mandrill.x filetype=binary skip=9 stride=4
variable 3 file=/usr/avs/data/image/mandrill.x filetype=binary skip=10 stride=4
variable 4 file=/usr/avs/data/image/mandrill.x filetype=binary skip=11 stride=4
```

Example 5

This ASCII description file reads a FORTRAN unformatted ARC 3D dataset. The file is 34x34x34, made up of floating point numbers. It is irregular; therefore there is both computational and coordinate data in two separate files. The vector length is six. The data file is written as a 24 byte header that must be skipped over followed by all vector 1 values, all vector 2 values, etc. The coordinate file is written as a 12 byte header (a full word for each of the X, Y, and Z dimensions) followed by all X cooordinates, all Y coordinates, then all Z coordinates. The person is using a relative file specification—the filenames will be interpreted relative to the directory of this ASCII description file.

```
# AVS field file
# to read an Arc 3D file that's 34x34x34z
ndim = 3
dim1 = 34
dim2 = 34
dim3 = 34
nspace = 3
veclen = 6
data = float
field = irregular
#
coord 1 file=for003.dat filetype=unformatted skip=20 stride=1
coord 2 file=for003.dat filetype=unformatted skip=157236 stride=1
coord 3 file=for003.dat filetype=unformatted skip=314452 stride=1
#
```

```
variable 1 file=for004.dat filetype=unformatted skip=32 stride=1
variable 2 file=for004.dat filetype=unformatted skip=157248 stride=1
variable 3 file=for004.dat filetype=unformatted skip=314464 stride=1
variable 4 file=for004.dat filetype=unformatted skip=471680 stride=1
variable 5 file=for004.dat filetype=unformatted skip=628896 stride=1
variable 6 file=for004.dat filetype=unformatted skip=786112 stride=1
```

Given that the coordinate file header is 12 bytes, why is the **skip** value 20? It is 20 because **read field** must be directed to skip over the one word FORTRAN unformatted record header, and the one word FORTRAN unformatted record trailer (12+4+4=20). The same 20 bytes must be added to the **skip** value for **coord**s 2 and 3. Similarly, the data file's 24 byte header must have 8 bytes added to it for a total of 32. **read field** correctly deals with the remaining "invisible" FORTRAN unformatted record header and trailer words in the remainder of the file, provided that all values pertaining to a dimensions (X, Y, or Z) and / or all values pertaining to a vector (e.g., all x-momentums) were written as one record. It will also work if the records were written as repeating groups (e.g., X, Y, Z; X, Y, Z; X, Y, Z; etc.) or (vec1, vec2, vec3, vec4, vec5, vec6: repeat). It will not work if the output was generated as "first half of X's; second half of X's", since the intermediate FORTRAN formatting words will throw off its **stride**s.

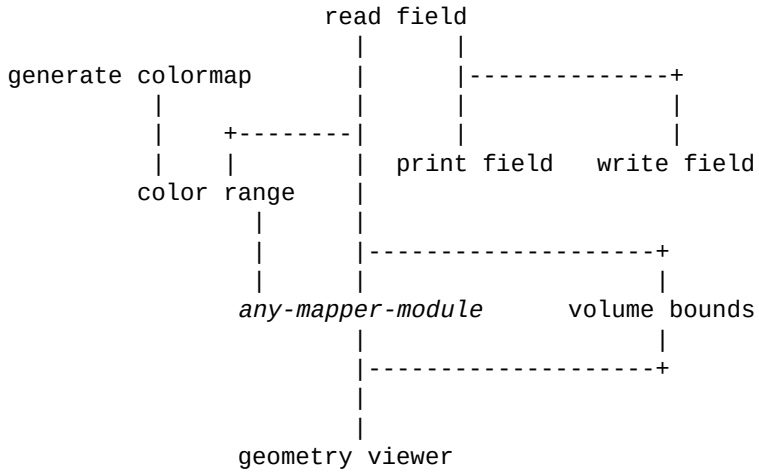
Hints on Using read field

If you did not write the application that produces your data, and you instead use an application acquired in binary form from an outside source, then you may not know what the format of the application's binary or unformatted Fortran data files actually *is*. This can be difficult to determine just by looking at the raw data files. However, most established packages do document their internal file format and you can usually contact the vendor's support organization for this information.

It can be hard to tell when you've read a dataset correctly. To check on this, use the **print field** module. The **print field** module will display the contents of an AVS field in ASCII format. You can look at the results in **print field**'s output display window. If this display window is too narrow, **print field** also writes its output to a temporary file that you can view with any text editor.

You can also use the **volume bounds** module to outline the resultant field's extents. If there has been a mistake reading the coordinate information for rectilinear and irregular datasets, it will usually show up very graphically with **volume bounds**.

Here is one useful "debugging" network for importing uniform or irregular volume data.



You can replace *any-mapper-module* with any of the modules that produce a colored geometry picture of the data (such as the **orthogonal slice/field to mesh** module pair). If you are trying to produce a vector field, it might be easier to try reading just one vector element at first until all the other variables are correct. You would then include an **extract scalar** module between the **read field** and mapper module in this network.

Once you have read the data correctly, you should write it permanently to disk in binary native AVS field format with the **write field** module. **read field** reads native format AVS fields much more quickly than it is able to parse foreign-format files.

You can also be creative with **read field**. For example, many imaging applications produce a series of separate images that actually represent slices through a volume (such as density slices through a cranium, for example). The scientist may wish to deal with the images as a 3D volume of data. You could write a very simple C or Fortran program that generates the uniform X and Y coordinates of the images as ASCII data, as well as a series of rectilinear Z coordinates that position the slices accurately in space. You can then use **read field** to read the original data file, and the new coordinate file to produce a rectilinear 3D volume representation of the data.

read field Limitations

read field does have limitations:

- Its parsing commands primarily deal with files that are structured with header information that can be skipped over, followed by data in a regular, repeating format. **read field**, for example, could not be used to read an image file that contained run-length encoded data.
- In many established packages (including PLOT3D) the data that is stored in the input data file is not the final data that is actually used by the packages when they perform their computations. For instance, values such as temperature or friction may be *calculated* from the raw data using well-known constant values and formulae. **read field** cannot, in principle, a priori know what these automatic calculations should be. It may be nec-

essary to write modules that further process the original field data into its final, usable form.

AVS Module: read plot3d

The **read plot3d** module in the Unsupported module library reads computational fluid dynamics data files in the National Aeronautics and Space Administration's PLOT3D format and converts them into AVS field format. (To access the Unsupported module library in the Network Editor, use the **Module Tools** submenu and the **Select Module Library** button.) There are two types of PLOT3D files, the XYZ grid files that specify the irregular coordinate information, and the Q solution files that contain a vector of values for each point in the grid.

XYZ and Q file pairs can contain a single set of grid / data mappings, or multiple grid / data mappings. The XYZ file can also contain an IBLANK value for each point, although these data values will not be stored in the output field. The data within the files can be in either binary, or Fortran formatted or unformatted format. XYZ grid file and Q solution file formats must match in all respects.

read plot3d requires that you know the format (dimensionality, whole / plane, number of grids, binary / formatted / unformatted, and whether IBLANK values are present) of the PLOT3D files that you are trying to read. It does not check to verify that the values it is given map reasonably to the data.

Q solution files contain three to five floating point values for each point in the grid: X momentum (1D), Y momentum (1D and 2D), Z momentum (1D, 2D, and 3D), density, and stagnation. The four header values (FSMACH, ALPHA, RE and TIME) are ignored.

read plot3d does impose some practical limits to the size of the data: No one dimension can be larger than 1,000,000; the output data can have no more than 1,000,000,000 points in any one grid; and the maximum number of data grids is 50.

read plot3d outputs an AVS field (field irregular float 1D, 2D, or 3D of 3-, 4-, or 5-vector). The AVS field output will match the dimensionality of the original PLOT3D dataset. At each point in the grid will be three to five floating point values: density, X momentum (and Y momentum, and Z momentum, if appropriate), and stagnation, in that order. **Note:** The output AVS field represents only the *first* grid of multi-grid parameter files. There is no way to pack multiple grids into a single AVS field.

An additional module, **cfd values**, is often used in conjunction with **read plot3d**. **cfd values** inputs the field data from **read plot3d** (including density, x-momentum, y-momentum, z-momentum, and stagnation), and allows the user to compute the following values:

- energy

- pressure
- enthalpy
- mach number
- temperature
- total pressure
- total temperature

AVS Module: read image

Although there is a distinct AVS image file format for storing images on disk, once AVS reads an image into a network, it treats image data as a 2D 4-vector byte AVS *field*. Therefore, the best way to approach importing image data is to view the problem as converting your image data into an AVS field.

read image Data File Format

The **read image** module and the Image Viewer subsystem (also implemented as the **image viewer** module) can read a file that contains an image—a 2D array of pixel values. In AVS, such files should have names that end with a *.x* suffix.

The file must begin with a two-word header, which specifies the dimensions of the image:

first word: number of pixels in horizontal direction (32-bit integer)
second word: number of pixels in vertical direction (32-bit integer)

There is no explicit limit on the size of an image.

The remainder of the file is a sequence of 4-byte (32-bit) words, one for each pixel of the image. The pixels are arranged rowwise; there is no padding at the end of a row.

The four bytes of a pixel are interpreted as four component values in the range 0..255. Three of the bytes are the red, green, and blue color components. The fourth byte is an auxiliary field, which is used by some AVS modules to represent an opacity / transparency value:

```
+-----------+-----------+-----------+-----------+
| auxiliary |    red    |   green   |   blue    |
+-----------+-----------+-----------+-----------+
```

The AVS *image* data file format is shown below. Image data files should have names that end with a *.x* suffix.

```
number_of_pixels_in_x_dimension
number_of_pixels_in_y_dimension
pixel_1
pixel_2
pixel_3
```

```
.
.
.
pixel_n
```

In the example above, the total number of pixels is:

(*number_of_pixels_in_x_dimension * number_of_pixels_in_y_dimension*)

Note: The AVS image data type is primarily designed to hold data that was originally produced as RGB (red, green, blue) data, whether as full 24-bit true color or some more limited format. Many other kinds of scientific data are *also* referred to as "image data." For example, medical imaging devices produce 2D or 3D arrays of numbers that represent some quantity, such as density. This is usually called "image data." However, such data is best represented as an *AVS field*, not as an AVS *.x* image. A field can always be converted into an image that the AVS Image Viewer can manipulate.

AVS Module: read volume

Although there is a distinct AVS volume file format for storing 3D uniform volumes on disk (where each element is a byte holding a value from 0-255); once AVS reads an volume into a network, it treats the volume data as a 3D uniform scalar byte AVS *field*. Therefore, the best way to approach importing this narrow class of volume data is to view the problem as converting your volume data into an AVS field. You are referred to the "Field Data" discussion earlier in this chapter.

read volume Data File Format

Measurement data often takes the form of a 3-dimensional array, which corresponds to a uniform lattice in 3D space. Each array value indicates one measurement (temperature, pressure, etc.) at the corresponding lattice point. Such data can be represented as a *uniform 3D field*, as described in the preceding section. For convenience, AVS also provides a simpler *volume data* format to accommodate this type of data.

The AVS volume data format requires that each value in the data array be a byte. (For other data types (e.g. single-precision), you must use the more general *field* construct.) Volume data files should have names that end with a *.dat* suffix.

Note: The **volume data** and **field** file formats are not compatible.

A volume data file begins with a three-byte header, which specifies the size of the array in the first (X), second (Y), and third (Z) dimensions. Since each dimension's size must be expressed as a 1-byte number, the largest array supported by this file format is 255x255x255.

The remaining contents of the file are the values of a 3D array of bytes, in column-major order ("FORTRAN-style"). For example, the values in a 50x20x10 array would be stored as follows (using FORTRAN notation):

DATA(1,1,1)
DATA(2,1,1)
DATA(3,1,1)
...
DATA(50,1,1)
DATA(1,2,1)
DATA(2,2,1)
DATA(3,2,1)
...
DATA(1,20,1)
DATA(2,20,1)
DATA(3,20,1)
...
DATA(1,20,2)
DATA(2,20,2)
DATA(3,20,2)
...
DATA(1,20,10)
DATA(2,20,10)
DATA(3,20,10)
...
DATA(50,20,10)

The volume data file format is shown below:

```
number_of_voxels_in_x_dimension
number_of_voxels_in_y_dimension
number_of_voxels_in_z_dimension
voxel_1
voxel_2
voxel_3
  .
  .
  .
voxel_n
```

In the example above, the total number of voxels is:

(*number_of_voxels_in_x_dimension* *
number_of_voxels_in_y_dimension *
number_of_voxels_in_z_dimension)

Programming Examples

If none of the above alternatives for importing data into AVS field format is satisfactory, you will need to write a module that will read the data into AVS field format. AVS contains a library of routines for building fields, and storing

and accessing field elements. These routines are described in "Appendix A" of the *AVS Developer's Guide*.

There are several source code examples of modules that read data into AVS field format. All are found in the directory */usr/avs/examples*.

read_image.c
read_image_f.f
These modules, one in C and one in Fortran, read data in AVS image file format into a 2D 4-vector byte field.

read_vol.c
read_vol_f.f
These modules, one in C and one in Fortran, read data in AVS volume file format into a 3D scalar byte field.

read_scans.c
This example module reads a set of hypothetically-formatted images stored in separate files, each representing a planar scan, into a uniform 3D integer field (a 3D volume of integer data). It assumes a certain location for the input files, and a pattern to their file names; fixed-size images (64 x 64), a fixed-size header in each image file, an integer of data for each pixel, and that the image data is stored in "row major order", that is that the second index of the data varies most rapidly (1,1 1,2 1,3, etc.). *read_scans.c* can be used as a template for a variety of image file formats that need to be converted to 2D or 3D AVS fields.

read_plot3d.c
The file *read_plot3d.c* is a C program that reads NASA PLOT3D 3D whole format datasets into AVS field format.

Note: The **read plot3d** example is *not* the same as the **read plot3d** module in the AVS Unsupported library.

This is probably the most sophisticated example source code module since its module description section includes the definition of several different kinds of interface widgets, it must deal with allocating memory to store the field based upon the changing size of the input datasets, it reads column-major order data, it reads vector data instead of simple scalars, and it uses formulae to produce output values not found in the input data.

Geometry Data

AVS geometry data contains a description of three-dimensional geometries and scenes, including object type, coordinate data, surface attributes, rendering modes, material properties, and transformations.

AVS objects are one of the following types:

Polyhedron
A list of vertices with an indirect list of pointers into these vertices for each polygon.

Polygon
A list of vertices for each polygon.

Mesh
A 2D array of values, either scalars (for a height field) or vertices.

Sphere
A list of center points and radii.

Polytriangle
A single list of vertices representing polylines, disjoint lines, or a triangle mesh, where the connectivity is implied by the particular data type.

AVS has routines that allow a module to change several properties of an object, including:

- The geometric data defining the object
- Surface or line color
- Render mode (Gouraud, Phong, wireframe, etc.)
- Parent (the name of the parent object)
- Object material properties
- Object, camera, and light transformation
- Object visibility, deletion
- Object color, light source color and camera background color
- Camera background color
- Light source on / off, type
- Texture mapping
- Transformation mode (controls how objects are transformed)
- Selection mode (controls how objects are picked)
- Center of rotation and scaling
- Viewable region of data
- Viewing projection

AVS represents graphical objects in an external binary data format called *geom*. *geom* format files are created using the *libgeom* library documented in the "Geometry Library" appendix to the *AVS Developer's Guide*.

AVS Module: geometry viewer

The **geometry viewer** module provides access within an AVS network to the complete Geometry Viewer subsystem. Many different modules can supply input geometries. That is, many *geometry*-format outputs can be connected to

geometry viewer's geometry input port. All the objects will be combined into a single scene. Each module providing input to **geometry viewer** can define attributes and geometries for any number of objects. Each of these modules can also define a hierarchical relationship among its objects.

You can also invoke **geometry viewer** with no inputs, so that the "scene" is initially empty. Objects can be added to a scene either by upstream modules or by the **Read Object** selection on the **geometry viewer** control panel. Geometries and descriptions sent by upstream modules can be saved to files using the **Save Object** and **Save Scene** selections. In this way, you can save visualization results and retrieve them later with **Read Scene** or **Read Object**.

AVS Module: read geom

The **read geom** module reads a file containing an AVS *geometry* and outputs the geometry to one or more modules connected to its output port. The resulting object will be named after the file from which it was read.

This module reads *geom* files only. See the section below on "AVS Geometry Filters" for a description of how to create AVS *geom* files.

AVS Module: pdb to geom

The **pdb to geom** module converts molecular data in Brookhaven Protein Data Bank format into AVS *geom* format. It is in the Data Input column of the Network Editor's module palette. It expects an input file with a *.pdb* suffix. There are two sample input datasets, */usr/avs/filter/example/bwdna.pdb* and */usr/avs/filter/example/crambin.pdb*. (This latter can also be found in the */usr/avs/data/pdb* directory.)

For instance, the table below shows part of a file written in the Brookhaven Protein Data Bank format. This file defines the structure of a particular protein molecule called "crambin". There is an AVS data input module to read files in this format. Users can supply their own data input modules for other data formats.

```
ATOM      1  HN1 THR     1      17.017  14.972   4.068
ATOM      2  HN2 THR     1      16.297  13.912   2.883
ATOM      3  N   THR     1      16.982  14.095   3.587
ATOM      4  HN3 THR     1      17.707  14.470   3.008
ATOM      5  CA  THR     1      16.949  12.808   4.348
ATOM      6  C   THR     1      15.686  12.779   5.142
ATOM      7  O   THR     1      15.236  13.827   5.603
  ...
```

AVS Geometry Filters

There are a number of geometry filters supplied with AVS that will convert data into the *geom* format. These filters fall into two classes:

- Filters that read several file formats commonly found in the technical community. These geometry filters can be used to import existing datasets into AVS *geom* format.
- Filters that read ASCII files in a format unique to AVS. Each of these geometry filters inputs ASCII data to create one of the fundamental geometry structures manipulated by the *libgeom* programming library, such as polygons, polyhedrons, meshes, and spheres. These AVS-specific geometry filters can be used in two ways:
  - You might be able to construct an ASCII file version of your geometry data that one of these geometry filters could read to create a *geom* object. However, the geometry filters are probably not robust enough to be used on a production basis for a real application. The *mesh.c* filter has the most general utility.
  - The geometry filters serve as example programs that show how to construct the primitive *geom* structures using *libgeom* calls. Two of the geometry filters are written in Fortran in addition to C.

The sources to geometry filters are all located in the directory */usr/avs/filter*. The executable versions of geometry filters are all located in the directory */usr/avs/bin*. Sample datasets in the formats these geometry filters expect are located in */usr/avs/filter/example*.

The following common file formats have geometry filters:

Table 2-2. AVS-Readable Geometry File Formats: General Use

File Format	File Suffix	Source Filename	Executable Filename
Mathematica ThreeScript	*.ts*	--	ts_to_geom
Movie BYU	*.byu*	byu.c	byu_to_geom
Protein Data Bank	*.pdb*	pdb.c	pdb_to_geom
UNC	*.ppoly*	ppoly.c	ppoly_to_geom
Wavefront	*.wfront*	wfront_geom.c	wfront_to_geom

There is a sample Movie BYU format dataset in */usr/avs/filter/example/cube.byu*, and a sample Mathematica ThreeScript dataset in */usr/avs/filter/example/cone.ts*.

The following AVS-specific geometry filters are provided:

Table 2-3. AVS-Readable Geometry File Formats: AVS Specific

Object Type	Source Filename	Executable Filename
polyhedron	polyh.c	polyh_to_geom
disjoint polygon	polygon.c, polygon.f	polyg_to_geom

Table 2-3. AVS-Readable Geometry File Formats: AVS Specific

Object Type	Source Filename	Executable Filename
mesh	mesh.c, mesh.f	mesh_to_geom
sphere	sphere.c	sphere_to_geom

The source to each AVS-specific geometry filter includes a description of the ASCII file format it expects to read. There are sample polygon, polyhedron, and mesh ASCII data files in */usr/avs/filter/example*.

The remainder of this section describes how to use these geometry filters.

Automatic Data Filtering

AVS geometry filters are normally executed at the shell level. The geometry filter facility is unique in this regard—if, during a **Read Object** function within the AVS Geometry Viewer, AVS determines (using the filename extension) that the file you select is in a known data format, it invokes the appropriate geometry filter *automatically* to create a corresponding *geom* file on disk. It then reads in the object from the *.geom* file.

When you execute the AVS function **Read Object**, you select a filename in the File Browser window. If the file has a *.geom*, *.scr*, or *.obj* extension, AVS reads it in directly.

If the file has another extension, AVS determines whether the file contains data in a "known format," as follows:

- It looks in its */usr/avs/bin* directory for utility programs named *..._to_geom*. Each such geometry filter determines a particular filename extension. For instance, the geometry filter utility *wfront_to_geom* determines the extension *.wfront*.
- It compares the extension of the **Read Object** filename with its list of geometry filter extensions.
- If there is a match, AVS automatically invokes the appropriate geometry filter utility, creating a *geom*-format file of the same name, with a *.geom* extension. This file is created in the directory that the original file is located in.
- The **Read Object** function is completed by reading in the newly created *.geom* file.

Shell-Level Usage of Geometry Filter Utilities

Each of the geometry filters can also be invoked from the shell, in the usual fashion. Each one reads a single file from *stdin* and writes a single *geom*-format file to *stdout*. Typically, you should redirect *stdout* to a file whose extension is *.geom*, so that it is directly readable by the AVS application. Command-Line Options

Most of the geometry filters don’t accept any command line options. The exceptions are as follows:

The **ts_to_geom** filter utility accepts the following options:

-bbox *xmin xmax ymin ymax zmin zmax*
Define a bounding box to scale the object down non-uniformly.

-bratios *x y z*
Alter the aspect ratio of the bounding box. *x*=1, *y*=1, *z*=1 is a uniform aspect ratio. *x*=1, *y*=1, *z*=0.5 forces the Z dimension to be half the size of the X and Y dimensions.

-boxed
Put a wireframe box around the object.

-noscale
Do not attempt to scale the object at all.

The **pdb_to_geom** filter accepts the following option:

-balls
Use the sphere representation instead of the default ball-and-stick representation.

Postprocessor Filters

AVS supplies an additional set of geometry filters, which you can use to postprocess the output of the *geom*-format converters. For instance, if a file in Movie BYU format defines an object with normals that point inward, you can make a *.geom* file with normals that point outward as follows:

```
byu_to_geom < myobject.byu | geom_flip > myobject.geom
```

The **geom_flip** postprocessor reads and writes a *geom*-format file, flipping the normals of the object defined therein.

The following table lists the postprocessor geometry filters supplied. Except for **send_to** and **animate_to**, each filter reads a file from *stdin* and writes a file to *stdout*.

Table 2-4. Postprocessor Filters

Filter Name	Description
geom_flip	Flip normals of object.
geom_pickable [-non]	Make all objects pickable (non-pickable), so that their attributes can be set (cannot be set) individually.
geom_scale	Scales the object uniformly, so that it lies within the unit cube, (-1,-1,-1) to (1,1,1). Also, converts the objects's normals to unit length (normalizes them).
geom_to_normls [-scale] *length*	Produces a disjoint-line object that represents the normals of the input object. The **-scale** option specifies the length of the normals. The default length is 1.
geom_to_text	Produces an ASCII version of the input *.geom* file.
text_to_geom	Produces a *.geom* version of the input ASCII file. The combination of **geom_to_text** and **text_to_geom** can be used to transfer geometry files between machines with different byte ordering or different word sizes.
geom_split [*name***]**	Creates a series of *.geom* files, each of which contains one of the objects defined in the input file. This is used for input files that contain more than one *geom* object. Each output file is named *name.n***.geom**. If you don't specify the optional *name*, the files are named *split0.geom*, *split1.geom*, etc.
send_to *filename*	Causes a currently-executing AVS program to read the specified file, which should be a *.geom* file.
animate_to -file *name geom-file1 geom-file2*	Creates a script that defines a cycle object, stores the script as *name.***obj**, and casues a currently-executing AVS program to read the script file. The cycle includes the specified *geom-file* sequence.

Programming Examples

If none of the geometry filters listed above is satisfactory for your application, then you can either write a custom geometry filter, or write an AVS geometry-producing module. As mentioned earlier, the latter approach will be the most effective solution.

In either case, you must use the *libgeom* calls documented in the *AVS Developer's Guide* "Geometry Library" appendix to create an AVS-compatible geometry. The source code versions of all of the geometry filters described above are useful templates for generating geometry filters. There are also several geometry-producing modules in the */usr/avs/examples* directory.

Unstructured Cell Data

Unstructured cell data (UCD) is commonly used in structural analysis and computational fluid dynamics. A UCD data structure consists of an irregular coordinate **structure** (or "model") made up of **cells**. Cells may be points, lines, quadrilaterals, triangles, tetrahedrons, pyramids, prisms, or hexahedrons. Each cell has a corresponding number of **nodes**. Data can be associated with the entire structure, with each cell, and with each node. The data is structured as a set of **components**. Each component can be either a scalar or a vector.

The UCD data type and the library of routines for creating and manipulating UCD are described in detail in the *AVS Developer's Guide* "Unstructured Cell Data" appendix.

AVS Module: read ucd

read ucd reads an AVS-compatible UCD structure from a file. The file must have a *.inp* suffix or it will not be displayed in **read ucd**'s file browser.. The file may be ASCII or binary.

Binary UCD files have a different format than ASCII UCD files. Specifically, if a file is binary then it is assumed that it is in the format output by the module **write ucd**.

ASCII UCD files have a simple format described below. For a more detailed description of both ASCII and binary file formats, see the "Unstructured Cell Data" appendix of the *AVS Developer's Guide*.

ASCII UCD File Format

The input file cannot contain blank lines or lines with leading blanks. The numbers down the left correspond to the above descriptions and are not part of the ASCII file. Comments, if present, must precede all data in the file—comments within the data will cause read errors. The general order of the data is:

1. Numbers defining the overall structure, including the number of nodes, the number of cells, and the length of the vector of data associated with the nodes, cells, and the model.
2. For each node, its node-id and the coordinates of that node in space. Node-ids must be integers, but any number including non-

sequential numbers can be used. Mid-edge nodes are treated like any other node.

3. For each cell: its cell-id, material, type (hexahedral, pyramid, etc.), and the list of node-ids that correspond to each of the cell's verticies. (The UCD appendix shows the order in which cell verticies are numbered.)
4. For the data vector associated with nodes, how many components that vector is divided into (e.g., a vector of 5 floating point numbers may be treated as 3 components: a scalar, a vector of 3, and another scalar, which would be specified as 3 1 3 1).
5. For each node data component, a component label / unit label pair, separated by a comma.
6. For each node, the vector of data values associated with it.
7. That is the end of the node definitions. Cell-based data descriptions, if present, then follow in the same order and format as items 4, 5, and 6.
8. The single model-based data descriptions, if present, comes last.

```
    # <comment 1>
          .
          .
          .
    # <comment n>
1.  <num_nodes>  <num_cells>  <num_ndata>  <num_cdata>  <num_mdata>
2.  <node_id 1>  <x>  <y>  <z>
    <node_id 2>  <x>  <y>  <z>
          .
          .
          .
    <node_id num_nodes>  <x>  <y>  <z>
3.  <cell_id 1>  <mat_id>  <cell_type>  <cell_vert 1> ... <cell_vert n>
    <cell_id 2>  <mat_id>  <cell_type>  <cell_vert 1> ... <cell_vert n>
         .
         .
         .
    <cell_id num_cells>  <mat_id>  <cell_type>  <cell_vert 1> ...<cell_vert n>
4.  <num_comp for node data>  <size comp 1>  <size comp 2>...<size comp n>
5.  <node_comp_label 1> ,  <units_label 1>
    <node_comp_label 2> ,  <units_label 2>
         .
         .
         .
    <node_comp_label num_comp> ,  <units_label num_comp>
6.  <node_id 1>  <node_data 1> ... <node_data num_ndata>
    <node_id 2>  <node_data 1> ... <node_data num_ndata>
         .
         .
         .
    <node_id num_nodes>  <node_data 1> ... <node_data num_ndata>
7.  <num_comp for cell's data>  <size comp 1>  <size comp 2>...<size comp n>
    <cell-component-label 1> ,  <units-label 1>
```

```
    <cell-component-label 2> ,  <units-label 2>
        .
        .
        .
    <cell-component-label n> ,  <units-label n>
    <cell-id 1> <cell-data 1> ... <cell-data num_cdata>
    <cell-id 2> <cell-data 1> ... <cell-data num_cdata>
        .
        .
        .
    <cell-id num_cells> <cell-data 1> <cell-data num_cdata>
8.  <num_comp for model's data>  <size comp 1>  <size comp 2>...<size comp n>
    <model-component-label 1> ,  <units-label 1>
    <model-component-label 2> ,  <units-label 2>
       .
       .
       .
    <model-component-label n> ,  <units-label n>
    <model-id> <model-data 1> <model-data num_mdata>
```

The UCD structure and library will support either integer or character node-, cell-, and model-ids, (referred to in the library documentation as **names**). However, the **read ucd** module only accepts integer node-ids, cell-ids, and model-ids. This is shown in the example below. The ids do not have to be consecutively numbered.

Also note that, at present, most of the UCD modules do not make use of cell and model-based data, thus the input data examples all show "0" for <num-cdata> and <num-mdata>. User-written modules can use the UCD library to manipulate cell- and model-based data.

Example ASCII UCD File

The following is an example of a simple UCD file. This UCD structure has 8 nodes in 1 hexahedral cell. Associated with each node is a single scalar data value, making up one component that this person labels "stress," and specifies a "lb/in**2" unit label. There is no cell- or model-based data. See the "Unstructured Cell Data" appendix in the *Developer's Guide* for more examples.

```
#
# Simple AVS UCD File
#
8 1 1 0 0                    <--8 nodes, 1 cell, 1 component of node data
1  0.000   0.000   1.000     <--Node coordinates
2  1.000   0.000   1.000
3  1.000   1.000   1.000
4  0.000   1.000   1.000
5  0.000   0.000   0.000
6  1.000   0.000   0.000
7  1.000   1.000   0.000
8  0.000   1.000   0.000
1 1 hex 1 2 3 4 5 6 7 8   <--cell id, material id, cell type, cell vertices
1 1                        <--num data components, size of each component
stress, lb/in**2           <--Component name, units name
1  4999.9999               <--Data value for each node component
```

```
2  18749.9999
3  37500.0000
4  56250.0000
5  74999.9999
6  93750.0001
7  107500.0003
8  5000.0001
```

Programming Examples

The AVS release includes several sample programs, one in Fortran and threee in C, that deal with AVS UCD-format files. These sample programs can be used as models for your own modules.. The UCD library these samples call is documented in the "Unstructured Cell Data Library" appendix to the *AVS Developer's Guide.*

read_ucd.c

The file */usr/avs/examples/read_ucd.c* is a C program source. It reads either a binary or ASCII format UCD dataset (file suffix *.inp*), creating the UCD structure format used by the various UCD visualization modules. **Note:** ignore the binary section of this example. It reads AVS 3 format UCD binary files; a format that has changed with the AVS 4 release. The ASCII section is still valid.

gen_ucd.f

The file */usr/avs/examples/gen_ucd.f* is the Fortran source to a module that first generates its own scalar or vector hexahedral data, then writes it out as a UCD structure.

ucd_thresh.c

The file */usr/avs/examples/ucd_thresh.c* is the C source to the **ucd threshold** module. It is a filter that both reads an existing UCD structure and creates a new UCD structure as output. There are some extra variables declared in the header that the program does not actually use.

ucd_extract.c

The file */usr/avs/examples/ucd_extract.c* is the C source to the **ucd extract** module. It is a filter that reads an existing UCD structure, extracts various components from the data, and outputs a new UCD structure. As with *ucd_thresh.c*, there are some extra variables declared in the header that the module does not use.

Colormap Data

An AVS colormap is a data structure which implements a transfer function that assigns a color to each value between an upper and a lower bound. A colormap consists of:

- Four arrays of floating-point values, one each for hue, saturation, brightness, and opacity. Each value is normalized between 0.0 and 1.0 inclusive.
- An integer indicating the number of colors — this is the length of each of the four arrays.
- Floating-point lower and upper bounds that determine the resolution of the colormap. The lower bound is a data value that maps to the first element of each array. The upper bound is a data value that maps to the last element in each array. All intermediate data values are mapped to elements within the arrays.

The hue, saturation and brightness (HSB) color space can be thought of as an inverted cone:

- The **hue** axis runs circularly around the cone. Example hue values and corresponding hues are given below:

  0.00 = red
  0.16 = yellow
  0.33 = green
  0.50 = cyan
  0.66 = blue
  0.83 = magenta
- The **saturation** axis runs from the center of the cone (white) to its perimeter (fully saturated color). Example saturation values are:

  0.00 = white
  0.50 = partially saturated **hue**
  1.00 = fully saturated **hue**
- The **brightness** axis runs from the tip of the cone (black) to the base (white). Example brightness values are:

  0.00 = black
  0.50 = partially darkened **hue**
  1.00 = full intensity **hue**

The **opacity** value determines the amount of transparency the color has:

0.00 = completely transparent
0.50 = partially transparent
1.00 = completely opaque

AVS Module: generate colormap

The **generate colormap** module produces an AVS *colormap* data structure, for use by modules that transform input data into color values. This module also

allows the user to both read and write AVS-compatible colormaps. These *colormaps* are stored on disk as ASCII files, in the following format:

```
number_of_entries
hue  saturation  brightness  opacity
hue  saturation  brightness  opacity
hue  saturation  brightness  opacity
 ...
low_value high_value
```

All values in this file are in floating-point format, with the exception of **number_of_entries**, which is an integer. The hue, saturation, brightness, and opacity values are normalized to the range 0.0 - 1.0. For examples of files containing AVS colormaps, look at several of the files in the directory */usr/avs/data/colormap*.

Molecule Data Type

The molecule data type (MDT) consists of ten defined CHEM objects that can be combined hierarchically in linked lists to represent the structure of molecules. At the root of the hierarchy is the CHEMmolecule. A CHEMmolecule is a data structure containing the molecule's name, and a unit in angstroms or bohrs. Chained off of the CHEMmolecule is an arbitrarily long list of CHEMatoms, CHEMchemunits, CHEMquantums, and user-defined data.

A CHEMatom is a data structure containing the atom's name, a color to represent it, its x, y, z position in space within the molecule, and its radius. The connectivity of the CHEMatom to other atoms, and the nature of the chemical bond (none, single, double, triple, hydrogen, disulfide, etc.) is specified by a link to a CHEMcandb data structure. CHEMatoms also contain a provision for user-defined data. By default, this area specifies the atom's atomic number, weight, hybridization type and charge.

CHEMchemunits are used to represent chemical substructures in the CHEMmolecule. For example, if the CHEMmolecule were DNA, CHEMchemunit could specify the animo acids of which it is composed.

The CHEMquantum data structure stores quantum information about the parent molecule such as charge, number of electrons, alpha and beta spin, a basis set for gaussian functions, molecular orbit, numbers of shells, etc.

The molecule data type and the *libchem* library that manipulates it is described in its own document, the *AVS Chemistry Developer's Guide.*

Within the AVS release:

/usr/avs/examples/chemistry
Shows source code examples to chemistry modules.

/usr/avs/data/chemistry
Shows sample chemistry data files.

/usr/avs/chem_lib

Contains sample AVS modules that illustrate use of the chemistry data type. They are not intended as "production" modules. These modules are not in the Network Editor's list of default module libraries. To use the chemistry modules, use the Network Editor's **Read Module Library** function on the **Module Tools** menu to load the file */usr/avs/chem_lib/chemistry*

/usr/avs/networks/chemistry

Shows the sample chemistry modules at work in networks. These may be read into the Network Editor using the **Read Network** function on the **Network Tools** menu.

libchem, the molecule data type, and the contents of the example directories above are also referred to collectively as the "Chemistry Developer's Kit" (CDK).

AVS Module: Read structure file

Unlike the other AVS data types, no "standard" file format for the molecule data type has yet been defined. Data content and representations tend to differ more widely within the chemistry field than many other disciplines.

Nonetheless, the files */usr/avs/examples/chemistry/CHEMcon_r.c* and *CHEMcon_rf.f* are C and FORTRAN examples that read files containing AVS molecule data type information and produce MDTs as outputs. Example files in this format are found in */usr/avd/data/chemistry*.

CHAPTER 3

STARTING AVS

Introduction

This chapter describes:

- How to start an AVS session.
- The layout of the main AVS menus; starting and switching among subsystems.
- Interactive facilities for learning about AVS.
- How to get online help on using AVS.
- Command line, startup file, and environment variable options that affect how AVS runs.
- How to add applications to the AVS Applications submenu.

Note: The first time you execute a newly-installed copy of AVS you might see a notice explaining that AVS must be licensed before it will run. Either you or your system administrator should follow the instructions given in the product installation documentation to obtain an AVS license.

Platform Dependencies

AVS runs on a variety of vendor platforms. Each platform may have unique requirements for command line, startup file, or environment variable settings to make AVS run correctly and optimally on that platform. The authoritative source for how to start AVS on your specific platform is the release notes that accompany your copy of AVS. On some platforms, these release notes are available as on-line printable files in a */usr/avs/relnotes* directory.

In addition to the operating system requirements such as the correct version of the OS and graphics libraries, adequate memory and swap space, these are the AVS startup options that most often need adjusting:

Gamma
NetworkWindow
NoHW
ScreenSize
VisualType
Colors (occasionally needed)

There may be additional, platform-specific options. Be sure to consult your platform's release notes for specifics.

Normally, you run AVS from a display that is directly-connected to the graphics workstation or computer on which AVS executes. However, you may also be able to run AVS from any workstation or "X terminal" with color display hardware and an X11 server that supports at least an 8-plane "PseudoColor visual." In this configuration, AVS executes as a remote X Window System client on another system which runs AVS. Your workstation or X terminal simply acts as an input/output device, displaying the AVS interface and its output visualization windows, and sending your keyboard and mouse commands to AVS on the remote system. See the "AVS on Color X Servers" appendix for more information on this feature.

Controlling AVS Startup

Before starting AVS, make sure that the system environment variable DISPLAY is set to indicate the display at which you are working. The X Window System uses the DISPLAY environment variable to tell it which display to create its windows upon.

The basic command to start AVS is simple:

```
avs
```

This is usually all that is necessary to start AVS. Beyond this simplest case, there are many options that you can use to modify how AVS behaves.

Three things can affect how AVS starts. They are listed here in their order of precedence:

1. Command line options.
2. The *.avsrc* startup file. The startup file contains keyword-value pairs. AVS always reads the system default startup file in */usr/avs/runtime/avsrc* first. Users may override or supplement these system default options with a personal *.avsrc* file. AVS will look for a personal startup file in *./.avsrc* (in the current directory), then *$HOME/.avsrc* (in your HOME directory). It uses the first of these two *.avsrc* files that it finds.
3. Environment variables.

Table 3-1 lists all AVS command line options, *.avsrc* startup file keywords, and environment variables. Each is explained in detail in the "AVS Command Line Options," "AVS .avsrc Startup File," and "AVS Environment variables" sec-

tions toward the end of this chapter. There may be additional vendor or platform-specific options, *.avsrc* keywords, and environment variables..

Table 3-1 AVS Command Line Options, .avsrc Keywords, and Environment Variables

Command Line Option	.avsrc Option	Environment Variable
class		DISPLAYCLASS
cli		
compile_library		
data	DataDirectory	
dials	DialDevice	DIALS
display		DISPLAY
gamma	Gamma	
geometry		
graph		
image		
library	ModuleLibraries	
modules		
netdir	NetworkDirectory	
network		
nodmc	DirectModuleCommunication	
nohw	NoHW	
parallel		
path	Path	
reindex		
renderer	Renderer	
separate		
server		
shm / noshm	SharedMemory	
size	ScreenSize	
spaceball	SpaceballDevice	SPACEBALL
timer		
version		
usage		
	Applications	
	BoundingBox	
	Colors	
	DisplayPixmapWindow	
	GridSize	
	HelpPath	AVS_HELP_PATH
	Hosts	
	ImageAutomagnify	
	ImageScrollbars	
	ModulePanelHeight	
	NetworkWindow	
	NetWriteAllParms	
	PrintNetwork	
	ReadOnlySharedMemory	
	SaveMessageLog	
	StackSelector	
	VisualType	
	WindowMgr	
	XWarpPtr	
		AVS_ADAPT_TABLE

Table 3-1 AVS Command Line Options, .avsrc Keywords, and Environment Variables (Continued)

Command Line Option	.avsrc Option	Environment Variable
		AVS_GEOM_WRITE_V30 AVS_MEM_CHECK AVS_MEM_HISTORY AVS_MEM_VERBOSE AVS_MG_TROFF EDITOR

The Main Menu: Basic Interface

When you start AVS, the main menu appears within a control panel along the left edge of the screen (see Figure 3-1).

To enter a subsystem, use any mouse button to click on the subsystem's menu button.

The **AVS Applications** selection produces an additional menu of AVS applications. The list will vary depending upon what applications are installed with your system. Base AVS includes two applications, the **AVS Demo** suite, and the **Data Viewer**.

You or your AVS system administrator can add applications to this Applications menu. See the "Adding to the Applications Menu" section later in this chapter.

Subsystem Control Panels

Each of the subsystems has its own control panel (usually, along the left edge of the screen). The control panel is made up of a series of buttons that invoke various subsystem functions. Click with any mouse button to select any control panel function.

When you click the **Close** button at the top of a subsystem's control panel, the control panel is "unmapped" from the screen, rather as though you had iconified the control panel (but without the icon). If you re-enter the subsystem at a later time, its control panel reappears in the same state that you left it. Sub-

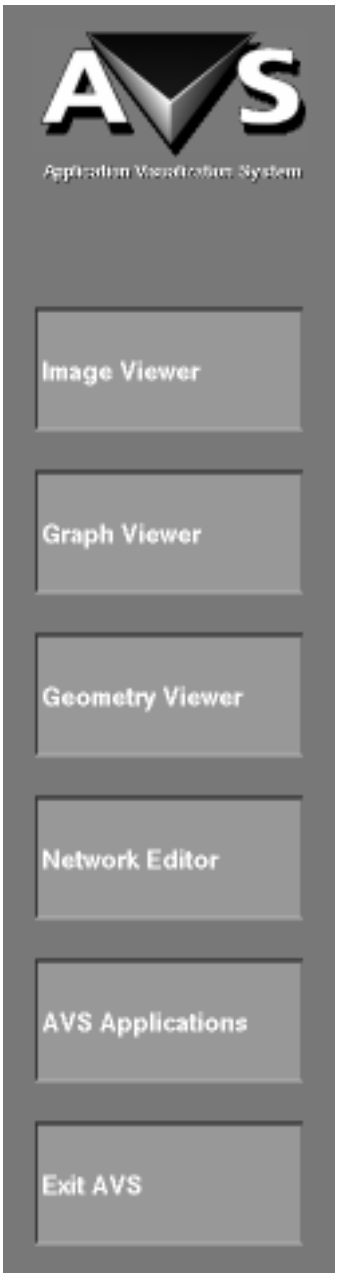


Figure 3-1: AVS Main Menu

systems are never truly "exited," their control panels just appear and disappear.

The exception to this is the Network Editor. It has a true **Exit** button. You must save your work before leaving the Network Editor with functions such as the **Write Network** button.

Subsystem control panels are like any other window on the screen. You can move and resize them. Often, you want more than one control panel on the screen at a time (e.g., both the Network Editor control panel and the Geometry Viewer control panel). Just use your window manager to move the control panels around the screen to more convenient locations.

If all subsystem control panels are closed, or if they are moved away from their original location, the AVS main menu reappears.

Switching Among the Subsystems: Data Viewers Button

You can switch from any main menu subsystem to any other subsystem. At the top of each subsystem's control panel is a **Data Viewers** button (see Figure 3-2). Press *and hold down* any mouse button over **Data Viewers**. A pop-up menu appears listing the other subsystems. Roll the mouse cursor down to the subsystem you want and release the mouse button. Its control panel will appear.

Figure 3-2: Data Viewers Button

Properly-speaking, you are not switching among subsystems so much as you are causing control panels to be mapped and unmapped from the screen. When a subsystem's control panel reappears, it is always the same position on the screen where it was **Close**d.

Each subsystem has only one control panel associated with it. Hitting **Data Viewers**, then **Graph Viewer** three times does not produce three Graph Viewer control panels; it just maps the same panel three times in succession.

The exception to this is the Network Editor. To get to the Network Editor, you must press the main menu's **Network Editor** button. This may involve closing other control panels or moving them with the window manager so that the original AVS main menu is no longer obscured.

As noted above, the Network Editor is also the only subsystem with a real **Exit** button that deletes the current state of your Network Editor work. The Network Editor asks you to confirm that you want your work deleted before it exits, giving you a chance to save your work.

Cancelling Operations

In general, AVS has no cancel function. If you press a button, or make a manipulation in the Geometry Viewer, Image Viewer, or Graph Viewer, then you must wait for the results. While you are waiting, *do not* click on other buttons or try to use the mouse buttons to move objects around just to see if the interface is alive. AVS will queue these operations. This last is particularly critical when you are dealing with large datasets or using AVS on a lower-performance platform such as an "X terminal."

The one exception is in the Network Editor. You can "hammer" a running module by dragging its icon to the Hammer icon in the lower right corner with the left mouse. The module's process will exit. (Some modules cannot be hammered. See the "Cancelling an Operation" section in the "Network Editor" chapter before using this facility.)

Be aware of the size of your data and the computational implications of operations you request. Many AVS modules and subsystems will warn you if an operation is going to take a long time and give you the chance to change your mind. You can use filter modules such as **crop** and **downsize** in AVS networks to reduce the size of field datasets.

Learning AVS

In addition to the documentation, AVS provides two interactive facilities that you can use to familiarize yourself with the system and the visualization process. Both are accessed from the main **AVS Applications** menu.

AVS Demo Suite

The **AVS Demo** suite is a pull-down menu interface that accesses a variety of pre-written Command Language Interpreter scripts that illustrate most aspects of the AVS interface.

Control
Provides the main controls for the **Demo** suite system.

General AVS
Contains scripts that illustratate the capabilities of the Network Editor, Geometry Viewer, Image Viewer, and Graph Viewer subsystems.

The **Modules** selection runs scripts that illustrate how most of the supplied AVS modules are used in visualization networks. The scripts are named after their key component module. They are the same scripts referred to at the bottom of each module's "man page" and in the AVS *Module Reference* manual.

Modules is thus similar to the **Help Demos** facility described below in "Using On-Line Help."

VisualizationTechniques
Creates networks of modules that illustrate the major scientific visualization techniques such as image processing and volume visualization.

Markets
Creates networks of modules that would typically be used to explore data in various disciplines such as medical imaging and computational fluid dynamics.

The **AVS Demo** suite is described in detail in the AVS *Tutorial Guide.*

The Data Viewer

The **Data Viewer** provides a simplified, pulldown menu interface for building visualization networks. Like the **Demo** suite, it is a useful tool for the novice user learning basic scientific visualization techniques and terminology.

The **Data Viewer** differs from the **Demo** suite in two key ways:

- The **Demo** suite uses pre-defined networks. As its name implies, it is meant to be *watched.* The **Data Viewer** allows you to dynamically construct your own visualization networks. You can add and subtract individual modules and techniques from the network to see their effects, gaining a better understanding of how network pieces are combined to create meaningful wholes. The **Data Viewer** facilitates a new user's active experimentation with the system.
- The **Demo** suite's scripts use pre-defined AVS sample datasets. The **Data Viewer** has no such restrictions. You can use it to become familiar with AVS using your own datasets.

The **Data Viewer** is described in detail in the AVS *Applications Guide.*

Using On-Line Help

At all times during an AVS session, on-line help is available. Help takes several forms.

Help Buttons

All of AVS's control panels include a **Help** button at the top. Clicking this button causes a **Help Panel** window to appear (see Figure 3-3).

The Help Panel has several areas of interest:

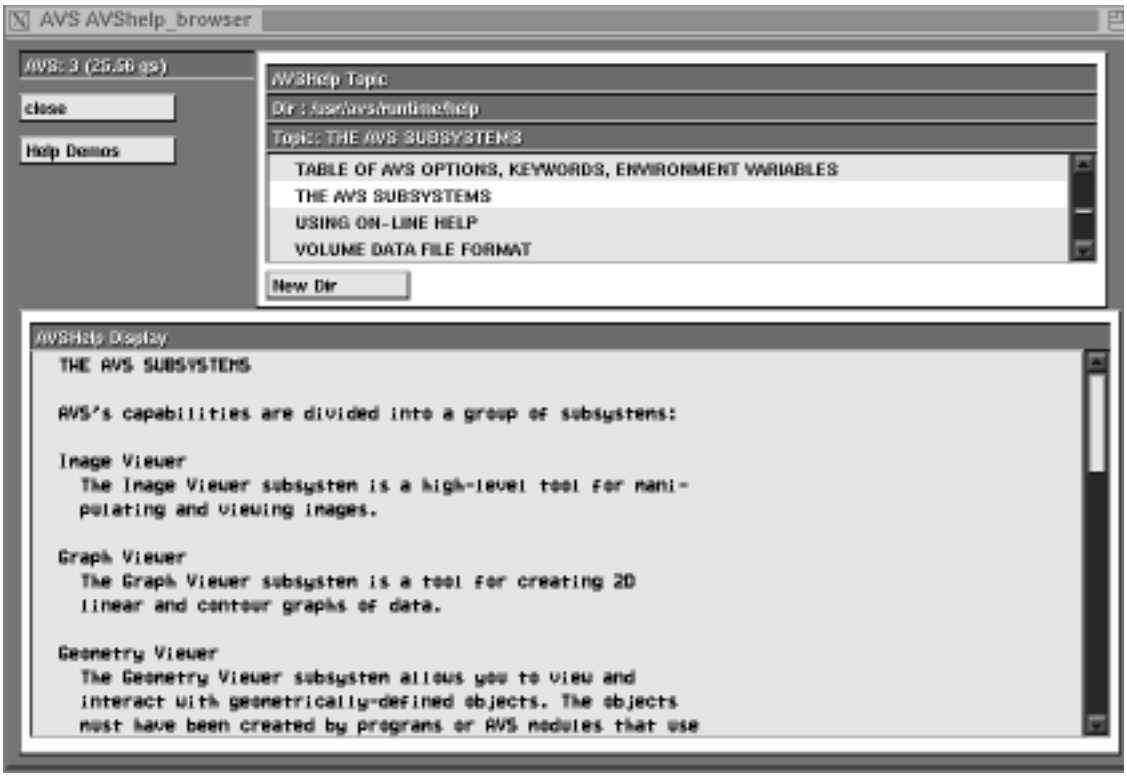


Figure 3-3: The Help Panel

Help Topic Browser

The **Help Topic Browser** at the top of the panel displays a list of help topics. To get help on a particular topic, just click on it. A text file is loaded into the browser's text viewing area.

Help Text Browser

The **Help Text Browser** is a scrolling window of ASCII text.

Help Demos

The **Help Demos** button at the upper left of the Help panel pops-up a scrolling browser of AVS "script" files. (Figure 3-4.) These files, written in the Command Language Interpreter (CLI) language, automatically execute parts of AVS. They can be used to dynamically illustrate the interface and visualization modules.

By default, the **Help Demo** browser comes up showing the sample module and network scripts in */usr/avs/demo/man_scripts*. Most of these are constructed from the example networks on the module man pages. (These are the same sample scripts as the AVS **Demo** suite's **Modules** selection.).

Clicking on a script causes it to bring up the Network Editor, load a sample network, and run some interesting data and parameter settings through it. When the script is finished, it leaves the network in the Network Editor so that you can experiment with it yourself. The next script

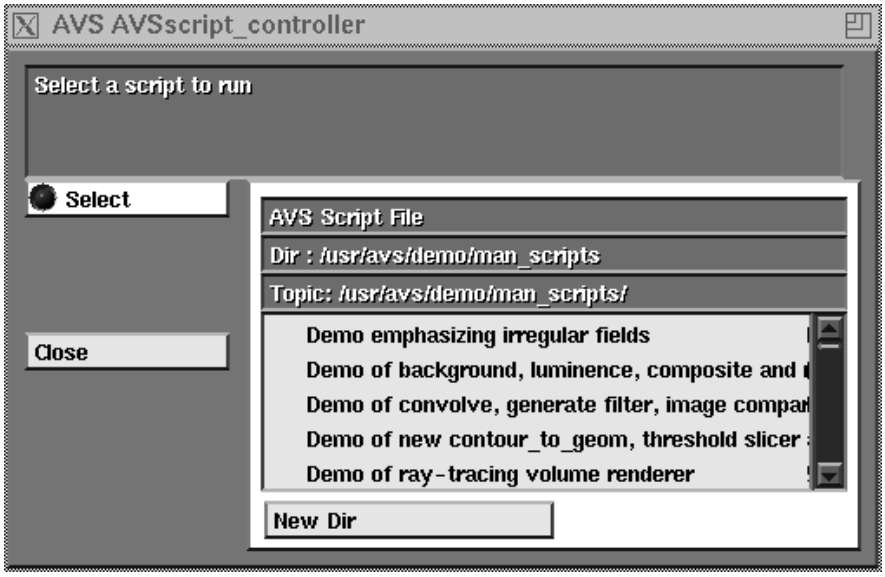


Figure 3-4: Help Demo Browser

read in will clear the previous network. You must get rid of the last network manually, either by hitting **Clear Network**, or by simply **Exit**ing the Network Editor.

While a script is running, you may cause it to **Pause**, **Continue**, or **Abort**.

Other demonstration scripts exist in */usr/avs/demo/image_viewer* and */usr/avs/demo/examples*. The *examples* scripts require that the */usr/avs/examples* modules that they illustrate must have first been compiled. The */usr/avs/examples/README* file has instructions for doing this.

To use the scrollbars present on all the Help panel browsers:

- The left mouse button scrolls upward.
- The effect of the middle button depends on exactly where the cursor is:
  - **In the arrow box at the top**. Click to scroll to the very top of the help text.
  - **In the elevator shaft**. Click and hold down the button to grab the elevator bar. Moving the bar up or down causes the help text to scroll accordingly.
  - **In the arrow box at the bottom**. Click to scroll to the very bottom of the help text.
- The right mouse button scrolls downward.

You can change the size of the viewing area by using the X window manager to make the entire Help panel window larger or smaller. You can also move the window using the window manager.

Click on as many topics as you like. When you're done, click the **Close** button to close the Help panel window.

Red entries in a help browser indicate subdirectories that contain additional help screens. You'll often see the red entry "../(help)" at the top of the Help Topic Browser list. This indicates the parent directory, */usr/avs/runtime/help*, which contains a group of help screens that provide overall AVS orientation.

Module Editor

In the Network Editor, each computational module in AVS is represented on-screen by an icon. Clicking on the small square at the right side of the icon with the middle or right mouse button opens a Module Editor window, which displays information about the module: a capsule description, its inputs and outputs, etc. Clicking on the **Show Module Documentation** box pops up a Help Text Browser like the one described above, displaying the complete manual page for that module.

Shell-Level Help

The module manual pages may also be available through the system shell command *man*(1). For this to be true, the */usr/avs/runtime/help/modules* directory must have been added to the *man* command's search path.

File Browsers and Dialog Typein Panels

In addition to the subsystem control panels, AVS uses a number of different kinds of interaction **widgets**. By far the most common are various **browsers**. These are new windows that pop-up on the screen when you press a control panel button, showing you a selection of choices, like the Help Browsers described above. If the length of the list exceeds the size of the browser window, you can use the scrollbars at the right of the browser to see all the choices. Most browsers are "sticky," that is, they remain on the screen until you explicitly remove them by pressing their **Close** button.

The most common browser is the **File Browser** (see Figure 3-5). File browsers are associated with each subsystem's "Read" function (e.g., **Read Object**, **Read Image**, **Read AVS Plot File**). They also appear on all of the **read** *datatype* modules.

The entries in a file browser are color-coded: black entries are files; red entries are subdirectories (the topmost red entry is usually the parent directory). To select one of the entries, click on it with any mouse button. Selecting a directory entry changes the working directory, causing filenames in that directory to be displayed, along with the names of any subdirectories.

Since a directory might contain a large number of entries, a file browser has a scroll bar along its right edge. Clicking inside the scroll bar makes additional entries appear:

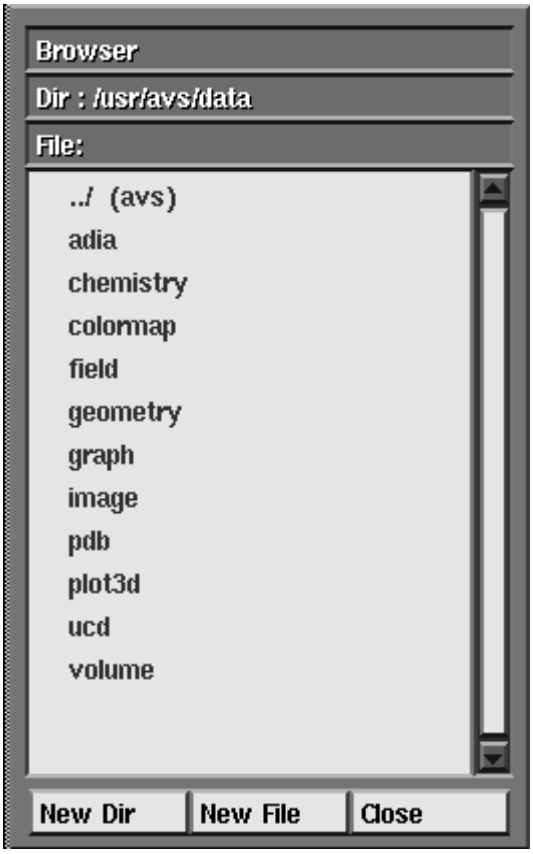


Figure 3-5: File Browser Widget

- The left mouse button scrolls upward.
- The effect of the middle button depends on exactly where the cursor is:
  - **In the arrow box at the top**. Click to scroll the list to the very top.
  - **In the elevator shaft**. Click and hold down the button to grab the elevator bar. Moving the bar up or down causes the list to scroll accordingly.
  - **In the arrow box at the bottom**. Click to scroll the list to the very bottom.
- The right mouse button scrolls downward.

A file browser has these buttons at the bottom:

New Dir

Pops up a dialog typein panel (Figure 3-6) in which you can type the name of another directory (full pathname or path relative to the current directory). File browsers default to showing you the contents of the */usr/avs/data* directory. (You can change this with the **DataDirectory** *.avsrc* file option discussed below.)

Be sure the mouse cursor is within the dialog typein box (but not on the **OK** or **Cancel** button) before you start typing the directory name. When

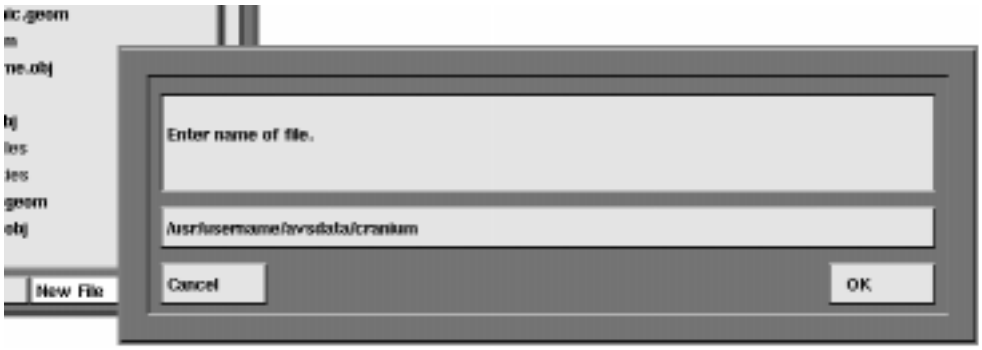


Figure 3-6: Dialog Typein Panel: Entering a Filename

you click the **OK** button in the dialog typeinbox, or press the **Return** key, or move the cursor outside the panel, the directory whose name you've typed becomes current, and its filenames are displayed in the browser window.

Should you inadvertently give a filename, it will select that filename as though you had used **New File** *and* select the directory it is in.

Use **Backspace** to erase the last character or **Ctrl-U** to erase the entire name. If you change your mind altogether, click the **Cancel** button.

New File
Pops up a dialog box that works the same way as the **New Dir** box. This allows you to specify the file to be processed, either with a full pathname or a name relative to the current directory. Should you give a directory rather than a filename, it will change to the directory.

Close
(not always present) This removes the File Browser widget from the screen.

You normally have to manually move your mouse cursor into a dialog typein panel when it appears. This follows X Window System interface conventions which say that a program should never "warp" a mouse cursor for a user. You can override this and have AVS move the cursor automatically into dialog typein panels and other dialog widgets that require a response by setting **XWarpPtr on** in your *.avsrc* startup file.

The other AVS control widgets, most of which are associated with AVS modules, are discussed in the "Network Editor" chapter.

Exiting AVS: Saving Work

To leave AVS, bring back the AVS Main Menu. This may involve moving and/or closing subsystem control panels or exiting the Network Editor. Then

press **Exit AVS**. AVS brings up a dialog box, asking you to confirm that you wish to exit.

AVS does not "save state" from one session to another. However, the Image Viewer, Geometry Viewer, and the Network Editor each have commands that perform a similar function. In the Image Viewer, you can press **Save Scene** under the **Views** menu; in the Geometry Viewer, you can **Save Scene** under the **Cameras** menu; and in the Network Editor, you can **Write Network** under the **Network Tools** menu. Each of these uses the AVS Command Language Interpreter (CLI) to save a snapshot of the current state of an individual image or geometry scene, or AVS network. You can read this snapshot in the next time you use AVS. Check with each subsystem's chapter to find out what each saves.

AVS Command-Line Options

There are quite a few options that you can use when issuing the **avs** command. All option keywords begin with a hyphen (e.g. **-data**). In many cases, the keyword is followed by an additional word (e.g. a directory name). You must separate the keyword and the additional word with whitespace (SPACE and / or TAB characters).

All options keywords can be abbreviated, as long as there is no ambiguity. For example, **-data** can be abbreviated to **-da**. But you cannot abbreviate it to **-d**, since this might indicate either **-data** or **-display**.

In several cases, you can use an entry in the AVS *.avsrc* startup file as an alternative to a command-line option. For example, a **DataDirectory** entry in the startup file is equivalent to a **-data** option. See the next section for details on the startup file.

-class *string*

(startup file equivalent: none) This is the command line option equivalent of the DISPLAYCLASS environment variable. You can use it to make AVS behave in different ways when it is started from different types of display hardware. **-class** has two effects:

1. An *Xdefaults* file specifies the "look" of the AVS interface; what shades of grey are used for command buttons, what fonts to use, whether the background is "stippled" or a flat color, etc. When **-class** *string* is given, AVS does not use the default */usr/avs/runtime/avs.Xdefaults* file. Instead, it looks for an *Xdefaults.**string*** file in the */usr/avs/runtime* directory and uses it. At present, the only alternate X defaults file supplied is *Xdefaults.X*.
2. If such a file is present, it will use an alternate startup file, */usr/avs/runtime/avsrc.**string***. Otherwise, it uses */usr/avs/runtime/avsrc*. It will also look for a *.avsrc.**string*** file in the current directory, then your HOME directory and use it instead of your usual *.avsrc* file.

   -class is often used when running AVS from an "X terminal." See the full discussion in the "AVS on Color X Servers" appendix.

-cli *[any CLI command]*

(startup file equivalent: none) Run AVS with the Command Language Interpreter functioning in the terminal emulator window from which AVS was invoked. **-cli** accepts an optional initial command string, which must be enclosed in quotes, e.g., *-cli "script -play name.scr"*. This command string will be executed after AVS starts up. See the chapter on the "Command Language Interpreter" in the *AVS Developer's Guide* for details.

-compile_library *source_filespec compiled_filespec*

(startup file equivalent: none) This is a utility for maintaining module libraries whose component modules are changing. It follows a "source module library" vs "compiled module library" paradigm. Specifically, **-compile_library** takes the *source_filespec* to be an AVS module library file containing a list of **file** commands followed by the name of a module binary file. It executes each module listed in order to extract the module description information. From this, it generates *compiled_filespec* as an AVS module library file containing the description information necessary to load the module into the Network Editor's Palette quickly without actually executing the module binary.

See the "Constructing a Module Library" discussion in the "Advanced-Network Editor" chapter for more information.

-data *directory*

(startup file equivalent: **DataDirectory**) Specifies the directory in which all subsystem data input file browsers, including the Image Viewer, the Graph Viewer, the Geometry Viewer, and the data input modules in the Network Editor, will initially look for data files (files used as input to computational modules). This is the major tool for redirecting AVS's default data input focus off the sample data files provided in */usr/avs/data* and onto your own data files.

The default data directory is */usr/avs/data*.

-dials *devicefilespec*

(startup file equivalent: **DialDevice**) Specifies the serial communications port to which a dialbox device is attached (e.g. */dev/tty2*). If **-dials** is present, AVS automatically connects the dialbox dials to the Geometry Viewer's rotation, translation, and scaling transformations. You must know which serial communications port your dialbox is connected to. This argument also corresponds to the environment variable DIALS. Dialboxes are not supported on all platforms.

-display *host:server.screen*

(startup file equivalent: none) Specifies the X Window System display on which AVS is to display. This overrides the current setting of the DISPLAY environment variable.

-gamma *number*

(startup file equivalent: **Gamma**) Controls the brightness of the display for all AVS windows except Geometry Viewer output windows produced with a hardware renderer. The default varies from platform to platform.

Values between 1.7 to 2.2 are good starting points for experimentation. Higher real values produce a lighter display.

-geometry *[geom-option(s)]*
(startup file equivalent: none) Automatically invokes the Geometry Viewer subsystem at startup. There will be no **Data Viewers** button to access other subsystems. If you use this option, it must be the *last* option on the command line, followed only by the options listed below that are specific to this subsystem. All other options that follow **-geometry** will be ignored.

-scene *scene-file.scene* or *geomcli-file.scr*
(startup file equivalent: none) This option executes the Geometry Viewer's **Read Scene** function, using the file *scene-file.scene* or *geomcli-file.scr*, depending upon the setting of the **AVS_GEOM_WRITE_V30** environment variable.

-filter *pathname*
(startup file equivalent: none) Specifies *pathname* as the directory to search for geometry conversion utilities, named *..._to_geom*. See the "Importing Data Into AVS" chapter.

The default directory for these programs is */usr/avs/bin*.

-defaults *filename*
(startup file equivalent: none) Specifies a Geometry Viewer defaults file. The format of this file is described in the "Geometry Viewer Script Language" appendix.

-geometry *Xgeometry*
(startup file equivalent: none) Specifies an X Window System geometry (e.g. 500x500-5-5) for the initial window created by the Geometry Viewer.

-noroll
Turns off track rolling. Track rolling occurs when you perform a transformation and release the mouse button while the mouse is still moving. This "flings" the transformable, causing it to continue in motion.

-usage
(startup file equivalent: none) Displays a list of Geometry Viewer startup options.

-graph
Automatically invokes the AVS Graph Viewer at system startup. There will be no **Data Viewers** button to access other subsystems.

-image
Automatically invokes the AVS Image Viewer at system startup. There will be no **Data Viewers** button to access other subsystems.

-library *filespec*
(startup file equivalent: **ModuleLibraries**) Specifies which AVS module library file to load into the Network Editor at system startup. Module li-

brary files are ASCII files describing sets of modules. */usr/avs/avs_library/Supported* is an example. This is the major tool that allows you to load your own sets of modules—either modules you've written yourself or subsets of the supplied modules that you have customized to your needs—instead of always relying on the system default Supported and Unsupported module libraries specified in the */usr/avs/runtime/avsrc* file.

To load more than one module library, use multiple pairs of **-library** *filespec* options.

It is equivalent to using the Network Editor's **Read Module Library** function.

-library causes AVS to load *only* the libraries specified on the command line.

-modules *directory* or *filename*

(startup file equivalent: none) Specifies a directory or file in which the AVS Network Editor will initially look for executable modules. All executable files in the directory are examined to determine whether they contain one or more modules. Those that do are added to the default module library modules in the Network Editor's module Palette.

-modules differs from **-library** above in that it loads *binary* module files, not ASCII module *library* files. It is slower to load modules as binary files rather than libraries.

You can use more than one **-modules** options to specify multiple individual module binaries, or to have AVS search through multiple directories for modules. This is the main tool for loading individual modules (perhaps modules that you are debugging) that you have not yet formalized into a module library. It is equivalent to the Network Editor's **Read Module(s)** function. It cannot be used to read remote modules.

The default modules directory is */usr/avs/avs_library*.

-netdir *directory*

(startup file equivalent: **NetworkDirectory**) Specifies the directory in which the AVS Network Editor subsystem initially will look for network files (**Read Network** and **Write Network** functions). This is the tool to use to redirect AVS's default network focus away from the samples provided in */usr/avs/networks* and onto your own network files.

The default network directory is */usr/avs/networks*.

-network *network-file*

(startup file equivalent: none) Starts AVS and brings up the Network Editor's module control panel with the controls for the network displayed. The full Network Editor subsystem is not displayed or accessible. This is one way to make an individual production network available to a user.

-nodmc

(startup file equivalent: **DirectModuleCommunication 0**) Turns off the default direct module-to-module communication. This is useful if you

want to perform timing tests to compare network execution speed with / without direct module-to-module communication.

-nohw

(startup file equivalent: **NoHW 1**) Tells the AVS Geometry Viewer to not initialize any hardware renderers. Without a hardware renderer, the AVS Geometry Viewer will use a software renderer to create its 3D scenes instead of the platform's native graphics facilities. **-nohw** is used when you are running AVS as a remote X client on a different hardware platform or when you are using an "X terminal." The software renderer creates an X image rendering of the 3D scene and ships only the image to the local X server for display rather than a stream of rendering commands that may not be understood by the local system.

-nohw is equivalent to the obsolete *-swrender*.

-parallel *n*

(startup file equivalent: none) Sets the maximum number of module processes that will attempt to execute in parallel at any one time. The default is 1 (no parallelization.) You should set this figure intelligently for the system(s) that you are running on. If two processors are available (a two-processor system, or a local and a remote system) then this figure can reasonably be set to 2. If you give a value that exceeds the number of processors available, the underlying operating systems will serialize the processes. There is no inherent upper limit to the *n* parameter.

Modules must be in separate processes to execute in parallel. Most modules supplied with AVS are combined into a single executable that runs as a single process. Thus, they will not run in parallel unless they are divided into separate processes. This may be done wholesale with the **-separate** option (which may greatly increase memory utilization), or precisely using the Network Editor's module group editing facility. See the discussion on parallel module execution in the "Advanced Network Editor" chapter for more information.

-path *directory*

(startup file equivalent: **Path**) Specifies the directory tree in which AVS itself is installed.

The default path is */usr/avs*. If you specify another path, then the default data directory and network directory are modified accordingly. For example:

If:	path	= */usr/local/avs*
Then:	data directory	= */usr/local/avs/data*
	network directory	= */usr/local/avs/networks*

This option is also useful to switch between multiple versions of AVS (for example, a test release and a production release).

-reindex

(startup file equivalent: none) This option creates AVS help system *.topics* files. It does not start an AVS session. It is useful if you are creating help files for applications that you want to be accessible through the AVS help

system. See the appendix on creating help files in the *AVS Developer's Guide* for more information.

-renderer

(startup file equivalent: **Renderer** *"string"*) Specifies which renderer will be the default selected in the Geometry Viewer when a camera window is first created. *"string"* is the literal name found on the renderer buttons under the Geometry Viewer's **Cameras** menu, usually either "Software Renderer" or "Hardware Renderer", though other strings are possible. It must match exactly, in spelling, case, and spacing. The double quote marks must be present. Where there is a hardware renderer available, **-renderer** defaults to "Hardware Renderer". If the user specified **-nohw**, then only one renderer is available, the software renderer, and this option is ignored.

-separate

(startup file equivalent: none) This option disables AVS's multiple modules in one process feature. It forces each module to execute as a separate process, whether or not it is combined in an executable with other modules. The option is primarily useful for debugging, or when parallel module execution is desired. (In this last case, it is better to not use **-separate** on a production basis, since it usually increases memory utilization. Instead, individually divide modules into different executables using the Network Editor's module process group editing facility.) See the section on "Multiple Modules in a Single Process" in the *AVS Developer's Guide*.

-server

(startup file equivalent: none) This option opens a connection that an external process can use to connect to AVS and exchange with it a stream of Command Language Interpreter (CLI) commands and their output. See the chapter on the CLI in the *AVS Developer's Guide* for details.

-shm/noshm

(startup file equivalent: **SharedMemory** *on/off*) This turns the AVS shared memory option on and off. When shared memory is on, AVS keeps only one copy of AVS field and UCD data that all modules in a network share. (GEOM-format data and pixmaps do not use shared memory.) This improves performance by saving memory and processor time. **-noshm** can disable shared memory if, for example, AVS's use of the finite shared memory area is interfering with other applications. On most systems, shared memory is on by default. (Note: shared memory may not be implemented on all systems. See your AVS release notes.)

-size *XDIMxYDIM*

(startup file equivalent: **ScreenSize**) Specifies size, in pixels, to use for AVS's virtual display screen size. AVS will automatically resize its interface to fit into the virtual screen. You could use this to confine AVS to run within one section of your screen instead of across the whole screen. The aspect ratio should be 5x4 for proper results.

-spaceball *devicefilespec*
(startup file equivalent: **SpaceballDevice**) Specifies the serial communications port to which a Spaceball device is attached (e.g. */dev/tty2*). If **-spaceball** is present, AVS automatically connects the Spaceball device to the Geometry Viewer's rotation, translation, and scaling transformations. You must know which serial communications port your spaceball is connected to. This entry also corresponds to the environment variable SPACEBALL. Spaceballs may not be supported on all systems.

-timer
(startup file equivalent: none) Writes Geometry Viewer performance data to *stderr*. This should be used in conjunction with the **Object Info** panel to display the number of polygons being rendered. To get the measurement, use track rolling to set the object in continuous motion (middle mouse button to rotate, release mouse button while mouse is still moving, thereby "flinging" the object into continuous motion). Wait several seconds (the longer, the more accurate), then press any mouse button in the window to stop the object. Minimize mouse movements while the measurement is being taken. The measurement looks like:

73 frames in 6.632989 seconds for 11.005596 FPS

FPS stands for "frames per second." By convention, the "standard unit" is */usr/avs/data/teapot.geom*, in the default-sized window, rendered with the default gouraud shading, with no additional rendering options (color, shading, etc.). In this case, FPS can be referred to as TPS ("teapots per second").

-version
Displays the AVS version number. (Does not start an AVS session.)

-usage
Displays a usage message for AVS. No AVS session is started.

AVS .avsrc Startup File

When it begins execution, AVS searches for a *startup file*, which specifies such things as which module libraries to load, the locations of various directories, how big to make the AVS interface, etc.

AVS always first reads the system default startup file in */usr/avs/runtime/avsrc*. Users may override or supplement the options in the system startup file with a personal *.avsrc* file. AVS looks for user *.avsrc* files in the order listed, using the first that it finds:

./.avsrc	(current directory)
$HOME/.avsrc	(home directory)

You can copy the system default */usr/avs/runtime/avsrc* file to your HOME directory or other directory, modify it according to your needs and preferences, and rename it with the "." prefix.

If you give the **-class X** command option, or set the DISPLAYCLASS X environment variable, AVS will look for a different startup file: */usr/avs/runtime/avsrc.X*. If this file is not present, it will use the standard */usr/avs/runtime/avsrc* startup file. In the same manner as the normal startup procedure, AVS will also look for a personal *.avsrc.X* file in the current directory, then your HOME directory. This file is used to customize AVS when you are running it from an "X terminal." See the "AVS on Color X Servers" appendix.

.avsrc Startup File Format

Each line of the AVS startup file consists of keyword-value pair, with whitespace separating the keyword and the value. For example:

```
ModuleLibraries      /usr/avs/avs_library/Supported  /usr/johnp/avs/modules/Modlib
NetworkWindow        867x567+407+2
NetworkDirectory     /usr/johnp/avs/nets
DataDirectory        /usr/johnp/avs/data
DialDevice           /dev/tty02
```

Often, the keyword corresponds to one of the command line options described in the preceding section. If you use a command-line option, it overrides the specification, if any, in the startup file.

.avsrc Startup File Keywords

The AVS startup file keywords are listed below. Where startup file keywords have command line equivalents, see the command line description above for the most complete discussion of the feature.

Applications *filespec*

(command line equivalent: none) Causes AVS to use a file other than */usr/avs/runtime/AVS.applns* to build the large Applications menu. This is how a user would create their his/her set of application networks and have them accessible from AVS's Applications menu without modifying the central system file. If a simple filename is given rather than an absolute file and pathname, AVS will look for the file in the directory defined by **Path** (by default, */usr/avs*).

BoundingBox *switch*

(command line equivalent: none) If **BoundingBox 1** is set, then the AVS Image Viewer and Geometry Viewer will come up with their **Bounding Box** control already turned on. A "bounding box" is a less compute-intensive style of moving geometric objects and Image Viewer subimages. Instead of moving the object "real time," it only moves a wirebox representation of the object. Only when you release the mouse button is the object/subimage rendered at its new location. **BoundingBox** is most useful when you are using AVS on lower performance graphics systems, with the software renderer, or from an "X terminal." **BoundingBox** is usually off by default.

Colors *r g b gray*

(command line equivalent: none) This option controls how many cells of a *system* colormap AVS will attempt to allocate to itself when it starts. *r g b gray* represent numbers for red, green, blue, and gray. This is primarily intended for people who are using AVS from an "X terminal" or Pseudo-Color workstation that objects to the number of colormap cells that AVS tries to allocate for itself.

Colors is also used to increase the number of gray colormap cells available from the default 22 (on PseudoColor systems). You might need this, for example, if you are doing medical image processing where many fine distinctions in gray shades are required.

See the "AVS on Color X Servers" appendix for more information on how **Colors** works.

DataDirectory *directory*

(command-line equivalent: **-data**) Specifies the directory in which the various AVS data input file browsers used in the subsystems (Image Viewer, Graph Viewer, and Geometry Viewer) and Network Editor modules "read data" modules (**read field**, **read geometry**, etc.) initially will look for data files. This is the main tool to refocus AVS's data input attention off the sample data files in */usr/avs/data* and onto your own data files.

DialDevice *devicefilespec*

(command-line equivalent: **-dials**) Specifies *devicefilespec* as the serial communications port to which a dialbox device is attached (e.g. */dev/tty1*). If **DialDevice** is specified, AVS automatically connects the dialbox dials to the Geometry Viewer's rotate, translate, and scale transformations. This entry also corresponds to the environment variable DIALS. Dialboxes may not be supported on all platforms.

DirectModuleCommunication *switch*

(command line equivalent: **-nodmc**) Turns direct module-to-module communication on and off. This is useful if you want to perform timing tests to compare network execution speed with/without direct module-to-module communication. Direct module-to-module communication is on by default.

DisplayPixmapWindow *Xgeometry*

(command line equivalent: none) Controls the default X Window System geometry (size and position) of the **display pixmap** module's window.

Gamma *number*

(command line equivalent: **-gamma**) Controls the brightness of the display for all AVS windows except Geometry Viewer output windows produced with a hardware renderer. The default varies from platform to platform. Values between 1.7 and 2.2 are good starting points for experimentation. Higher real values produce a lighter display.

GridSize *n*

Controls the size in pixels of the Layout Editor's alignment squares when **Snap to Grid** is switched on. The default is 10.

HelpPath *directory ...*

(command line equivalent: none) Expands the list of directories that AVS will search to find a module's documentation when you click **Show Module Documentation** in the module's Module Editor window. This is useful when you are using modules other than the set provided with AVS. For the format of the Help path, see the "On-Line Help" appendix of the AVS *Developer's Guide*.

Hosts *fullfilespec*

(command line equivalent: none) Gives the name of a "Hosts" file that lists machines, access methods, and directories of remote modules. It provides a personal override to the system default */usr/avs/runtime/hosts* file when you click on the Network Editor's **Read Remote Module(s)** button under **Module Tools**. See the "Running Remote Modules" section in the *AVS User's Guide* "Advanced Network Editor" chapter for details.

ImageAutomagnify *switch*

In AVS 2, the display image window would try to select an appropriate image magnification factor when the window changed size. In AVS 3 and later releases, images are not automatically resized when the window is resized. Turning this option on will restore the AVS 2 behavior of automatically magnifying the image. The default is *off*.

ImageScrollbars *switch*

(command line equivalent: none) If set to the value **off**, suppresses the adding of scrollbars to display windows that are too small for the image they are currently displaying. (You can always see more of the image simply by dragging it with the mouse.)

ModuleLibraries *filespec filespec ...*

(command line equivalent: **-library**) Specifies which libraries of modules will be loaded into the Network Editor's module palette. The *last* module library listed will be the "default" library showing in the module palette when you enter the Network Editor. The other module libraries listed can be called up by clicking on their iconic representation at the top of the Network Editor's main panel. There is no way to continue the list of module libraries to a new line; the list must be on one (perhaps very long) line.

ModulePanelHeight *integer*

(command line equivalent: none) Controls the proportion of the Network Construction window devoted to the module Palette as opposed to the Workspace.

NetworkDirectory *directory*
(command-line equivalent: **-netdir**) Specifies the directory in which the AVS Network Editor subsystem initially will look for network files (**Read Network** and **Write Network** functions).

NetworkWindow *Xgeometry*
(command line equivalent: none) Specifies the X Window system geometry of the Network Construction Window, which includes the Network Editor menu, the Module Palette, and the Workspace in which you construct networks of modules. You may need this if your display is substantially smaller than the usual 1280x1024 pixels.

This is not affected by the **ScreenSize** option. That is, it is assumed to be in pixels for the screen you are using and not the canonical 1280x1024 screen size.

NetWriteAllParms *switch*
(command line equivalent: none) Save all parameter values when writing out a network with the Network Editor's **Write Network** button, not just those changed since the network was created. The default is to save only the changed parameters.

NoHW *switch*
(command line equivalent: **-nohw**) **NoHW 1** tells the AVS Geometry Viewer to not initialize any hardware renderer. Without a hardware renderer, the AVS Geometry Viewer will use a software renderer to create its 3D scenes instead of the platform's native graphics facilities. **NoHW 1** is used when you are running AVS as a remote X client on a different hardware platform or when you are using an "X terminal." The software renderer creates an X image rendering of the 3D scene and ships only the image to the local X server for display rather than a stream of rendering commands that the local display may not understand. The default is **NoHW 0** (do initialize hardware renderers) on systems that support a hardware renderer.

NoHW 1 is equivalent to the obsolete **SWRender 1**.

Path *directory*
(command-line equivalent: **-path**) Specifies the directory tree in which AVS itself is installed.

PrintNetwork *command*
(command line equivalent: none) The Network Editor's **Print Network** button normally sends output to your default printer. This lets you specify an alternate print command to execute. The output filename is appended to the string you provide. The command should a regular shell command such as:

```
lpr -Plw2
```

ReadOnlySharedMemory *switch*
(command line equivalent: none) Shared memory is normally "read only." Occasionally, the system developer might wish to keep shared memory

turned on, but allow it to be written into. Setting **ReadOnlySharedMemory 0** accomplishes this. The default is **1**. (Note: shared memory may not be implemented on all systems. See your AVS release notes.)

Renderer

(command line equivalent: **-renderer** "*string*") Specifies which renderer will be the default selected in the Geometry Viewer when the first camera window is created. *"string"* is the literal name found on the renderer buttons under the Geometry Viewer's **Cameras** menu, usually either "Software Renderer" or "Hardware Renderer", though other strings are possible. It must match exactly, in spelling, case, and spacing. The double quote marks must be present. Where there is a hardware renderer available, **Renderer** defaults to "Hardware Renderer". If the user specified **NoHW 1**, then only one renderer is available, the software renderer, and this option is ignored.

SaveMessageLog *switch*

(command line equivalent: none) If set to the value **on**, causes the AVS message log to be preserved when the AVS session ends normally. By default, the message log (*/tmp/avs_message.log_**XXX***, where ***XXX*** is the AVS process number) is deleted automatically. The log file is always preserved if AVS exits abnormally (e.g. **Ctrl-C** interrupt, system crash).

ScreenSize *XDIM*x*YDIM*

(command line equivalent: **-size**) Specifies the size of AVS's virtual display in pixels, confining AVS to run within this area. AVS scales its interface to fit the virtual screen.

SharedMemory *switch*

(command-line equivalent: **shm/noshm**) Specifying **SharedMemory off** turns off AVS's shared memory feature.

SpaceballDevice *devicefilespec*

(command-line equivalent: **-spaceball**) Indicates the serial communications port to which a Spaceball device is attached (e.g. */dev/tty1*). If **Spaceball** is specified, AVS automatically connects the Spaceball to the Geometry Viewer's rotate, translate, and scale transformations. This entry also corresponds to the environment variable SPACEBALL. Spaceballs may not be supported on all platforms.

StackSelector *option*

(command line equivalent: none) People who build very large networks sometimes find that the Network Editor's control panel "overflows," making some of the module buttons difficult to access. Setting **StackSelector choice_browser** displays the module names as a fixed-size scrolling list similar to the file browsers instead of as the default **radio_buttons**.

VisualType *visualtype*

(command line equivalent: none) This command may be necessary when you are seeing less color rendition than you know your display is capable of.

AVS normally uses the X server's default visual. Occasionally, this is the wrong visual to use. For example, the default may be set to PseudoColor when there actually is a TrueColor visual available. (The standard X Window System command to list which X visuals are available and which is being used as the default is *xdpyinfo.* This command may not be available on all platforms. Its use is described in the "AVS on Color X Servers" appendix.)

VisualType lets you specify a *visualtype*, either **PseudoColor, TrueColor,** or **DirectColor. A**VS will then search the X server's visual list until it finds the *first* visual with the given visual type and use it.

You can also specify an explicit visual using the string **VisualID** followed by a number *n* that is the decimal equivalent of the X server's hexadecimal visual id for the visual you want to use. For example:

```
VisualType  VisualID  41
```

This degree of precision would be necessary if there were, for example, multiple TrueColor visuals defined, where the first was a more limited 12-plane and a subsequent TrueColor visual the full 24-planes.

This option may also be useful to people using AVS from "X terminals." See the appendix for more information on determining X visuals.

WindowMgr *mgr*

(command line equivalent: none) This option ensures that the Network Editor's Layout Editor and the X Window System window manager that you are using work correctly together. The default for this parameter is specified in the */usr/avs/runtime/avs.Xdefaults* file. The currently recognized values are: **awm**, **mwm** (Motif-style window managers), **twm**, **uwm**, **dxwm** (a DEC window manager), and **olwm** (Open Look).

XWarpPtr *on*

(command line equivalent: none) Causes the mouse cursor to be automatically moved ("warped") into typein dialog boxes when they appear. **XWarpPtr** is off by default.

AVS Environment Variables

AVS uses the following environment variables. Only DISPLAY must be set correctly before AVS will work.

AVS_ADAPT_TABLE *switch*

Turns on an option that can make modules processing irregular fields with unevenly-distributed data points execute faster in AVS networks. This option is *off* (**0**) by default. See the detailed discussion under "Optimization--Adaptive Block Tables" in the "Advanced Network Editor" chapter.

AVS_GEOM_WRITE_V30 *switch*
A 1 value causes the Geometry Viewer's **Save Scene** and **Save Object** functions to save scenes and objects as Geometry Viewer Script Language *.scene* and *.obj* files, as occurred in AVS Release 3.0 and earlier, rather than in a single CLI *.scr* file. It is provided for backward compatibility. It is 0 (off) by default.

AVS_HELP_PATH
Specifies one or more locations in the file system for AVS to use when searching for on-line help files. See the "On-line Help" appendix of the *AVS Developer's Guide* for more on this variable.

AVS_MEM_CHECK *switch*
AVS_MEM_HISTORY *switch*
AVS_MEM_VERBOSE *integer*
These three environment variables are all used by the alternate memory allocation routines defined in the include file */usr/avs/include/mem_defs.h*. These routines replace the UNIX standard memory allocation utilities such as *malloc* with AVS utilities that perform extensive dyanimic memory allocation / deallocation bug checking. As such, they are only of interest to the advanced module writer.

See the "Memory Allocation Debugging" section in the "Advanced Topics" chapter of the AVS *Developer's Guide* for more information on these utilities.

AVS_MG_TROFF *switch*
Causes the AVS Module Generator to generate its module man page documentation templates in *troff* format rather than the default preformatted text man page using tabs and blanks. This option is 0 (off) by default.

DIALS *devicefilespec*
Indicates the serial communications port to which a dialbox device is attached. Dialboxes may not be supported on all systems.

DISPLAY *host:server.screen*
Used by the X Window System to indicate the display screen at which you're working.

DISPLAYCLASS *string*
string is used to specify an alternate */usr/avs/runtime/Xdefaults* file, such as the supplied */usr/avs/runtime/Xdefaults.X*. Also causes AVS to use alternate *.avsrc.**string*** startup files, both the default in the */usr/avs/runtime* directory (no such alternative is supplied with the release), and user *.avsrc* files. Both may be customized to make AVS behave differently on different types of display hardware, such as an X terminal. **-class** is the command line equivalent.

SPACEBALL ***devicefilespec***
Indicates the serial communications port to which a Spaceball device is attached. Spaceballs may not be supported on all platforms.

Adding to the Applications Menu

The items that appear on the Applications menu are defined in the file */usr/avs/runtime/AVS.applns*. An "application" is a single AVS network. You can customize the Applications menu, either by modifying the system default AVS.applns file, or by using the **Applications** *.avsrc* startup file option to specify a different applications file.

The applications file on the AVS release tape is typically organized as follows:

```
# AVS Application File
#
# builtin AVS2 Image Viewer
# builtin AVS2 Volume Viewer
builtin AVS Demo
$Path/networks/dv/data_viewer  Data Viewer
```

To add a network to the Applications menu, append a line to the */usr/avs/runtime/AVS.applns* file like the following:

```
/usr/username/networkfile Your Application
```

# denotes a comment. The items on the network definition line are:

- The first item is the absolute pathname to the network file.

  This file would have been created with the Network Editor, then saved with the **Write Network** button. (You might have subsequently edited this ASCII network file to, for example, remove references to specific input files, or to change the default location of display windows.)
- The remainder of the line is taken as the label for the Applications menu button. You do not need to enclose it in quotes, even though it may contain blank characters.

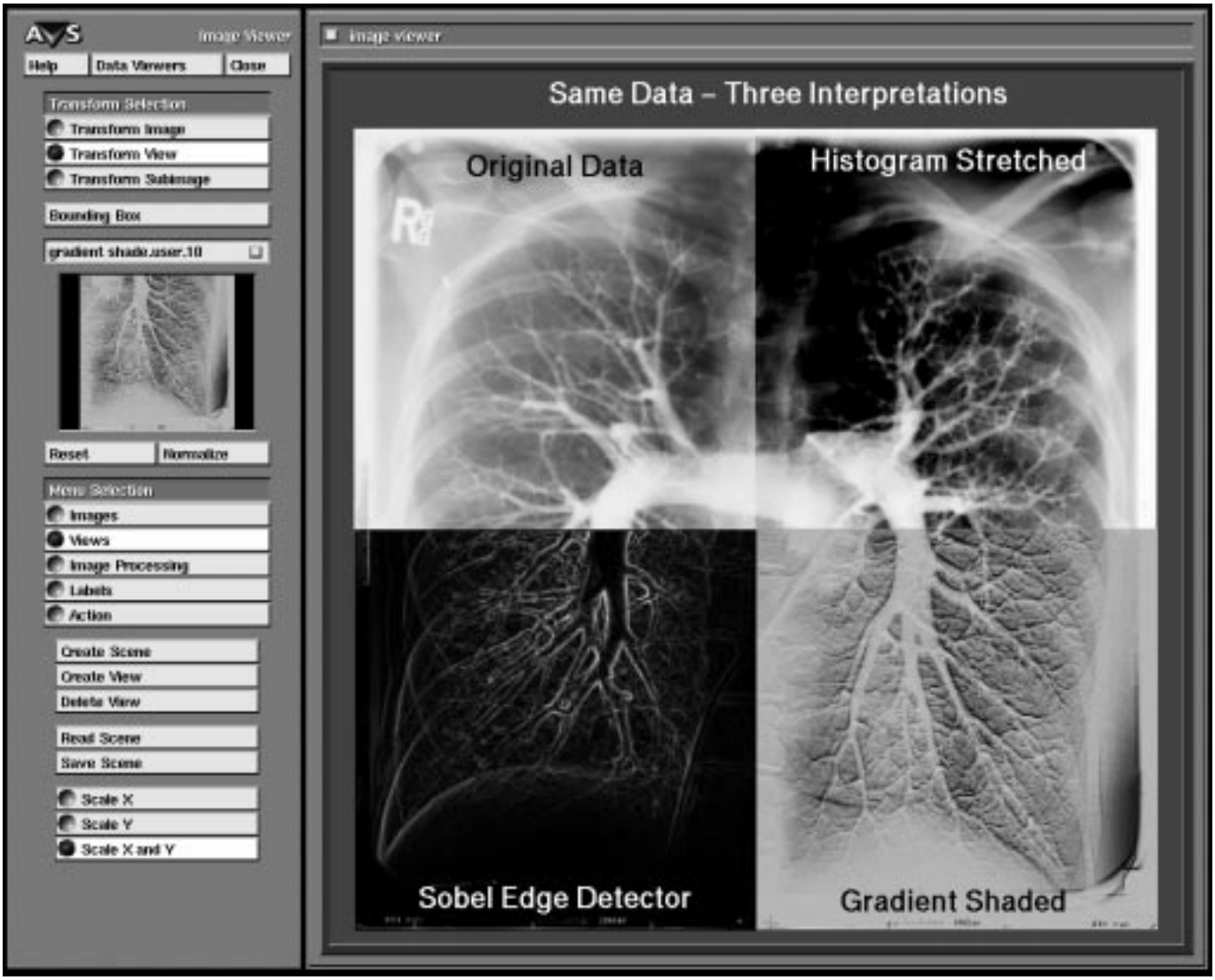
Your own applications file would have the same format. No comment lines are required. The **Applications** line in the *.avsrc* file has the following format:

```
Applications      filespec
```

Where *filespec* is the name of the alternative applications file. If *filespec* starts with a /, it is assumed to be an absolute pathname. Otherwise, it is assumed to be relative to the AVS **Path** value.

As you continue to add networks to an applications file, the buttons on the menu panel will be resized to accommodate the newcomers, up to a reasonable limit.

When a user presses your application button, AVS reads in the network file defined for it and puts its network control panel upon the screen, along with any output display windows.


AVS
Image Viewer
Help
Data Viewers
Close
Transform Selection
Transform Image
Transform View
Transform Subimage
Bounding Box
gradient shade.user.10
Reset
Normalize
Menu Selection
Images
Views
Image Processing
Labels
Action
Create Scene
Create View
Delete View
Read Scene
Save Scene
Scale X
Scale Y
Scale X and Y
image viewer
Same Data – Three Interpretations
Original Data
Histogram Stretched
Sobel Edge Detector
Gradient Shaded

CHAPTER 4

IMAGE VIEWER SUBSYSTEM

Introduction

The AVS Image Viewer subsystem is an interactive tool for displaying, manipulating, and processing *images.*

The Image Viewer exists in two forms: as the Image Viewer subsystem accessible from the main AVS menu, and as the **image viewer** module in the Network Editor's module Palette.

Figure 4-1 Image Viewer Subsystem

Figure 4-2 Image Viewer Module

The Image Viewer performs these functions:

Image 'Display Manager'

In the same way that a window system such as the X Window System manages a display screen full of many *windows*, of different sizes, overlapping, stacked one above the other that you can create, delete, move around, resize, raise and lower; the Image Viewer manages a window full of AVS *images*, of different sizes, overlapping, stacked one above the other, that you can create, delete, move around, resize, raise and lower.

The images can come from two sources. You can read them in directly from disk with the Image Viewer's **Read Image** button, or they can flow into the **image viewer** module from an AVS network.

Image Processor

The Image Viewer's **Image Processing** function calls up a sample choice of networks that implement image processing techniques such as edge detection, and contrast stretching. These techniques can be applied to whole images, or to interactively defined *parts* of images called **subimages**. (Subimages are sometimes referred to as "Regions of Interest" (ROI).) The results can be viewed then erased, or made a permanent part of the output image.

You can apply image processing techniques *serially* to the same image "in situ". You can interpolate, then manipulate contrast, *without* writing intermediate images to disk and reading them back in again between each step.

You are not restricted to just the sample image processing networks supplied. You can create your own networks, save them, and then call them up through the Image Viewer's interface.

The figure facing the first page of this chapter shows the Image Viewer's image processing in action. Various image processing techniques are being used to enhance each quandrant of a chest X-ray.

Supporting these two basic functions are these additional features:

Views

Sets of images are collected together in a **scene**. There can be multiple scenes, i.e., there can be multiple sets of images on the screen at once.

Each scene can have multiple **viewports**. A **viewport** is a window displaying the same set of images in the same configuration, but from a different point of view.

Configurations of images in scenes can be saved to disk, then read in again at a later session.

Labels

Whole scenes and individual images can have alphanumeric **labels** attached to them in various font styles, sizes, and colors.

Action Animation

The **Action** submenu implements a form of "flipbook" animation. A sequence of images flowing into the Image Viewer from the network can be collected into a cycle of images that can be replayed at a controlled speed. You can save the cycle of images to disk and replay them at a later session.

Command Language Interpreter

Most of the Image Viewer's functions can be driven from a command file through the AVS Command Language Interpreter (CLI).

Entering the Image Viewer

The Image Viewer can be entered in four ways:

From the shell directly

The following command line invokes the Image Viewer automatically when AVS starts execution:

```
avs -image
```

When you start AVS and the Image Viewer in this fashion, you cannot transfer to any other AVS subsystem. See the "Starting AVS" chapter for additional command line options that affect the way the Image Viewer is invoked.

From the main menu

You can start the Image Viewer from the AVS main menu. It is the first choice.

From another subsystem

At the top of each of the four major AVS subsystem control panels (Image Viewer, Graph Viewer, Geometry Viewer, Network Editor) is a button titled **Data Viewers**. Position the mouse cursor over **Data Viewers**, then press *and hold down* any mouse button. A pop-up menu appears. Still holding the mouse button down, roll the cursor down the pop-up menu until "Image Viewer" is highlighted, then release the mouse button. This calls up the Image Viewer's control panel. If you transfer to the other subsystems, then return to the Image Viewer, the Image Viewer's control panel will remain in the state that you left it.

In a network

You can include the **image viewer** module in an AVS network. If you click on the **image viewer** module's "dimple" with the left mouse button, it calls up the Image Viewer control panel.

There are some important distinctions between the way the Image Viewer handles images it receives through its own **Read Image** menu button, and images that flow into it from a network in its **image viewer** module guise. Images received from a network are titled differently. The Image Viewer **Action** submenu can only create animations of images that it receives through a network.

Using the **image viewer** module as a general rendering utility to display data images and pixmaps produced by networks is as powerful and flexible as using the **geometry viewer** module to display geometries. It is a more powerful alternative to the **display image** module.

Both the **geometry viewer** and **graph viewer** modules have image output ports. Thus, you can send an image version of the contents of these windows to the Image Viewer for display, manipulation, image processing and animation by connecting them to the **image viewer** module's input port.

The **image viewer** module also has two output ports:

- The rightmost image output port can be connected to the **image to postscript** module to obtain a PostScript file version of the contents of a scene window.
- The leftmost output port is an image picking port. This port is normally invisible. When you click on an image with the left mouse button, it produces a data structure that reports the X, Y location within the image selected. Other modules can use this information, for example, to display the original numeric value present at that location in the field before it was converted to an ARGB image.

Whichever way you invoke the Image Viewer, you work with objects that are represented in AVS's *image* format. See the "Importing Data Into AVS" chapter for a discussion of *image* format.

There are sample AVS image format files in the directory */usr/avs/data/image*.

Leaving the Image Viewer

If the Image Viewer was invoked from the shell command line as "avs -image" then at the top of its main control panel will be a button labeled **Exit**. Press **Exit** with any mouse button to return to the Unix shell.

If the Image Viewer was entered from the main AVS menu or through the **Data Viewers** pop-up menu from another subsystem, then there will be a **Close** button at the top of its main control panel. **Close** is not really an exit button. **Close** simply takes down the Image Viewer's control panel; one could get a similar effect by using the system's window manager to iconify the control panel. When you later re-enter the Image Viewer, the control panel is in the same state that you left it. The only real way to exit the Image Viewer is to exit AVS altogether from the main menu.

If the Image Viewer was invoked as the **image viewer** module, then there is still only one Image Viewer, even though there may be multiple **image viewer** modules. The modules are associated with individual Image Viewer *scene* windows, not with the Image Viewer itself. Throwing away an **image viewer** module deletes its associated scene window.

Image Viewer: Basic Layout

The Image Viewer has three types of windows:

- The main Image Viewer Control Panel
- The **viewports**. These are display windows that contain sets of images.
- Various pop-up browser windows and typein panels that are used for file input, selecting image processing techniques, designating the current image, and specifying input and output files.

Image Viewer Control Panel

Figure 4-3 is the main Image Viewer control panel. Throughout the control panel and its submenus, press *any* mouse button to select *any* of the control buttons.

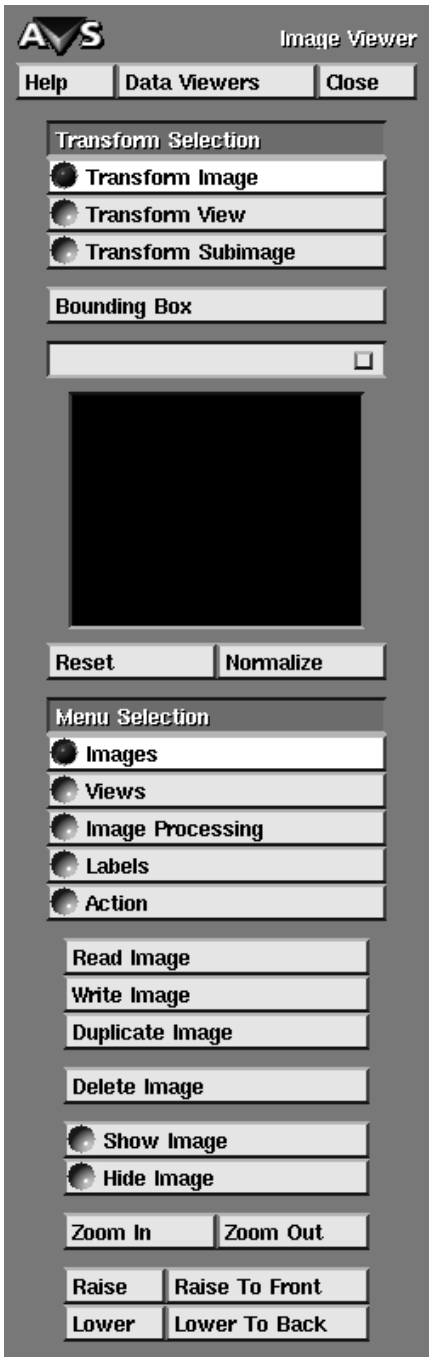


Figure 4-3: Image Viewer Control Panel

The following sections discuss each of the major areas of the main control panel.

Top Control Bar

Figure 4-4 Top Control Bar

Help

Pressing any mouse button over **Help** invokes the AVS Help Browser window. By default, the Help Browser will display a selection of topics specific to the Image Viewer. The browser's **close** button removes it. The Help Browser is described in detail in the "Using Online Help" section of the "Starting AVS" chapter.

Data Viewers

Pressing any mouse button over **Data Viewers** brings up a pop-up menu that switches among the three AVS viewers (Image Viewer, Graph Viewer, and Geometry Viewer). Press any mouse button and *hold it down*. The pop-up menu appears. While still holding the mouse button down, roll the mouse cursor down the list until the subsystem you want to switch to is highlighted; then, release the mouse button.

Switching to another subsystem does not "exit" or "halt" the Image Viewer, it just brings up the new viewer's control panel, covering up the Image Viewer's.

Close

Pressing any mouse button over **Close** removes the Image Viewer's control panel from the screen. It does not remove any Image Viewer viewport or browser windows, nor does it "exit" the Image Viewer. The effect is as though one had "iconified" the control panel.

You do not need to **Close** the Image Viewer control panel to make room for other subsystems' control panels. Just use your window manager to move it to another part of the screen.

Transform Selection Controls

Figure 4-5 shows the Transform Selection area. There are three kinds of things that you can move around in the Image Viewer using mouse button controls: images within viewports; the viewport's *position over* the scene of images (like moving a frame over a stationary picture); or a rubber-banded **subimage** region over an image. The same mouse button combinations move all three things the same way (e.g., the right mouse button always moves things left / right / up / down). The mouse button "verb" is always consistent.

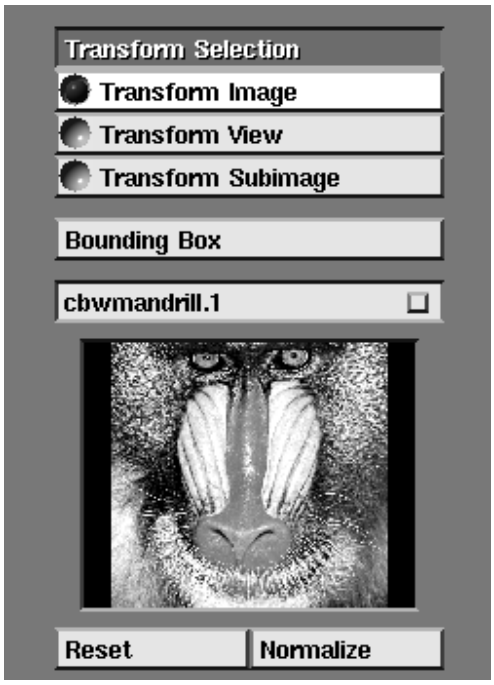


Figure 4-5 Transform Selection Area

The three **Transform Selection** buttons select *which* of the three objects (image, viewport, subimage) is going to be moved by a mouse button command. It is a mouse button mode switch.

Transform Image

With **Transform Image** selected, the right mouse button moves the current *image* up / down / left / right. The shift-middle mouse button combination scales the current image, making it larger or smaller.

Transform View

With **Transform View** selected, the right mouse button moves the current *viewport* left / right / up / down over the fixed set of images— shifting the point of view rather than the objects. The shift-middle mouse button moves the point of view in towards the images, or back away from the images.

Transform Subimage

Transform Subimage designates that the right mouse button will move the rubber-banded *subimage* region left / right / down / up. Shift-middle mouse button *does not*, however, resize the subimage region. This is easily-enough done by just making a new subimage region.

To define a subimage, press and hold down shift-left mouse button. Drag the mouse cursor to define the subimage region, then release the mouse button. Subimages can be defined at any time; **Transform Subimage** does not have to be selected.

Bounding Box

The **Bounding Box** toggle switch changes the way images, viewports, and subimages move with the mouse. It is a good thing to select when you are using AVS on a less-powerful or heavily-loaded graphics workstation.

Normally, when you move or resize an image, viewport, or subimage, the system does its best to update the rendering of the picture *continuously*, in "real time" as it tracks the mouse. So, as you move an image from point A to point E, the system tries to keep the picture "live." In practice, what you get are several intermediate pictures, B, C, D, as the image moves from A to E. The more horsepower your system has, the smoother and more continuous the image motion. The more loaded a given system is or the more complex the image sets, the slower the image display.

Bounding Box avoids this resource-expensive effort at real time redisplay. With **Bounding Box** turned on, when you place the mouse over the current transformable and press the right or shift-middle mouse button, a white wireframe box enclosing the area of the image/viewport/subimage appears. As you hold the button down and move the mouse, the *bounding wireframe box* moves—the image/viewport/subimage does not. You move the bounding box to the destination position, then let go of the mouse button. Only then is the image/viewport/subimage rendered at its new position.

Toggling **Bounding Box** affects all images, viewports, and subimages on the screen.

Current Image Controls

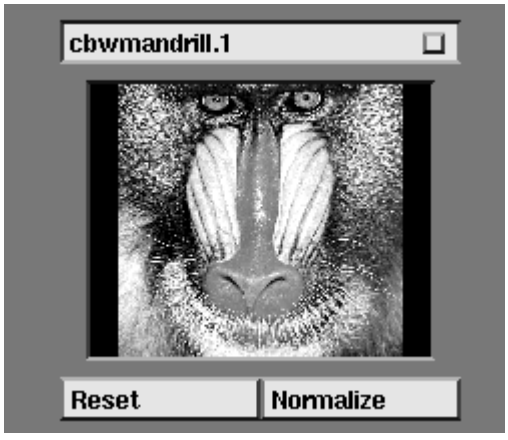


Figure 4-6 Current Image Control

The next set of buttons and the miniature image window control operations on the **Current Image**. See Figure 4-6. You can have many images existing in many scenes and many viewports. When you start performing image processing techniques, or moving images around with the mouse buttons, or raising and lowering images with respect to one another in a scene, *one image* has to be the object of the command: something has to be the **Current Image**.

Image Title Bar—How Images are Named

The Image Title Bar shows the name of the Current Image.

Each image that enters the Image Viewer gets a name. The name has two parts, a name and a sequence number: *name.sequence#*. If the image was read in directly from a disk file with the **Read Image** command, its *name* will be the same as its disk file name without the ".x" extension. If the image came into the Image Viewer from an AVS network, its *name* will be the same as the name of the AVS module that sent it to the Image Viewer (e.g., "crop"). *sequence#*'s are assigned sequentially, according to the order the images entered the Image Viewer.

The Image Viewer has no concept of "top" image as the AVS Geometry Viewer has (see the "Geometry Viewer" chapter). Images are not organized into hierarchies, nor do Image Viewer scenes and viewports get names.

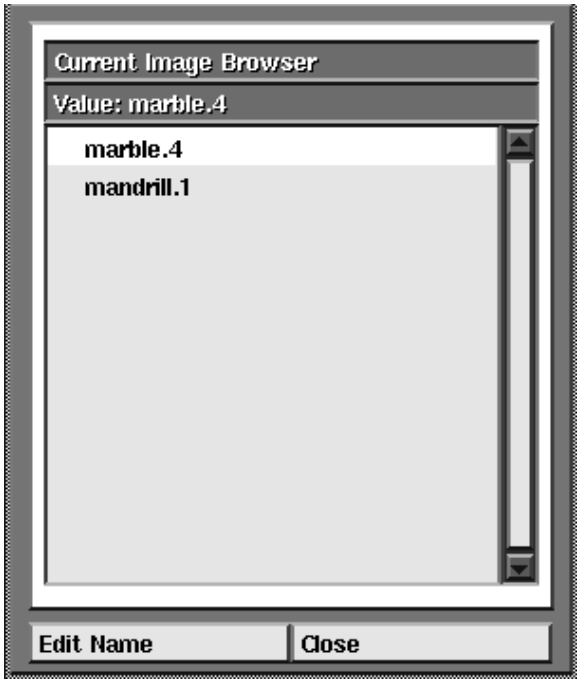


Figure 4-7 Current Image Browser

Current Image Browser—Retitling and Picking Images

The Image Title Bar also provides an alternate way to select the Current Image. It is useful when there are large numbers of images, some of which may be in obscured windows not easily designated with the mouse cursor.

Use any mouse button to click on the "dimple" at the right of the Current Image title bar. This produces a **Current Image Browser** window. (Figure 4-7.)

Like all AVS browsers, you pick an image by highlighting its name with the mouse cursor and pressing any mouse button. The Current Image

Browser window is "sticky," that is, it stays on the screen until removed by clicking on its **Close** button. Closing the Current Image Browser window is like iconifying it. If you call it up again, it will be in the same state as you left it.

The Current Image Browser window also renames images. Click on the browser's **Edit Name** button. A new window pops up with a typein area for the new image name. The mouse cursor must be moved inside the typein area or anything you type is ignored. For simple editing functions, the **Backspace** key deletes the previous character, and **Ctrl-U** erases the whole line.

Note: Do not retitle images if you intend to save the network containing the **image viewer** module. When the network is read in again, it will not be able to find the retitled image.

Current Image Window

The miniature Current Image Window displays an immediately-recognizable miniature picture of the Current Image.

You can use the Current Image Window to select the current image in yet another way. With the mouse cursor in the Current Image Window, start clicking any mouse button. This cycles through the images in the currently-selected scene.

Reset/Normalize

The **Reset** button causes the Current Image to revert back to the same size and same position it had when it first entered the scene, undoing any moving or rescaling that might have been done to the image. It does not remove Image Processing techniques performed on the image, even those not made permanent with **Set Current Image**, nor does it reset an image's name. **Reset** works on individual images; it does not reset whole scenes.

The **Normalize** button expands the size of an image until the larger of its two dimensions fills the current viewport. This works even if the viewport window has been resized. If an image is offset within the viewport window, it is enlarged to the same degree it would have been if it were centered in the window, but its relative position within the viewport is not altered. **Normalize** does not work when **Transform View** or **Transform Subimage** is selected.

Function Key Usage

The Image Viewer uses the function keys on the terminal keyboard as follows:

Table 4-1 Function Keys

Function Key	Function
F1	Transform Image
F2	Transform View
F3	Transform Subimage
F4	no function

Table 4-1 Function Keys

Function Key	Function
F5	toggle **Bounding Box** mode
F6	cycles through images in a scene
F7	Reset
F8	Normalize

Menu Selection Controls

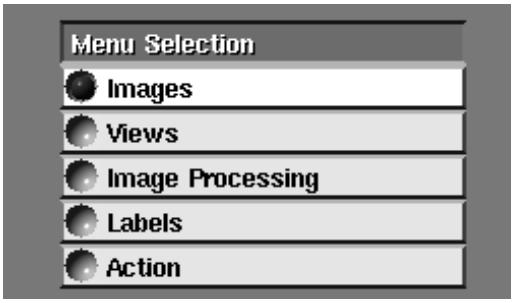


Figure 4-8 Main Image Viewer Menu Buttons

This area of the main control panel contains five buttons that select among the five Image Viewer submenus. This is the "main Image Viewer menu." The default is **Images**.

Submenu Controls

This is the only part of the main control panel that changes. When you select among **Images**, **Views**, **Image Processing**, **Labels**, and **Action**, their unique submenus containing their individual controls appear in this area. Figure 4-9 shows the submenu for the **Images** selection. Buttons in this submenu area are somewhat indented, indicating their sub-level status. Some sub-level menus have their own sub-submenus that also appear in this area.

When the Image Viewer first comes up, the default **Images** submenu is showing here. As you move among these submenus, their state is saved for the next time you enter them.

The state of these submenus may also differ among scenes and images. For example, in one scene an image may be hidden while in another it is visible (**Show Image**/**Hide Image**). The correct state of the controls is saved for each image and scene; as you change the current image or scene, the controls change to reflect the correct settings.

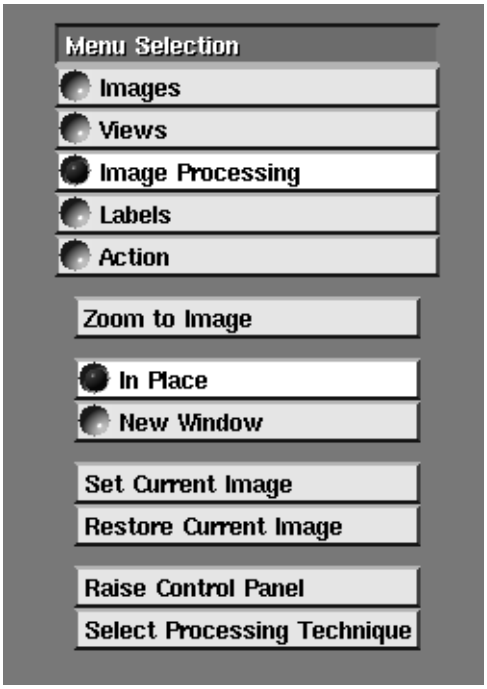


Figure 4-9 Submenu Controls for Image Processing Button

Viewport Windows and Scenes

Image Viewer **Scenes** are sets of images, of various sizes and positions, with a particular "stacking order", i.e., A is above B, which are both above C.

Scenes, in themselves, never appear on the screen. What you see on the screen is one or more **viewport windows** looking *onto* a scene.

In Figure 4-10, the top two windows are two viewports onto the same scene, while the bottom window is a viewport onto a different scene. The bottom viewport is the current viewport, designated by its red border.

Transforming Viewports

Viewport windows move in two dimensions. To move a viewport, **Transform View** has to be toggled at the top of the Image Viewer main control panel. The transformations that work on viewports are as follows:

- **Right Mouse Button**—Moves the viewport up / down / left / right over the collection of images. This is like moving a frame over a collage of pictures.
- **Shift-Middle Mouse Button**—Moves the viewport inward and outward to and from the collection of images. The effect is similar to "zooming" in and out from the scene.

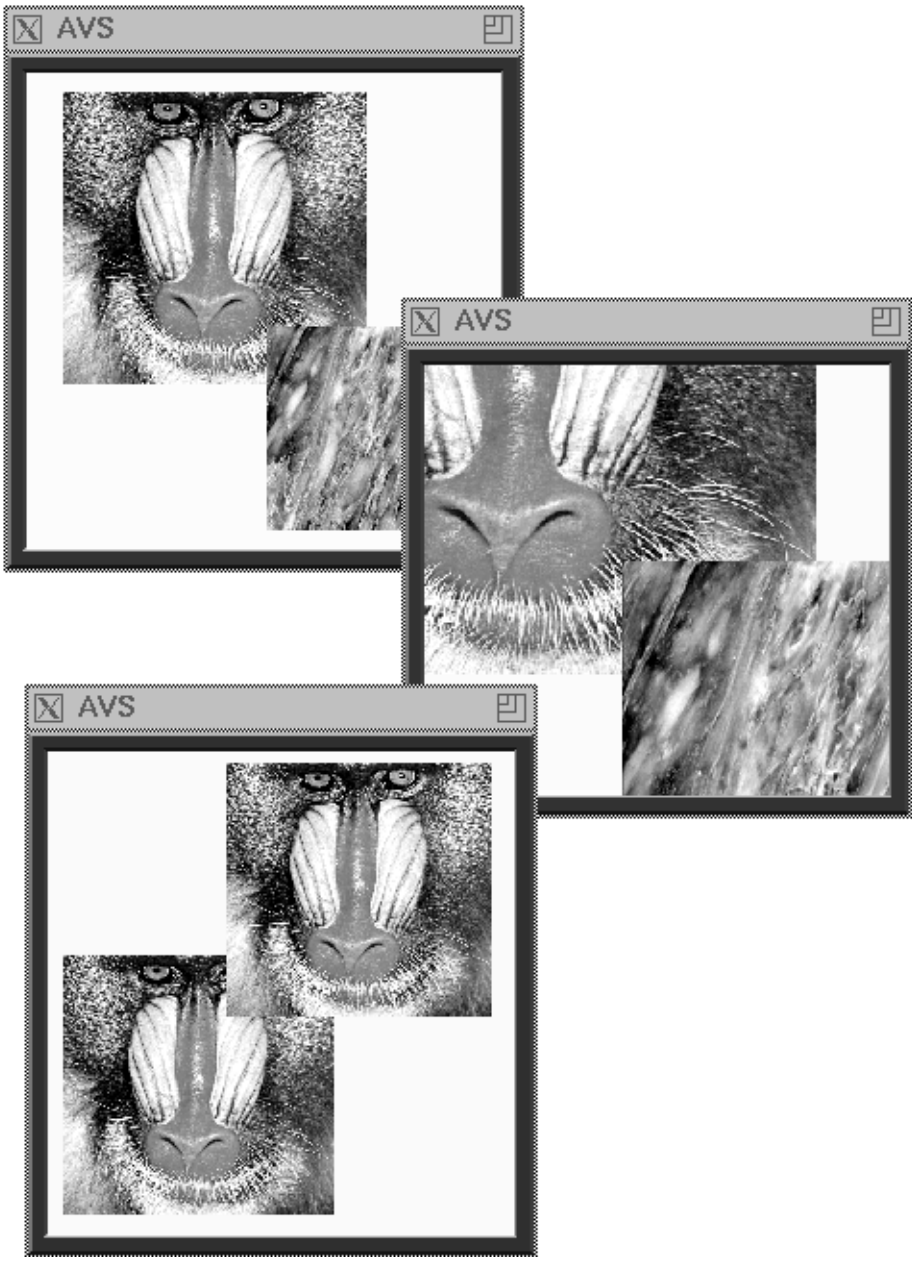


Figure 4-10 Viewport Windows

Current Viewport: Switching Among Viewports

Just as there is a current image that will be the object image of any command, so there is a **current viewport** that will be the object of any viewport command. The current viewport is surrounded with a red border.

- **Any Mouse Button** —Pressed anywhere in any viewport window makes it the current viewport.

Resizing Viewports

You resize a viewport window using whatever X Window System window manager you have running, just like any other X window.

When you resize a viewport window, the sizes and relationships among the images inside the viewport do *not* change. As a window gets larger, you just get more background. The Image Viewer does try to keep at least part of all the images in a scene visible in the viewport window by shifting the viewport slightly, but it will abandon this strategy and clip images if it has to.

Other aspects of viewports and scenes are discussed below under the **Views** submenu.

Browsers

The **Read Image**, **Read Scene**, **Select Processing Technique**, and the Current Image Title bar all bring up AVS browsers. The browsers are "sticky," that is, they remain up on the screen until they are taken down with their **Close** button. They maintain their state from incarnation to incarnation. To pick an item from the browser, highlight it with the mouse cursor then press any mouse button. If a list is too long to fit on one page of the browser, it can be scrolled.

Each browser's use is discussed under the menu button that invokes it.

The remainder of this chapter discusses each of the Image Viewer submenus and the Image Viewer Command Line Interpreter.

Images Submenu

The **Images Submenu** (Figure 4-11) provides the basic utilities for manipulating individual images. (**Note:** If you are running AVS on a system with the visual type PseudoColor, five additional controls will appear at the bottom of the list. These are explained at the end of this section.)

Activate the functions by positioning the mouse cursor over the button until it is highlighted, then press any mouse button. The buttons always work upon the *current* image or scene.

The current scene is the one that has one of its viewports surrounded with a red border. There is one way to designate the current scene:

- Move the mouse cursor until it is in one of the viewports looking upon the scene. Press **any** mouse button.

As noted earlier, there are three ways to select the current image:

- Position the mouse cursor over the image in a viewport that you want to become the current image. Press the **left** mouse button.

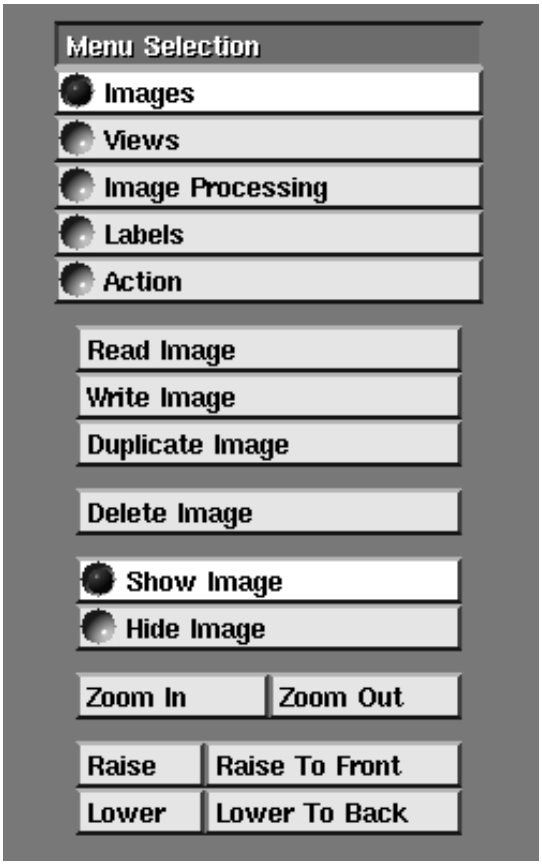


Figure 4-11 Images Submenu

- With the mouse cursor over the Current Image Window on the Image Viewer control panel, press **any** mouse button. The Current Image Window will cycle through the images in the current scene.
- Press any mouse button over the dimple at the right of the Current Image Title bar. A pop-up browser appears listing all the image names in the current scene. Roll the mouse cursor down the list until the image name that you want is highlighted, and press **any** mouse button.

As a by-product of selecting a current image, you are also setting the current scene to be the scene of which the image is a member.

Read Image

Press **Read Image** to read an image directly into the current scene. The image file must be in AVS image file format. **Read Image** produces a pop-up file browser. *Which* directory the file browser will be showing is controlled by three things, in this order of precedence:

- If you invoked **avs** with the **-data** *directoryname* option, then the file browser will come up displaying the contents of *directoryname.*
- If your *.avsrc* file contains a DataDirectory specification, then the file browser will come up displaying the contents of the DataDirectory.
- In the absence of other instructions, the file browser will come up displaying the contents of the */usr/avs/data* directory. Sample image-format files supplied with AVS are found in its *image* subdirectory.

The file browser displays:

- **Directory names** in red
- Filenames with the **.x** image file suffix in black
- Filenames with the **.ims** Image Viewer scene file suffix in black.

All other files are invisible. If you want to read in an image file that does not have the *.x* or *.ims* suffix, you must press the file browser's **New File** button and type out the full filename explicitly.

The file browser window stays up on the screen until you press its **Close** button. If you **Close** the file browser window, then open it again by pressing **Read Image**, it will once again display the contents of the directory chosen according to the precedence list above.

Write Image

Write Image writes a copy of the current image into a file in AVS image file format. In the absence of any **-data** command line option or *.avsrc* file **DataDirectory** specification, the Image Viewer will try to write the file into the */usr/avs/data* directory. This is a system-owned directory that may not permit users to write to it. Instead, move the mouse cursor into the typein window and type a full directory / filename specification for a directory that you have write access to. When in the typein window, **Ctrl-U** deletes the entire line, and **Backspace** deletes the previous character. The Image Viewer will automatically append the *.x* image file suffix. Finish the typein by pressing **Enter** or by clicking the **OK** button. (Note: on systems with filename length restrictions, there may be a warning in the typein window cautioning you to keep your filenames less than a certain length.)

When you write the current image, you are saving it "as you see it" on the screen.

Duplicate Image

The **Duplicate Image** button makes a copy of the current image within the same scene. It does not copy images between scenes. (To do this, first **Write Image** from one scene, change current scenes, then use **Read Image** to get it into a new scene). The new image will have the same *name* as the original im-

age, but a higher *sequence#*. It will be centered within the scene, rather than the viewport.

Delete Image

This deletes the current image from the current scene. There is no "undo."

Show Image/Hide Image

By switching between these two modes, the current image can be made visible or invisible within a scene. Though invisible, an image can still be manipulated just as though it were still visible.

Zoom In/Zoom Out

Zoom In/Zoom Out is a separate way of resizing images where the overriding intent is to keep the original image data exactly preserved.

Zoom In makes an image larger by multiples of four; 4x's, 16x's, etc. its original size. **Zoom Out** makes a "Zoomed In" image smaller again by the same fixed multiples. Figure 4-12 shows the eye section of the *mandrill.x* image enlarged to great size. Zooming is similar to rescaling the size of the image with the shift-middle mouse button, but with these important differences:

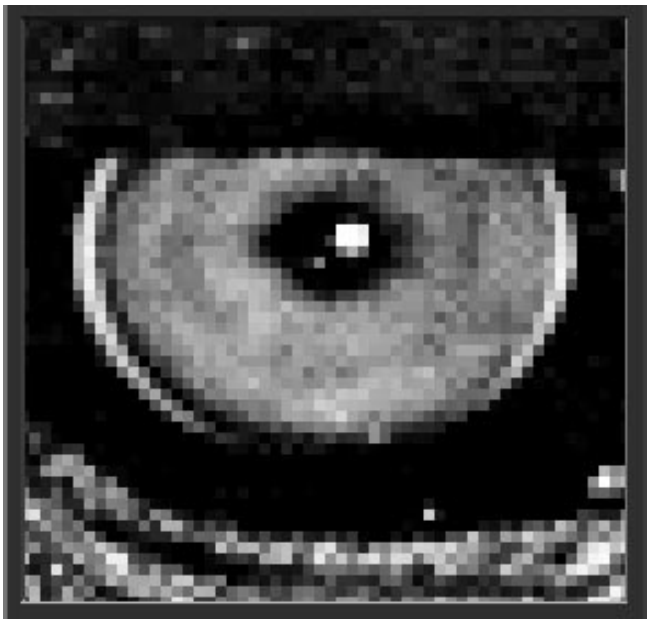


Figure 4-12 Zoom In Image Showing Pixel Replication

- Both **Zoom In/Out** and rescaling with the shift-middle mouse button redraw the image using "pixel replication." The difference is that **Zoom In** works by taking one pixel in the original image and making it into four pixels ("replicating" it) in the new image, enlarging the area of the image by four. **Zoom Out** takes the four equal pixels in an image *that was made larger with Zoom In* and turns them back into one pixel. **Zoom In/Out** are inverse operations that resize an image by multiples of four. On-screen image fidelity is always preserved.

  On the other hand, when you rescale an image with the shift-middle mouse button, the result does not have to be an even multiple of the original image size. The picture might be 30% larger or smaller. In this style of rescaling, an algorithm is used to select 30% of the pixels in each dimension of the image, and replicate (or delete) only them. The picture gets bigger or smaller, but absolute image fidelity on the screen is sacrificed. (Since the Image Viewer keeps a copy of the original image, you don't actually lose any pixel data if you shrink an image, then enlarge it again. You will only see the effect if you use **Write Image** on a highly-shrunken image, then read it back with **Read Image**.)
- Because **Zoom In** and **Zoom Out** are following a precise approach to enlarging and shrinking images, you cannot **Zoom Out** an image (make it smaller) unless you have first used **Zoom In** to make it larger.

  Nothing happens if you press **Zoom In** on an image that has been rescaled to be *smaller* than its original size with the shift-middle mouse button. Image data would be lost.

  Similarly, nothing happens if you press **Zoom In** on an image that has been rescaled larger or smaller in just the X or Y direction. Again, image fidelity would be compromised.

  For all the same reasons, you cannot **Zoom In** on an image, then use shift-middle rescaling, then try to use **Zoom Out**. When you press the latter, nothing will happen.

Raise/Lower: Image Stacking Order

Multiple images in scenes have a "stacking order" in the same way that windows on a display screen have a stacking order. Though the images might not even overlap, still one image is on top, one is on the bottom, and any additional images are in some order in the middle. **Raise** raises the current image *one* level in the stacking order. **Lower** drops the current image one level in the stacking order.

Thus, if you have three images in this stacking order A, B, C, and C is the current image, pressing **Raise** will change the display and stacking order to A, C, B.

Raise to Front/Lower to Back

Raise to Front takes the current image, wherever it is in the stacking order, and makes it the top image. **Lower to Back** makes the current image the bottom image.

Color Dithering Options

The following five controls only appear when running AVS on a system with a PseudoColor visual type. On pseudo color (8-plane) systems, it is necessary to reduce the 16,777,215 (256 red x256 green x256 blue) color values possible in an AVS image down to one of the approximately 216 or so colors available.

There are normally 6 red tones, 6 green tones, 6 blue tones, and 22 gray tones available on a pseudo color device. To display the true color image, AVS takes the origial red value for each pixel and finds the closest numeric value from amongthe 6 reds available. It does the same for green and blue.

AVS then takes the pixmap that is made up of these three best-matches and applies a *color dithering* algorithm to the pixmap. Dithering uses the fact that the human eye will interpolate between dots of color, creating the impression of a color value between two actual color values. The dithering process corrects for information lost in the 256-to-6 true color to pseudo color reduction by comparing how far off each final pixel value was from the original value against a dithering matrix or mask. Some pixel values have their red / green / blue values adjusted up or down to create a closer approximation to the original true color image.

There are many dithering algorithms possible. These controls allow you to select among several dithering options:

dither
Selects the default ordered dither.

floyd-steinberg
Generates better pictures than the default dithering algorithm, but is slower.

random
Uses a randomly-generated dither mask.

monochrome
Computes the luminence of the colors in the input image by combining the red, green, and blue values for each point according to a linear relation. The luminence values are then used to find a grayscale equivalent for each pixel. Selecting **monochrome** converts the color image into a monochrome image resembling a black and white photograph. To extend the grayscale, use the **Colors** *.avsrc* keyword.

none
Each color in the original image is approximated by the closest color in the colormap as described above. No dithering is applied.

AVS does not perform image color dithering on TrueColor or DirectColor visual displays.

Views Submenu

The Views submenu (Figure 4-13) provides the basic utilities for manipulating scenes and viewports.

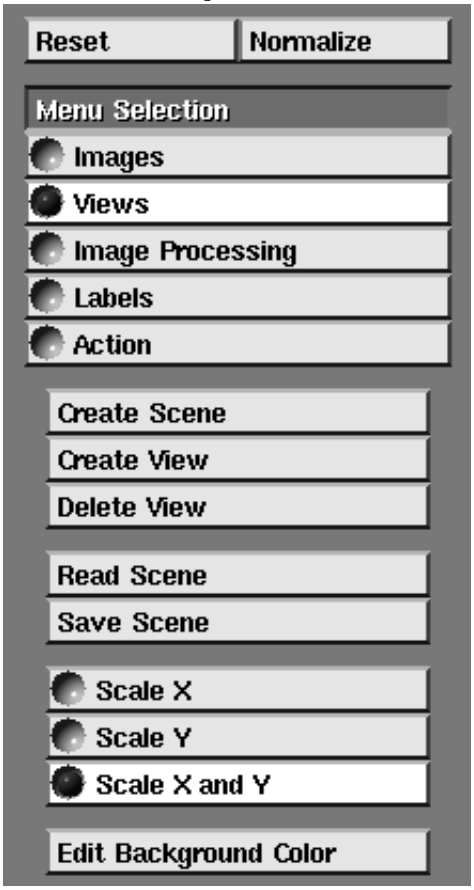


Figure 4-13 Views Submenu

Create Scene

Pressing **Create Scene** makes the Image Viewer start a new, empty scene without any images. In addition, the Image Viewer creates a new viewport onto the empty scene. The new viewport gets the red border that shows it has

become the current viewport, and its empty scene becomes the new current scene. This is the fundamental way to manipulate two or more independent sets of images on the screen at the same time.

Note that when you are in the AVS Network Editor and you drag an **image viewer** module down from the "Data Output" column in the palette into the Network Editor workspace, you have done the equivalent of a **Create Scene**.

Create View

Create View makes a new viewport window onto the current scene. The new viewport becomes the current viewport, with its red border. What you see is a new window that contains the same set of images as the viewport you were in when you pressed **Create View**. It will be the default Image Viewer viewport size.

This new viewport will always show exactly what the older viewport displays— images moved in one will move in the other, etc.— unless you toggle **Transform View** at the top of the Image Viewer control panel. Then, movements you make (right mouse button or shift-middle mouse button) are happening to the viewport's position over the scene, not to the contents of the scene itself.

Delete View and Deleting Scenes

Pressing **Delete View** deletes the current red-bordered viewport. If that viewport is the *last* viewport onto a scene, it will also delete that scene.

If there are other viewports on the screen, one of them will become the red-bordered current viewport / scene in the reverse order that they were originally created.

Save Scene

Save Scene saves the current state of a scene into a file that can be read back at a later session. It is the way to save a snapshot of your work.

Pressing **Save Scene** puts a filename typein window up on the screen. By default, the scene file will be saved in the directory specified by the **-data** *directoryname* option on the **avs** command line, or by the directory defined as the DataDirectory in your *.avsrc* file. If neither of those were specified, it will try to save the file in the user-unwritable */usr/avs/data* directory. To change the directory, type in a complete file specification including the directory path.

Scene files should end with the file suffix **.ims**. This suffix will be automatically appended to the filename.

When you save a scene, you save:

- File references to all the images that have been read into a scene using **Read Image**. The image itself is not saved, just the full filename from where it was originally read.
- All viewports associated with the scene, their background color, and any transformation to the viewport's position over the set of images. The position of the viewports on the display screen is also saved.
- Any transformation that has occurred to the images themselves, including their position within the scene, and any rescaling.
- All scene and image labels, their sizes, fonts, colors, and positions.

Save Scene does *not* save:

- Any image processing technique that was performed on an image, even those made permanent with the **Image Processing** submenu's **Set Current Image** command. (To save a processed image, use **Save Image**.)
- Any image that entered an **image viewer** module from an AVS network.

The general reason in both cases is that the **.ims** file contains references to image files to read in, and image-processed images and images that have arrived through an AVS network have no file associated with them. **.ims** scene files are ASCII files, written in the Image Viewer's set of of Command Language Interpreter (CLI) commands.

Read Scene

Read Scene reads a **.ims** scene file into the Image Viewer. It works just like the other file browsers in the Image Viewer. Note that the file browser will *not* show scene files unless they end with the **.ims** file suffix. To read in a scene file that has no **.ims** file suffix, press **New File** and type in its name.

Scale X, Y, X and Y

This is a set of radio buttons that modifies how the shift-middle mouse button rescales images. It does not affect scenes or viewports. By default, **Scale X and Y** is selected. This just means that when you resize an image with the shift-middle mouse button, it gets bigger or smaller in both the X and Y direction evenly.

If **Scale X** is chosen, then the shift-middle mouse button changes the size of an image *only* in the X direction, making it wider or narrower (Figure 4-14). If **Scale Y** is toggled, then shift-middle mouse button makes the image taller or squatter in the Y direction alone. This is done by causing the "pixel replication" described above under "Zoom In/Zoom Out" to occur only across rows of pixels (X), or only down columns of pixels (Y).

Figure 4-14 Image Scaled in X Only

Edit Background Color

Pressing this button brings up a color editor panel. This panel (Figure 4-15) establishes the background color of the current viewport.

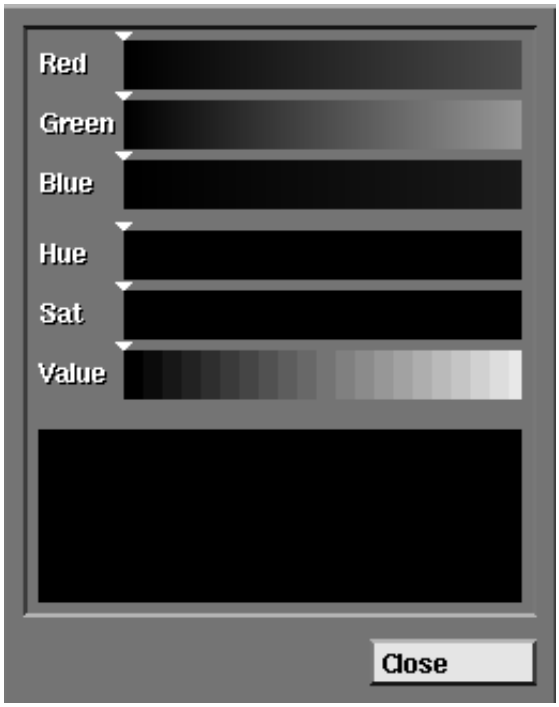


Figure 4-15 Background Color Controls

There are two ways to set the color; either using the Red Green Blue color model, or the Hue Saturation Value model. If you change one set of controls the other set moves accordingly. The default background viewport color is black.

Image Processing Submenu

The **Image Processing** submenu (Figure 4-16) contains the facilities for performing image processing techniques upon images.

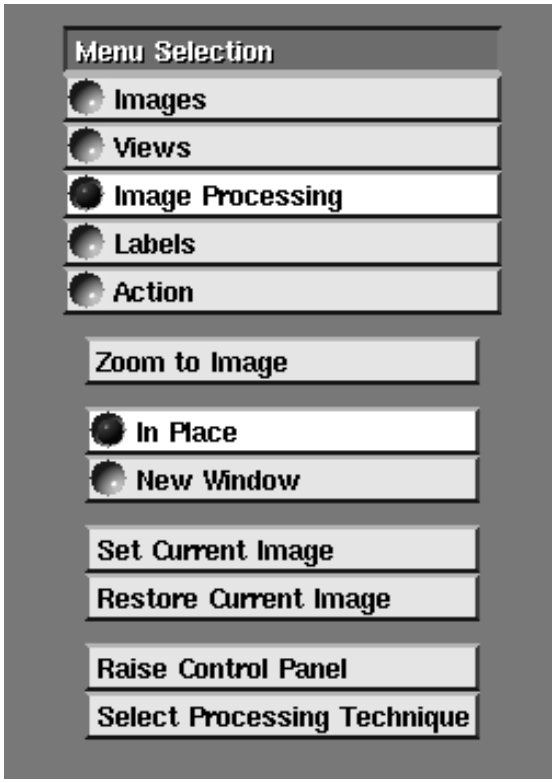


Figure 4-16 Image Processing Submenu

You can apply the technique to an entire image (**Zoom to Image**), or to a movable, rubber-banded region of interest of the image called a **subimage** (shift-left mouse button). The techniques can be applied experimentally to an image or subimage, then erased (**Restore Current Image**), or the change can be made a permanent part of the Image Viewer's working copy of the image (**Set Current Image**). Techniques can also be applied *serially* to the images and subimages. For example, you can contrast stretch an image, then use edge detect upon the modified version of the image. The results of the technique can be displayed on the original image (**In Place**), or made to appear in a separate "scratch" scene (**New Window**).

Any image, whether it entered the Image Viewer from a file through **Read Image** or from an AVS network, can have the techniques performed on it.

The image processing techniques themselves (**Select Processing Technique**) are pre-defined AVS networks containing AVS modules. The Image Viewer comes with a set of sample networks, but it is possible for you to define and use your own image processing networks. This is described at the end of this section.

Basic Procedure: Select Processing Technique

First, you might want to see and watch the networks that implement the techniques. Call up the AVS Network Editor from the main AVS menu. Then, enter the Image Viewer by pressing **Data Viewers** at the top of the left Network Editor control panel and select Image Viewer. The large Network Editor window remains on the screen, while the Image Viewer control panel appears on the left.

Figure 4-17 Processing Technique Browser

1. Using either **Read Image** or a network with the **image viewer** module at the bottom, get an image or images into the Image Viewer viewport.
2. Press **Image Processing** to bring up the correct submenu.
3. Press **Select Processing Techniques**. This brings up the Processing Technique Browser (Figure 4-17). Some twenty techniques are listed alphabetically, usually by their core module name. A scrollbar at

the right of the browser gives you access to the rest of the techniques.

If you happen to be watching the process in the Network Editor, two modules appear in the workspace: **IV read image** and **IV write image**. These are special internal modules that retrieve images from an Image Viewer scene (**IV read image**), and return them to the Image Viewer (**IV write image**).

4. Roll the mouse cursor down the technique browser and press any mouse button to pick a technique. You will see:
   - If the modules that make up the image processing network have control widgets associated with them, then the Network Editor control panel containing these widgets appears.
   - If the modules that make up the image processing network do *not* have any control widgets associated with them, no control panel appears. The visual clue that everything is now "ready" is the mouse cursor. When you select a technique, it turns into the "busy" clock. When it changes back to its regular form, the network has been loaded.
   - Two of the techniques, **geometry mesh** and **geometry contour** AVS geometries as output instead of AVS images. In this case, a **geometry viewer** output window will also appear on the screen in addition to the existing Image Viewer viewport. Similarly, the **generate histogram** technique produces a **graph viewer** window. The output from these techniques will appear in these new windows.

The image processing technique is now "loaded" and ready to apply to an image or subimage.

At this point that you can adjust the technique's control widgets, if present, to control the parameters of the technique's modules.

Note: The Image Viewer control panel window may be obscured by the technique's control widget panel. Use your window manager to move one of the control panels to another part of the screen.

Figure 4-18 shows the network for the **gradient shade** technique as it would appear in the Network Editor. Immediately next to the network is part of the control panel for the technique showing **gradient shade**'s dial widgets.

Zoom to Image: Techniques on Whole Images

When you press **Zoom to Image**, the network processes the entire current image. With the default **In Place** toggled, The resulting image is displayed in place of the original image in the current scene. The original image is still present and can be recalled with **Restore Current Image**, or by pointing at the image with the mouse cursor and pressing the left mouse button, just as though you were redesignating the image to be the current image.

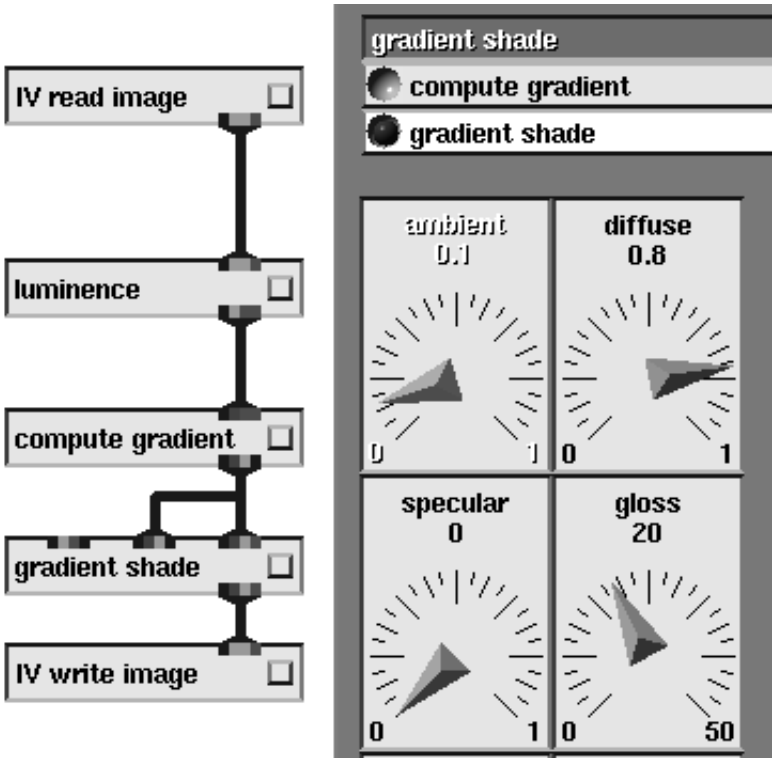


Figure 4-18 Network and Control Panel for Gradient Shade Technique

You must press **Zoom to Image** each time you want to apply a technique to an entire image. (Subimages do not require you to press **Zoom to Image**.)

Shift-Left Mouse Button: Techniques on Subimages

To apply an image processing technique to a part of an image, you specify a **subimage** area.

Position the mouse cursor over the portion of the image you are interested in, press *and hold down* **shift-left mouse button**. With the mouse button still held down, move the mouse. You will see a rubber-band rectangle appear in a contrasting color. Move the mouse until the area of the screen you want to process is enclosed, then release the mouse button. At this point, the network executes, performing the technique on the subimage.

With the default **In Place** selected, the modified subimage appears as a section of the original current image. (See Figure 4-19.) The original whole image is still present and can be recalled with **Restore Current Image**, or by clicking on the image with the left mouse button.

To redefine a subimage area, just press shift-left mouse button again in a new area. The old subimage disappears as you draw the new subimage area.

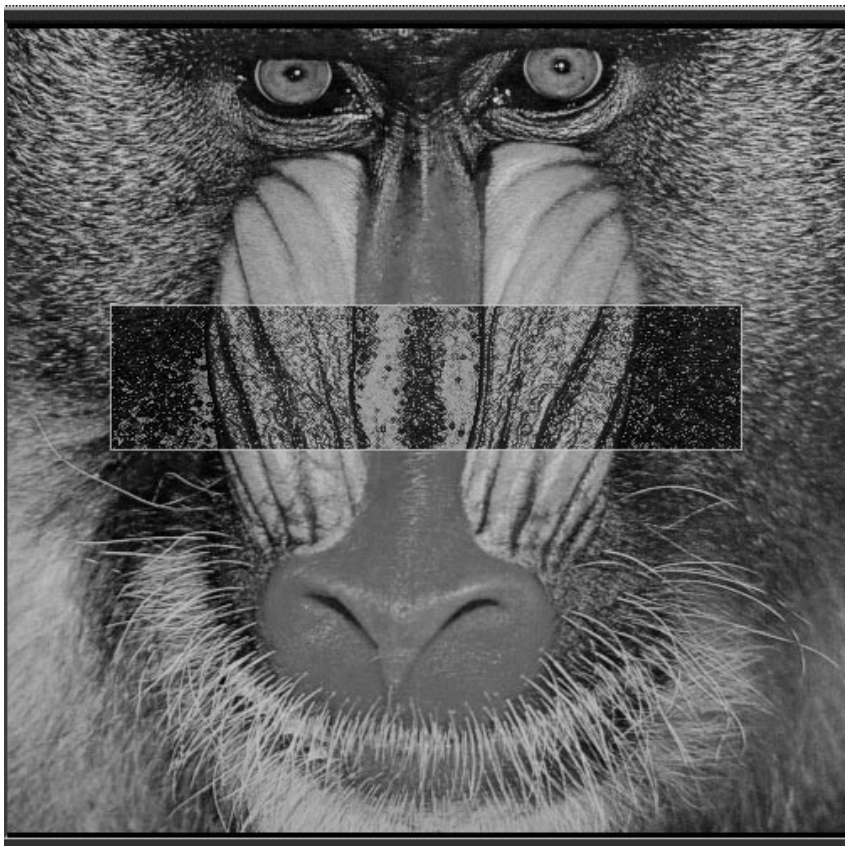


Figure 4-19 Subimage Area

Subimages are movable. Near the top of the Image Viewer control panel is the **Transform Subimage** button. When you press this, you can move the subimage around with the right mouse button, just as though it were a whole image. The network attempts to process the pixels under the moving subimage *while it moves* as though it were, instead of a moving "magnifying glass," a moving "image processing technique glass." Needless to say, when you are trying to do this, small subimages work faster than large subimages.

If you also switch on **Bounding Box** below the **Transform Subimage** button, then you can move the subimage, but the image processing will not occur in "real time" as the bounding box moves. Instead, the pixels will only be processed after you release the right mouse button and the subimage is in its new location.

You can't use shift-middle mouse button to resize subimages. Instead, just redraw the subimage area to be larger or smaller.

In Place/New Window

These two switches control where the processed image or subimage will be displayed. The default **In Place** causes the output image to be displayed in

place of the current image in the current scene. Subimages are displayed on top of their original image. Neither is permanent unless you press **Set Current Image**.

New Window creates a new, empty scene, and a new viewport looking onto that scene.

Where **In Place** puts the processed image in place of the original image, **New Window** puts the processed image into the new scene and viewport. It acts as a kind of "scratch" window.

If you process the whole image with **Zoom to Image**, the entire new image appears in the new window. The image will be the same size as the original image. If you process a subimage, then just the processed subimage rectangle appears in the new window (Figure 4-20).

Figure 4-20 Subimage in a New Window

The **New Window** scene can only hold one *processed* image. Every time you process an image or subimage, the result replaces the processed image in the **New Window** scene.

There can be only one **New Window** scene and viewport; you can't make a new one every time you enact a technique. Also note that you cannot perform image processing techniques on the contents of the **New Window** scene. This kind of serial processing of an image is discussed below.

Set Current Image: Performing Multiple Techniques on One Image

All of the image processing techniques applied to an image or subimage **In Place** are temporary. The processed version of the image is displayed in place

of the original image, but the Image Viewer keeps a copy of the original image.

Set Current Image makes the changes to the image permanent. The Image Viewer throws away its original copy of the image, making the modified version of the image into the "actual" image.

This is how you perform multiple image processing techniques on one image.

1. Selecting a processing technique and experiment with it on the current image until you have achieved the desired result.
2. Make the modifications permanent with **Set Current Image**. (Perhaps using **Duplicate Image** to save this intermediate version.)
3. Select a new processing technique, and continue the process on the new version of the image.

You cannot use **Set Current Image** to make the changes shown in the scratch **New Window** version permanent on the original image.

There is no way to "undo" the modifications to an image once they have been made permanent with **Set Current Image**. You can, of course, re-read the original image from disk, or re-execute the network that produced the original image to get a new copy into the Image Viewer.

Restore Current Image

Restore Current Image undoes the temporary changes that have been made to an image or subimage. The processed version of the original image that is being displayed in its place is discarded and the original image restored. Clicking the left mouse button anywhere in any Image Viewer viewport has the same effect.

Restore Current Image does not undo the effect of **Set Current Image**.

Raise Control Panel: Window Management

When one is doing image processing, there can get to be a lot of windows on the screen. The thing to realize is that there are usually just *two* control panels in active use: the Image Viewer's control panel, and the technique's control panel that holds the widget controls for the modules in the network. (If you picked one of the two techniques that produce geometries, you might also need to find space for the Geometry Viewer's control panel. If you picked the technique that produces a graph, you might also need to find space for the Graph Viewer's control panel.) Again, not all techniques produce a control panel.

When you invoke a technique with a control panel, that control panel is placed *over* the Image Viewer's control panel. The Image Viewer's control panel is still there (unless you explicitly remove it with **Close**).

There are two strategies to deal with the context switching that has to go on between the technique's control panel and the Image Viewer's:

- Use your window manager to move the technique's control panel (and any other control panels) somewhere else on the screen, perhaps even partially off the screen. The Image Viewer control panel becomes unobscured. Switch contexts by just moving the mouse cursor back and forth between the control panels.
- Alternatively, switch between the two control panels, having first one then the other "on top." When the technique's control panel is on top, raise the Image Viewer's control panel by pressing **Data Viewers** and selecting Image Viewer. When the Image Viewer is on top, raise the technique's control panel by pressing this **Raise Control Panel** button.

Select Processing Technique

As described earlier, this button produces a browser that displays the sample image processing technique options (Figure 4-16). Though it is the last button on the submenu, it is usually the first item selected. Table 4-2 summarizes what the image processing techniques actually do. The techniques provided are samples; you can design your own image processing techniques. This is described in the next section.

For more information on each technique, including what the techniques dials, slider bars, and other manipulators do:

- See the appropriate module man page in the *AVS Module Reference* manual.
- Refer to the online module man pages. These are accessible by clicking the right mouse button on the module's icon in the Network Editor. This produces a "Module Editor" window. Press the **Show Module Documentation** button to see the man page.

The technique networks themselves are stored in the directory */usr/avs/networks/iview.*

Table 4-2 Sample Image Processing Techniques

Technique	Control Panel?	Produces	Description
add background	yes	image	blend image with a shaded backdrop
clamp	yes	image	set pixel values <min = min, and >max = max
contrast stretch	yes	image	removes low-level noise or increases image contrast

Table 4-2 Sample Image Processing Techniques

Technique	Control Panel?	Produces	Description
convolve	yes	image	apply various convolution filters to image
display histogram	yes	graph	plot distribution of RGB pixel component values
edge detect	no	image	find edges in an image (**sobel** module)
geometry mesh	yes	geometry	change image into a surface where height represents pixel value
geometry contour	yes	geometry	draw contour lines connecting equal pixel values
gradient shade	yes	image	create pseudo-3D image using shading where values change rapidly
histogram stretch	yes	image	enhance low-contrast or uneven pixel distribution images
local area ops	yes	image	change pixels to be like neighbouring pixels with various filters
luminence	no	image	create black and white version of an image
mirror	yes	image	reverse image about X or Y axis
pseudo color	yes	image	repaint RGB pixel components any color with colormap editor
threshold	yes	image	set pixel values <min and >max to 0

Limitations

With **In Place** selected, the Image Viewer does not correctly handle modules that change the extents (size) of an image, such as **crop** and **downsize**. When you press **Zoom to Image**, it places the new, smaller image *on top* of the existing image in a kind of collage, instead of replacing it. This is true even if you **Set Current Image**.

There is a workaround. You can select **New Window** to receive the new, smaller image. With the new window the current window, write the cropped or downsized image to disk with **Save Image**. You can then read it back into the Image Viewer with **Read Image**.

You also need to be a bit careful if you are simultaneously using networks created by the Image Viewer's Processing Technique browser with networks you yourself have created or read into the Network Editor's workspace area.

Two independent parties, the Image Viewer and you, are trying to share the same Network Editor workspace. The Network Editor does not distinguish between you. Any Network Editor function that you perform (e.g., **Clear Network**, **Write Network**) is going to function on **all** networks in the workspace, both yours and the Image Viewer's. The Image Viewer will not properly handle saved Image Viewer networks that also contain "peer," independent networks. The Image Viewer can get confused if a **Clear Network** function has deleted the **IV read image** and **IV write image** modules that it needs to import/export data to the Network Editor.

You can use multiple networks from different sources in the Network Editor workspace, just do not perform global functions on them.

Defining Image Processing Techniques

It is possible to create your own directory of image processing technique networks.

When the Image Processing submenu's **Select Processing Technique** button is pushed, the Image Viewer *first* looks in the *current directory* for a file called **image_tech.lst**. The "current directory" is the current directory for the terminal emulator window that invoked AVS. Not finding a local **image_tech.lst**, it proceeds to use the system default in */usr/avs/networks/iview/image_tech.lst*.

Thus, to create your own image processing technique networks:

1. Create a directory to hold the technique networks and utility files. *You will have to invoke AVS with this as your current directory*. The Image Viewer does not automatically look for techniques in your HOME directory, nor according to any command line or *.avsrc* file option.
2. Copy the file */usr/avs/networks/iview/base.net* into your techniques directory. **Select Processing Techniques** uses this file to bring up the **IV read image** and **IV write image** modules when first invoked.
3. Create a *image_tech.lst* file. Use the one in the system directory */usr/avs/networks/iview/image_tech.lst* as a model. Here is what the first few lines of that file look like:

```
"add background"    "back"
"clamp"             "clamp"
"contrast stretch"  "contrast"
                .
                .
                .
```

   The left column is the alphanumeric text string that will appear in the Technique Browser window.

   The right column is the name of the network file that contains the image processing network, *without* its **.net** suffix. These network

files must also be in the technique directory; one cannot give system file directory specifications.

The *image_tech.lst* file must also be in your technique directory.

4. Create the technique networks using the AVS Network Editor. Use its **Write Network** option to write the networks to your technique directory.

   The **IV read image** module must be at the top of your network to retrieve an image from the Image Viewer. **IV write image** must be at the bottom of the network to return the image to the Image Viewer.

   These modules do not appear in the Network Editor's module palette. Instead, either use the Network Editor's **Read Network** function to load the file */usr/avs/networks/iview/base.net*, or create an instance of the modules by invoking the Image Viewer's **Select Processing Technique** option and picking any technique. Then enter the Network Editor. (You are likely to be doing this anyway, as you construct and test a network.)

A system administrator can expand the list of image processing techniques available to everyone by creating networks for them in */usr/avs/networks/iview* and editing its *image_tech.lst* file.

Labels Submenu

The **Labels** menu selection (Figure 4-21) provides access to the Image Viewer's annotation text facility. You can attach one or more *labels* to any image. Each label consists of a single line of text. As you manipulate the image—move it, resize it, temporarily hide it, permanently delete it, etc.—the image's label(s) react accordingly.

You have considerable typographic control, with a wide range of fonts, type styles, sizes, and colors to choose from. You can also control the position of each label relative to its associated image; one alternative is to have the label become a **title**, which always appears at the same location in the scene, no matter how the image is transformed.

Current Label: Creating Labels

To create a label, first make sure the image to be labeled is the current image. If necessary, click on the image with the left mouse button. Then, click the **Labels** menu selection to bring up the Labels submenu.

Place the cursor in the empty box below **Current Label**, and type any string of printable characters. Use **Backspace** (erase last character) and **Ctrl-U** (erase entire line) to make corrections.

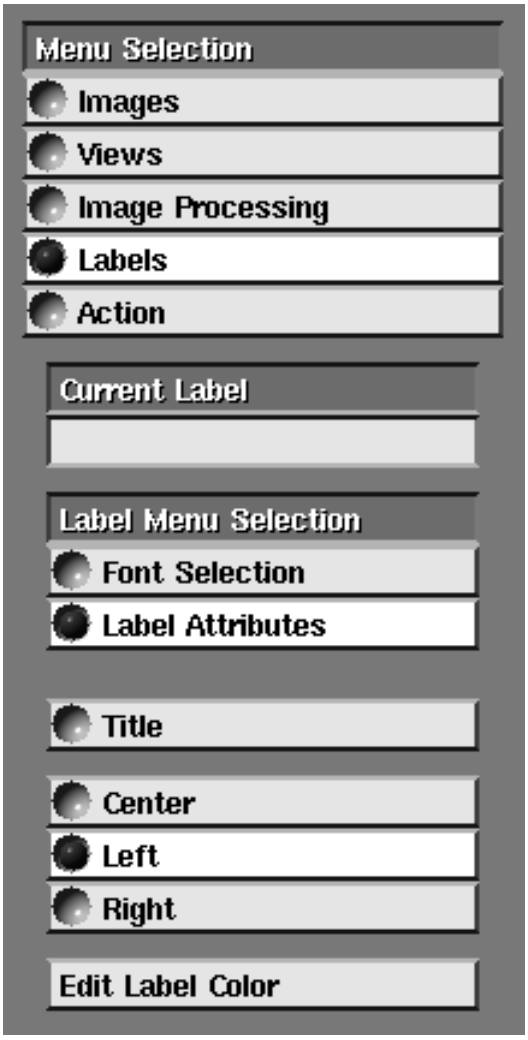


Figure 4-21 Labels Submenu

When you've finished the label, press **Return** or move the mouse cursor out of the **Current Label** box. When you do so, the label appears on the current image.

To create additional labels for the same image, select the image again by clicking on it with the left mouse button. This clears the **Current Label** box. (In addition, you may want to check that the Image Title Bar shows the image and its name.) As before, type in a text string and press **Return**.

Editing a Label

To change the text of a label, first click on the label with the left mouse button to make it appear in the **Current Label** box (Figure 4-22). Then move the cur-

sor into the box and type the changes. As when you first create a title, **Backspace** erases the last character and **Ctrl-U** erases the entire label.

Figure 4-22 Current Label Typein

Picking and Moving a Label

Each of an image's labels is "attached" to a particular point on the image. Initially, this *base point* is the center of the image. You can move an existing label so that its base point is at a different X-Y location.

Click and hold down the left mouse button on the label. Drag the cursor to any other location, then release the button.

Title: Making a Label Into a Title

It is sometimes desirable to have one or more labels that are associated with a scene, but which don't move around the screen as the images are transformed. Such labels are called **titles**. For instance, you might want a title string for a scene to appear in the upper left corner of the window wherever the images are displayed. You can change any regular label into a title label by clicking the **Title** selection.

A title label "lives" in the scene's X-Y coordinate system. You can change the position of a title label using the left mouse button.

Label Menu Selection

The annotation text facility includes a two-level function menu, which allows you to customize the appearance of each label. The top-level choices, **Font Selection** and **Label Attributes** are always visible. Font Selection Submenu

The submenu for **Font Selection** includes a list of the available fonts, a **Bold/Italic** selection, and a **Font Height** slider bar.

Fonts

The set of font radio buttons selects the X Window System font to be use for the label.

The fonts that appear may vary from system to system.

Figure 4-23 View with Labels and Titles

Bold
Italic

Selects the type style. You can click both of these choices to produce a bold-italic label. (Not all systems support bold and / or italic fonts.)

Label Height

Selects the point size of the label. Labels do *not* scale continuously; instead, AVS makes best use of the available X Window System fonts. As you move the slider to indicate a larger or smaller size (using any mouse button, by clicking or by dragging), the label size changes when a different font provides the closest fit. Label Attributes Submenu

The submenu for **Label Attributes** includes the following choices:

Title

Makes the current label into a title, whose position is fixed within the scene's coordinates. See "Making a Label Into a Title" above.

Center
Left
Right

Specifies which part of the label is placed on the *base point*. Initially, it is the bottom center. The alternatives are the lower left corner and the lower right corner.

Edit Label Color

Creates an RGB-HSV color editor with which you can specify the color of the label.

Action Submenu: Flipbook Animation

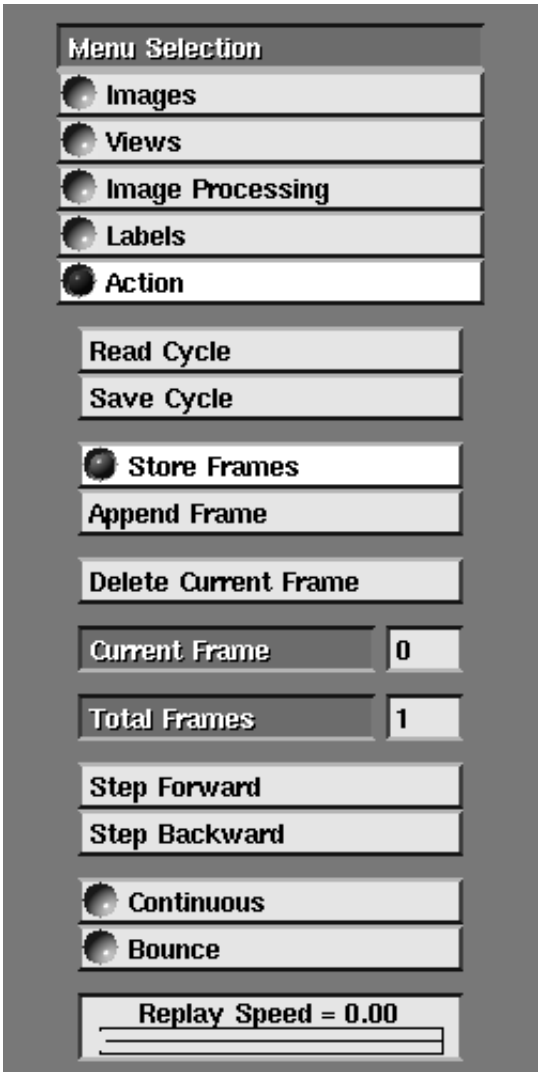


Figure 4-24 Action Submenu

The **Action** submenu (Figure 4-24) is used to create and play back simple flipbook animations of images. You can save these animations to a disk file and read them in again at a later session for replay. Because any data displayed in an AVS output window is either a geometry, a pixmap, or an image; and geometries and graphs are always convertible to images (**geometry viewer** and **graph viewer** modules), and pixmaps convertible to images (**pixmap to image** module), you can animate and save any sequence of displays produced by AVS. All it takes is disk space.

It is helpful to understand what **Action** *won't* do:

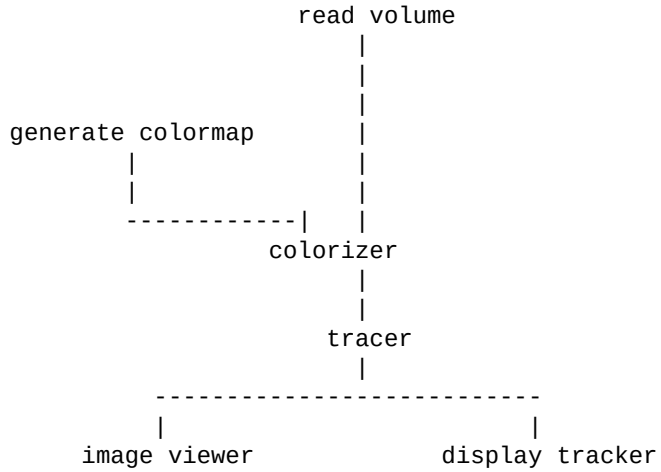
- You cannot animate scenes. For example, you cannot have several images in a scene and create an animation of them swirling about one another.

(This is possible, but you have to use the Image Viewer CLI interface, not **Action**.)

- You cannot create animations of changes to images that have been read into the Image Viewer *with the Read Image button*. For example, you cannot press **Read Image**, read in *mandrill.x*, then create an animation of the mandrill's image as its contrast fades, or as it zooms in to gigantic proportions. (Again, this is possible, but you have to use CLI.)

What **Action** can do is create an animation out of a series of images flowing into an **image viewer** *module* from another *module* through an AVS network.

Here is one possible network:



There are many opportunities for animation here. By adjusting the controls on **display tracker**, you could animate the volume rotating in "space." Adjusting the **Alpha** (opacity) controls on **tracer** itself would animate the hydrogen atom going from mostly opaque to nearly transparent. Adjusting the **generate colormap** module's colormap editor would animate the volume data changing color and / or opacity.

To begin an animation, turn on the **Action** submenu's **store frames** button. This is like pressing a record button. Thereafter:

> *Each time* **tracer** *outputs a new image of the hydrogen atom, this image flows into the* **image viewer** *module and is added as a new page in the flipbook.*

A network to capture geometry animations is shown at the end of this section.

Size: A Caution

Images get large quickly. Each pixel is one 32-bit word. A 512x512 image (roughly 6"x6" on the screen) takes up about 1 megabyte of memory. A flip-

book animation consisting of 20 images thus takes up 20 megabytes of memory. Be aware of the memory size and swapspace limitations of your machine.

To save this 20 frame animation as a cycle for future replay will also require 20 megabytes of memory. When flipbook images are stored, they are stored in their original size. For example, if a 256x256 image enters the **image viewer**, where it is then rescaled to 512x512, it is stored in its original 256x256 size.

Store Frames

Store Frames turns "image recording" on and off. It also causes the remainder of the **Action** control widgets, normally invisible, to appear. It is off by default. With **Store Frames** on, each image flowing into the **image viewer** module is added sequentially to the flipbook. The count shown under **Total Frames** will increment. When you have captured all the frames you want to, turn off **Store Frames**.

Append Frame

Append Frame adds single images, *already in* the Image Viewer window, to the end of the image cycle. It is the oneshot "add just this frame." Frames are always added at the end of the cycle. (To change the order of frames, you can edit the ASCII cycle file created by the **Read Cycle** button described below.)

Total Frames

The **Total Frames** counter counts the number of frames that are stored in the cycle. It increments as images are added, either through **Store Frames** or **Append Frames**.

Current Frame

Current frame shows which of the numbered frames is presently being shown when you play cycles of images back. Note that this counter starts at zero, where **Total Frames** starts at 1.

Step Forward/Step Backward

These two controls play back cycles of images one frame at a time. As you move forward or back, the **Current Frame** indicator updates the number of the frame being shown.

Continuous

Continuous is an on/off switch that plays the cycle of images continuously, instead of stepping through them one-by-one manually. It plays them in a continuous cycle, e.g., 9 10 11 12 0 1 2. To turn replay off, press **Continuous** again.

Bounce

Bounce also plays the cycle of images continuously, but in a different order than **Continuous**. With **Bounce** toggled, frames count up then count down, e.g., 9 10 11 12 11 10 9. To turn **Bounce** replay off, press it again.

Replay Speed

This slider bar widget speeds up or slows down the rate at which images are replayed. (This is also affected by the power and load of your machine.) Moving to the right slows the replay down.

Delete Current Frame

This removes the current frame from the cycle. The total number of frames, and the sequence shown in **Current Frame** are immediately renumbered. To delete an entire cycle, just click on **Delete Current Frame** repeatedly.

Save Cycle

Save Cycle saves Image Viewer flipbook animation cycles to disk where they can be retrieved at a later date. **Save Cycle** raises a typein window that prompts for a filename in the defined DataDirectory. To save the cycle elsewhere, type in a full directory and filename specification. As always, **Ctrl-u** deletes the entire typein line, **Backspace** deletes the previous character, and **Enter** or **OK** completes the entry. AVS automatically appends the **.cyc** file suffix that denotes a cycle.

Note: On systems with filename length restrictions, the typein may include a caution advising you to make sure that the total filename length does not exceed your system's limit.

The cycle is not one file. The file *name*.**cyc** defines the cycle. It is an ASCII file written in Image Viewer CLI commands. The images that make up the cycle are automatically stored in separate files in the same directory, one file for each frame. The file *name*.cyc.x is the original image. They have the same

name as the primary cycle file, a midfix (e.g., .cyc0, .cyc1, .cyc2, etc.) that denotes the image's sequence in the cycle, and the standard image file suffix, **.x**.

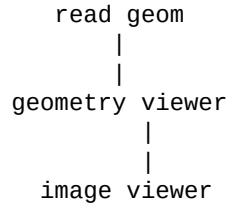
As noted earlier under "Size: A Caution," be aware of the size of the images in the cycle and the number of images and be careful that there is enough available disk space to hold them.

Read Cycle

Read Cycle raises a file browser displaying the defined DataDirectory. Only directory files and files with the **.cyc** suffix are displayed. (*.x*, and *.ims* files, if present, are also displayed.) By clicking on the primary cycle file, the entire cycle is read into the Image Viewer. Because this could consist of many large files, it may take some time. **Read Cycle** also puts up the **Action** animation controls. You can now replay and edit the cycle.

A Network for Geometries

This network uses the right image output port of the **geometry viewer** module to create an animation of changes occurring in the Geometry Viewer with the Image Viewer.



At the top of this network is the **read geom** module. However, this could be replaced with any set of modules (e.g., **read field, isosurface**) that eventually produces a geometry.

Image Viewer Command Language Interpreter

It is possible to drive the Image Viewer with commands rather than the X display interface. The commands can be either typed in interactively from a terminal emulator window while AVS is running, or they can be read from a script file.

This opens many possibilities:

- One could create scripts that animate the Image Viewer itself, not just the network-produced images within it.
- One could create demonstration, illustration, and test scripts.

- One could create scripts that batch-process images.

To run AVS with the Command Language Interpreter (CLI) active, type this:

```
avs  -cli  other-options
```

This starts AVS as usual, but also starts the CLI command line interpreter in the invoking window. (You might have to press carriage return to get the **avs>** prompt.)

To get a list of the Image Viewer CLI commands, type the following:

```
avs>  help Image
```

This produces a list of the many Image Viewer CLI commands. To get help on an individual commands, type "help" plus the command name:

```
avs> help image_create_scene
image_create_scene  Create a new scene with scene location and size
      Usage:  image_create_scene <xlocation ylocation width height>

 avs>
```

To see sample versions of Image Viewer CLI files, there are several places to look:

- Create a fairly complex scene with multiple viewports, images, transformations, etc., then save it out with **Save Scene**. This *.ims* scene file is written in Image Viewer CLI.
- Look in the directories */usr/avs/demosuite/General/Image* and */usr/avs/demo/image_viewer*.

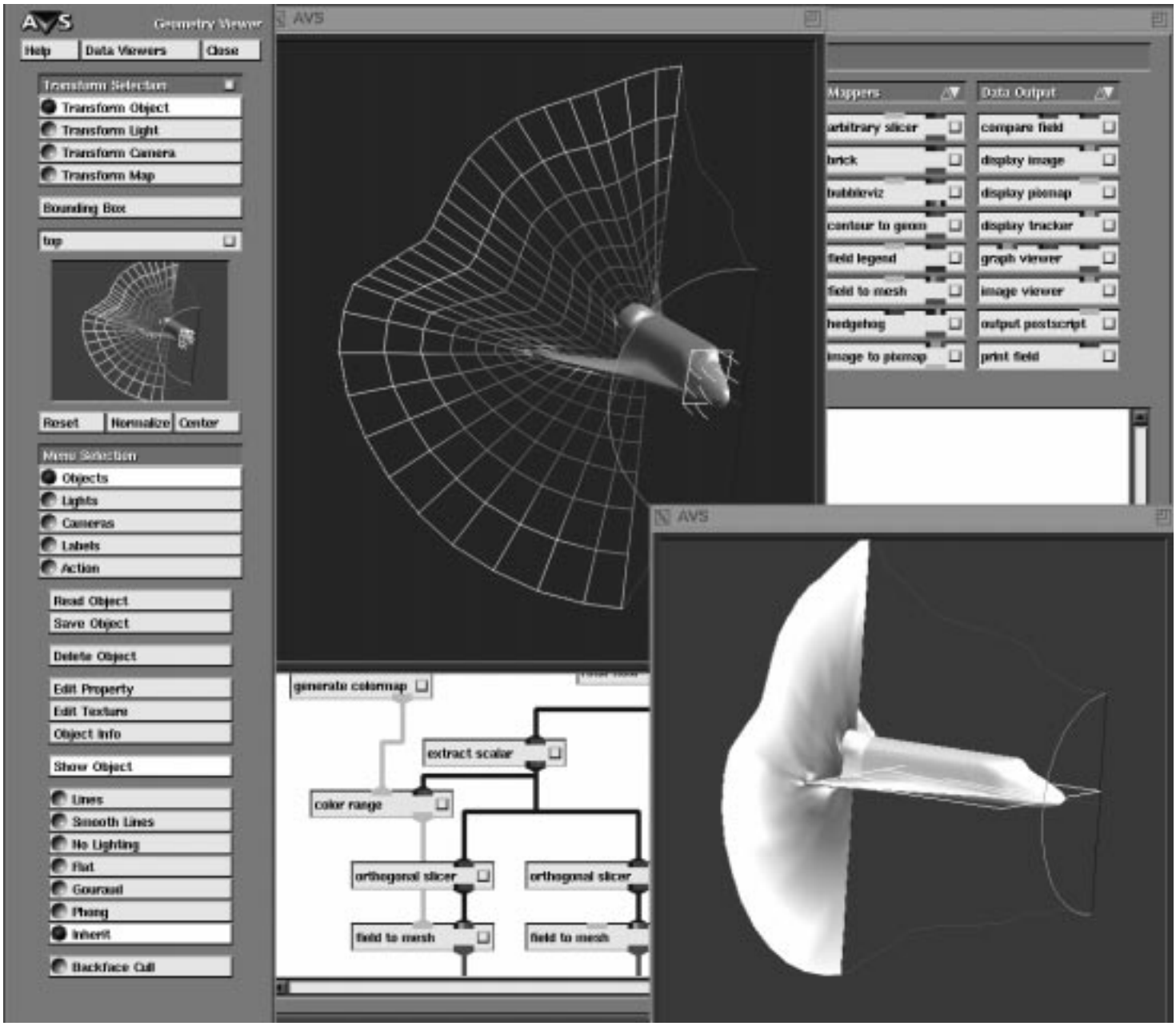
  Script files have a **.scr** suffix.

  Scripts can be run interactively. The files in *demosuite/General/Image* are accessed through the AVS **Demos** facility on the **Applications** menu. Those in *demo/image_viewer* are accessed through the Help Browser. Click on **Help**, then click **Help Demos**. This puts up a demo browser.

  You can also run scripts interactively through the CLI interpreter as follows:

  ```
  avs> script -play filename
  ```

The Command Language Interpreter and the Image Viewer set of CLI commands are documented in detail in the "Command Language Interpreter" chapter of the *AVS Developer's Guide*.


AVS
Geometry Viewer
Help
Data Viewers
Close
Transform Selection
Transform Object
Transform Light
Transform Camera
Transform Map
Bounding Box
top
Reset
Normalize
Center
Menu Selection
Objects
Lights
Cameras
Labels
Action
Read Object
Save Object
Delete Object
Edit Property
Edit Texture
Object Info
Show Object
Lines
Smooth Lines
No Lighting
Flat
Gouraud
Phong
Inherit
Backface Cull
AVS
Mappers
Data Output
arbitrary slicer
brick
bubbleviz
contour to geom
field legend
field to mesh
hedgehog
image to pixmap
compare field
display image
display pixmap
display tracker
graph viewer
image viewer
output postscript
print field
generate colormap
extract scalar
color range
orthogonal slicer
orthogonal slicer
field to mesh
field to mesh
AVS

CHAPTER 5

GEOMETRY VIEWER SUBSYSTEM

Introduction

The AVS Geometry Viewer subsystem is an interactive tool with which you can manipulate and view one or more 3D objects of the fundamental AVS data type *geometry.*

The Geometry Viewer exists in two forms. It is an AVS subsystem accessible from the main menu and from any other AVS subsystem through the **Data Viewers** pulldown menu. It also exists as the **geometry viewer** module in AVS networks.

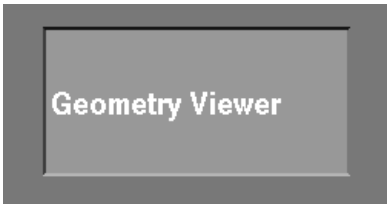


Figure 5-1 Geometry Viewer Subsystem

Figure 5-2 Geometry Viewer Module

It is also possible to drive the Geometry Viewer from the AVS Command Language Interpreter (CLI). You can type Geometry Viewer CLI commands to the CLI and interactively view the results, you can compose a script of CLI Geometry Viewer commands that will run automatically, or you can write a module that sends Geometry Viewer CLI commands to the Geometry Viewer through the AVS kernel. The CLI interface is discussed in the "Command Language Interpreter" chapter in the *AVS Developer's Guide.*

A *geometry* is a collection of points in 3D space, along with additional information (typically, indicating connectivity). The geometry defines a simple or complex 3D object, with the specified points as its vertices. (A geometry can also include a color, transparency value, and / or normal for each vertex.)

The geometries can enter the Geometry Viewer from two sources:

- You can read *.geom* format files directly into the Geometry Viewer through the **Read Object** button on the **Objects** submenu. These *.geom* format files were either created by a program making calls to the *libgeom* library, or were saved to disk during an earlier Geometry Viewer session. The geometries can be pure geometries (*.geom* files), or included as part of higher abstractions called **objects**, or as components of an entire 3D **scene**. Objects and scenes are stored in CLI script files (*.scr* files).

  The Geometry Viewer can also read some external file formats directly (Wavefront, Movie.BYU, Brookhaven Protein Data Bank, etc.) and transparently convert them to *.geom* files. See the "AVS Geometry Filters" section of the "Importing Data Into AVS" chapter for a list of supported external file formats.
- Geometries can flow into the **geometry viewer** module from an AVS network. (See Figure 5-3 and Figure 5-4.) Most AVS mapper modules (**scatter dots, isosurface, arbitrary slice, hedgehog, ucd to geom,** etc.) create geometries as their visualization representation of numeric data. This is signified by their red (geometry) output ports. For example, the **isosurface** module takes the numeric data in a 3D field, then uses the "level" value you give it to create a 3D contour surface through the volume. This 3D contour is output as a geometry of verticies in 3D space connected together with surfaces. Note that *the numbers from which the geometry was derived are not present in the geometry itself.*

  The **geometry viewer** module's red geometry input port can accept connections from multiple modules. The geometries from all the modules will be combined together into one **scene**.

Sample geometries, objects, and scenes are found in the directory */usr/avs/data/geometry*. Sample scripts that produce networks with geometries are accessible through the Help Panel's **Help Demos** browser.

Renderers

The Geometry Viewer supports multiple **renderers**: both a **software renderer** that implements its own graphics rendering techniques in software, outputting an X image; and a **hardware renderer** that uses the workstation platform's underlying software and hardware graphics facilities to produce its screen rendering. (On systems without hardware rendering facilities, only the software renderer is implemented.)

In general hardware rendering, if present, will have superior performance. However, the software renderer implements certain rendering techniques (2D and 3D texture mapping, transparency, arbitrary clipping planes, volume ren-

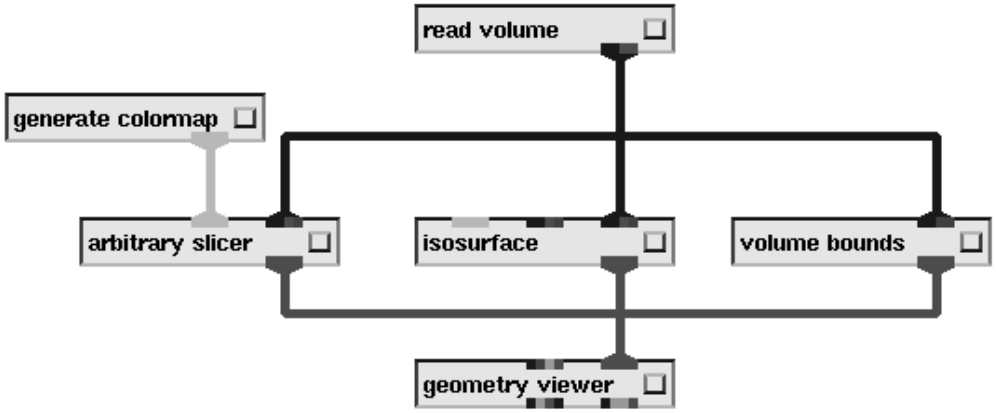


Figure 5-3 A Network that Produces Three Geomtries

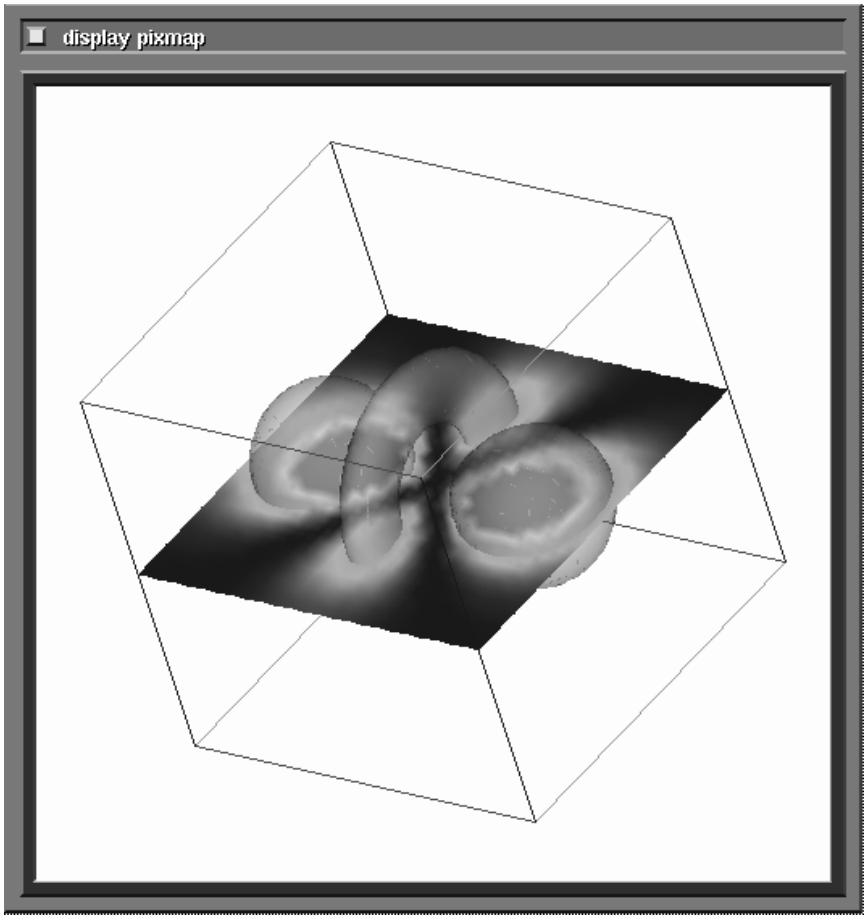


Figure 5-4 The Three Geometries Combined in One Scene

dering, outline gouraud) that are often not supported by hardware renderers, but which are highly effective for scientific visualization.

If you need these rendering techniques (for example, if you are using modules such as **brick**, **clip geom**, or **volume renderer**) or if you require transparency, you can switch between hardware and software rendering while in the Geometry Viewer using the renderer selection menu under the **Cameras** submenu. If a renderer does not implement a particular rendering technique, the text in its menu entry is drawn in gray rather than black.

In addition, various command line options and *.avsrc* file keywords (**-nohw**, **-renderer**, **NoHW**, and **Renderer**) control which renderers will be initialized, and which will be the default when the Geometry Viewer's first view window appears. You may need to use **-nohw**, for example, if you are running AVS from an X terminal that has no hardware rendering support.

The renderer selection is on a per-camera basis. Each camera has a current renderer that it uses to draw the objects in that window. The renderer can also be set through the CLI.

Some platforms may support additional renderer options.

Entering the Geometry Viewer

The Geometry Viewer can be entered in four ways:

From the shell directly

The following command line invokes the Geometry Viewer automatically when AVS starts execution:

```
avs -geometry
```

When you start AVS and the Geometry Viewer in this fashion, you cannot transfer to any other AVS subsystem. See the "Starting AVS" chapter for

additional command line options that affect the way the Geometry Viewer is invoked.

From the main menu

You can start the Geometry Viewer from the AVS main menu.

From another subsystem

At the top of each of the four major AVS subsystem control panels (Image Viewer, Graph Viewer, Geometry Viewer, Network Editor) is a button titled **Data Viewers**. Position the mouse cursor over **Data Viewers**, then press *and hold down* any mouse button. A pop-up menu appears. Still holding the mouse button down, roll the cursor down the pop-up menu until "Geometry Viewer" is highlighted, then release the mouse button. This calls up the Geometry Viewer's control panel. If you transfer to the other subsystems, then return to the Geometry Viewer, the Geometry Viewer's control panel will remain in the state that you left it.

You can move the Geometry Viewer's control panel with your window manager. This is convenient if there is more than one control panel (e.g., the Network Editor's) you need to use at one time.

In a network

You can include the **geometry viewer** module in an AVS network. If you click on a **geometry viewer** module's "dimple" with the left mouse button, it calls up the Geometry Viewer control panel.

Spaceballs and Dialboxes

It is possible to have either a dialbox or spaceball input device automatically connected to the Geometry Viewer's rotation, translation, and scaling transformations. When you first start AVS, add one of the following to the *avs* command line:

```
-dials devicefilespec
-spaceball devicefilespec
```

Replace *devicefilespec* with the serial communications port to which the dialbox or spaceball is connected (e.g., */dev/tty2*).

Alternatively, add either of the following to your *.avsrc* startup file:

```
DialDevice devicefilespec
SpaceballDevice devicefilespec
```

These can also be specified as the DIALS or SPACEBALL environment variables. The details of using the dialbox or spaceball are discussed below in the "Transformations" section.

Dialboxes and spaceballs may not be supported on all platforms. See the release notes that accompany AVS on your platform.

Leaving the Geometry Viewer

If the Geometry Viewer was invoked from the shell command line as "avs -geometry" then at the top of its main control panel will be a button labeled **Exit**. Press **Exit** with any mouse button to return to the system shell.

If the Geometry Viewer was entered from the main AVS menu or through the **Data Viewers** pop-up menu from another subsystem, then there will be a **Close** button at the top of its main control panel. **Close** is not really an exit button. **Close** simply takes down the Geometry Viewer's control panel; one could get a similar effect by using the system's window manager to iconify the control panel. When you later re-enter the Geometry Viewer, the control panel is in the same state that you left it. The only real way to exit the Geometry Viewer is to exit AVS altogether from the main menu.

If the Geometry Viewer was invoked as the **geometry viewer** module, then there is still only one Geometry Viewer, even though there may be multiple **geometry viewer** modules. The modules are associated with individual Geometry Viewer *scene* windows, not with the Geometry Viewer itself. Throwing away a **geometry viewer** module deletes a scene window.

Scenes, Objects, Lights, and Cameras

The Geometry Viewer creates a three dimensional stage called a **scene** within which you view geometries. You can create multiple scenes with the **Create Scene** function under the **Cameras** submenu, or by using multiple **geometry viewer** modules in a network.

Each scene is composed of:

- A **world space**. **World space** is an unbounded three dimensional volume. Location in world space is measured by **world coordinates**. World coordinates are a single set of right-handed X, Y, and Z axes. When the Geometry Viewer first comes up, the world coordinate origin (0,0,0) is positioned in the center of the view window. The XY plane is parallel to the screen face. Positive Y is straight up, positive X is to the right, and positive Z is coming straight out of the screen toward you. World coordinates are also referred to synonymously as **scene coordinates**. You do not see world/scene axes unless you turn on **Axes for Scene** under the **Cameras** submenu. Then they will appear, unlabeled, extending from +5 to -5 in world space in the X, Y, and Z directions. From a conceptual point of view, world space is immobile.
- A collection of 3D **objects** assembled into world space. Objects have attributes, such as surface color, various light reflectance characteristics, and a rendering method (lines, Gouraud shaded, etc.). You can selectively hide objets so that they are temporarily invisible, although still part of the scene.
- A collection of **lights**, defined in the same world coordinate space. Each light can be a different color.
- One or more **view windows**, each of which provides its own view of the collection of objects, as they are illuminated by a collection of lights. Each view window is considered to be a **camera** viewing the objects.

  Different cameras can produce different views, because each can have its own position in world coordinates. In addition, cameras can vary in the way that they display an object (depth cueing lines, with a perspective projection, Z buffering to expose or hide hidden surface or lines, etc.), and the renderer, hardware or software, used to produce the picture.

Several scenes may be visible onscreen at the same time. You can manipulate the various view windows with Geometry Viewer functions such as **Create Camera**. You can also manipulate the windows with an X Window system window manager—they are just like any other windows.

Within each view window, you can **translate** (move) the objects, lights, and cameras; **scale** them (make them larger or smaller), and **rotate** them. These three operations are called **transformations**. Objects, lights, and cameras are collectively called **transformables**.

Note: In order to save *exactly* what you see in a view window, including the positioning of multiple objects, all of the lighting effects and colors, and all camera manipulations including **Perspective**, you must use **Save Scene**. New AVS users often spend some time composing an attractive scene, then make the mistake of using just **Save Object**, which saves none of these features.

Objects

Objects are read into world space with the Geometry Viewer's **Read Object** function under the **Objects** submenu, or flow into the **geometry viewer** module from another module.

Objects are organized into a **hierarchy**.

At the top of the hierarchy is a root object named simply **Top**.

All objects that you read into the Geometry Viewer, either through **Read Object** or that flow into the **geometry viewer** module, are children of the Top level object, and peers of each other.

The hierarchy is normally only these two levels—Top and everything else. Though you cannot do it from within the Geometry Viewer itself, it is possible to create multi-level object hierarchies with the Geometry Viewer's Script Language (see appendix), the CLI *geom_set_parent* command, and the **libgeom** library.

Each object:

- Has its own object coordinate system. The Top level object's coordinates are initially coextensive with the world coordinate axes. The child objects' coordinates are initially coextensive with those of the Top level object.
- Each object has **extents**. Its extents are the minimum rectangular volume in its own coordinates that the object occupies (max x - minx, max y - min y, max z - min z). The **Bounding Box** button will show you the current object's extents as you scale, rotate, or translate the object.
- Each object has a **center**. This is the point, in its own coordinate system, about which the object will **rotate**.
- Each object has a **name**. The name is of the form *name.number*. The name is set by the program that created the geometry. Geometry objects that come from modules are usually named after the module that created them. The Geometry Viewer appends a sequence number to the object name to distinguish between objects coming from two modules with the same name. You can rename objects using the Current Object Browser described below.

- Objects can also have surface **properties** such as color, transparency, reflectivity, etc. A simple geometry in a *.geom* file has no properties. It may have optional attributes such as vertex color, vertex transparency values, and **normals**—a vector perpendicular to each plane used to calculate the angle of reflection. These attributes are part of the geometry's data structure. Properties, on the other hand, exist only in the way the Geometry Viewer portrays an object. Object properties can be saved permanently in *.scr* and / or *.prop* property files. The surface properties determine how the object reflects light. By selecting various property combinations (**Edit Property** in the **Object** submenu), you can make an object appear to be made of different materials like metal or clay, or even semi-transparent glass.

You can transform the Top level object, which carries all its child objects along with it, within world coordinates. You can also transform individual objects *within* the Top level object's coordinate system.

Where Objects Appear in World Space

When you read an object into the Geometry Viewer with the **Read Object** button, it will be placed into the Top level object's coordinate system at its true location and size. The object *might* be all or partially outside the current view window. Pressing the **Normalize** button on the Geometry Viewer's control panel will bring it immediately into the view window.

When an object enters the Geometry Viewer (**geometry viewer** module) through a network from a mapper module such as **arbitrary slicer**, **volume bounds**, or **hedgehog**, etc., one of two things may occur:

- As with **Read Object**, the object will be placed at its true location and size within the Top level object's XYZ coordinate system. On occasion, part or all of the object may be outside the current view volume. (This is how, for example, the **arbitrary slicer** module behaves.) You can always get the entire object into the view volume by pressing the **Normalize** button.
- Some mapper modules elect to normalize an object within the view volume themselves. By making an additional call to the *libgeom* library, they instruct the Geometry Viewer to "rescale and translate" the *Top level object's* coordinate system so that the object will fit within the display viewport. If a mapper module makes this call, the object will *appear* centered in the display window and filling it automatically. If you bring up the Transformation Options panel (described below), and examine the **Absolute** scale and translation of the Top level object and the mapper's object, you will see that the mapper object is at its real size and location in the Top level object's coordinate space, and that the Top level object has been rescaled and its coordinate axes shifted within world coordinates to accommodate the object in the view.

  This approach has the advantage that you always immediately see all of an object. On rare occasions, this advantage can be a disadvantage if you are, for example, creating an animation. The scene in the display window

may jump each time a new geometry enters as the Top level coordinates are normalized.

If multiple modules are sending output to the **geometry viewer** module, and if any one of the modules makes this call, then all objects will appear within the view volume. The **volume bounds** module makes this call, for example, and will affect the output of **arbitrary slicer** even though **arbitrary slicer** does not make the call.

Note: In AVS 2, rectilinear and irregular data were mapped as described above. Uniform data, however, was not. Uniform data objects were always rescaled and translated to fit within a cubic volume in the Top level object's coordinate system extending from -1, -1, -1 to 1, 1, 1. This is no longer the case. Now uniform data lives in the space [0,0,0] to [maxX-1, maxY-1, maxZ-1].

Geometry Viewer Control Panel

Figure 5-5 shows the main Geometry Viewer control panel. Throughout the control panel and its submenus, press *any* mouse button to select *any* of the control buttons. The control panel is just another window on the screen. You can use your window manager to move it around, or even partially off the screen.

Top Control Bar

The three buttons at the top of the Geometry Viewer have the same meaning as in other AVS subsystems. The **Help** button invokes the Help Panel, with a selection of topics relevant to the Geometry Viewer. **Data Viewers** is a pop-up menu that brings up the control panels of the other subsystems. Note that if you use **Data Viewers** to switch to another subsystem, the Geometry Viewer's control panel is not taken down from the screen, it is merely obscured by the new control panel. You could use your window manager to move it to another part of the screen and have multiple control panels showing at once. **Close** unmaps the Geometry Viewer control panel from the screen. It does not "exit" the Geometry Viewer.

Transform Selection Area

There are four transformables in the Geometry Viewer: objects, lights, cameras, and 2D texture maps. (**Note:** some hardware renderers do not support 2D texture mapping and won't display this menu entry. The software renderer does support 2D texture mapping.)

The four Transform Selection menu buttons select *which* class of transformable (objects, lights, cameras, or 2D texture maps) is going to be affected by a mouse button or Transformation Options panel selection. It is a transformation mode switch. (See Figure 5-6.)

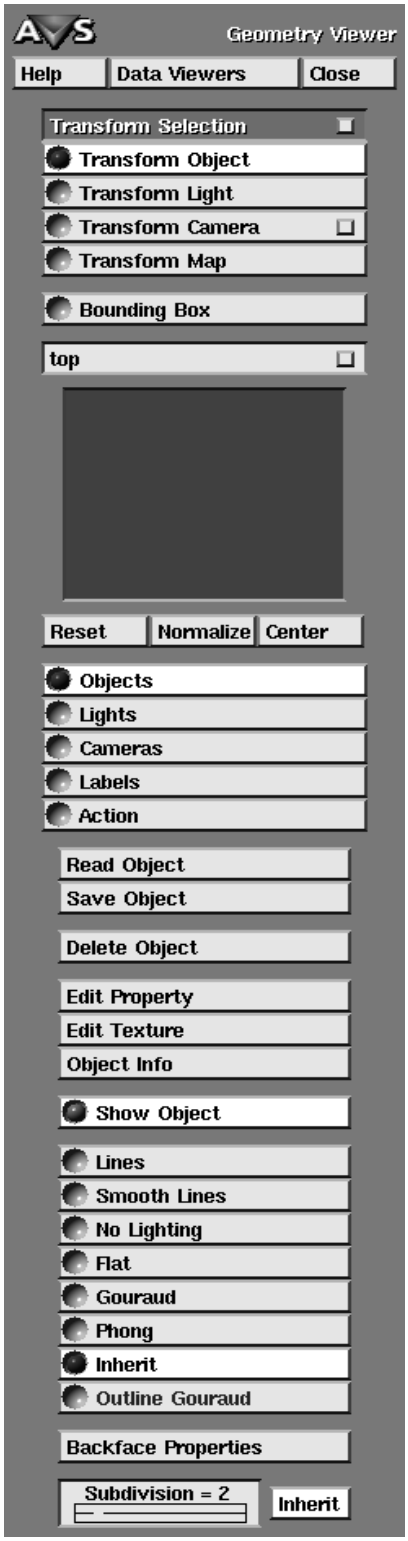


Figure 5-5 Geometry Viewer Control Panel

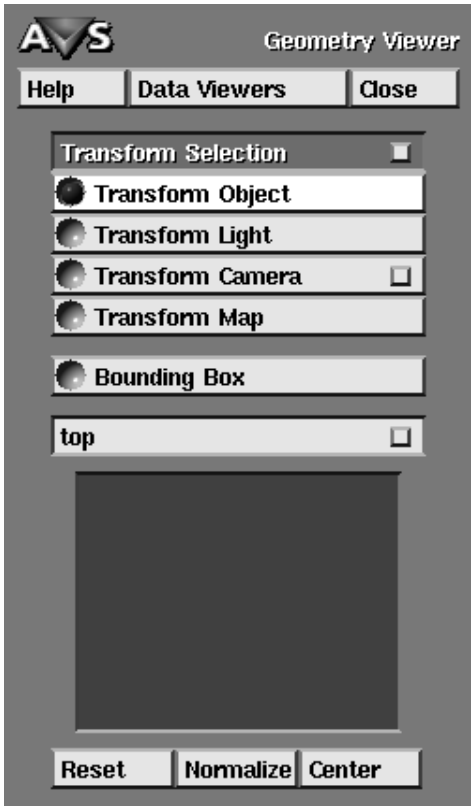


Figure 5-6 Transformation Selection Area

There are two main styles of transforming objects:

- Direct manipulation transformations performed with the workstation's mouse buttons in concert with the main keyboard's **Shift** key. In this type of transformation, you simply make the transformable the "current transformable," then point at it with the mouse cursor, press the correct mouse buttons, and "move" the object around the view window. This direct manipulation movement is highly interactive.
- **Precise** object transformations performed with the Transformation Options typein panel, or using the keyboard's arrow keys. In this mode, you also make the transformable the "current transformable," then you type in to the Transformation Options panel the exact **Absolute** or **Relative** movement you want the object to make.

The two interaction styles are quite different. We deal with each separately.

Note: Cameras can be transformed in yet a third way—with the Camera Options panel—using an intuitive "From" and "At" style that is suitable for animation. This is discussed in the "Cameras" section below.

Mouse Transformations

Transforming Objects

To transform objects, click the **Transform Object** button (or, equivalently, press function key **F1** with the mouse cursor in the graphics window). (**Transform Object** is the default.) This sets the mouse buttons to perform the following functions:

Left Mouse

Selects a particular object, making it the current object. The selected object appears in the Current Object Indicator (the small window) in the control panel. Repeated clicks on the same object move up the object hierarchy, progressively expanding the selection. For example:

One click: Selects **wing**, part of one wing of the jet object.

Two clicks: Expands the selection to **jet**, the entire jet object.

Three clicks: Expands the selection to the entire Top level object, which includes the jet and, perhaps, several other objects.

Four clicks: Having reached the Top level, returns to **wing**, as selected with a single click.

Middle Mouse

A dragging action (holding down the middle mouse button, then moving the mouse) rotates the current object in 3D space. The object behaves as if it were attached to a **virtual trackball** (see Figure 5-7) whose center is at the center of the view window. The mouse cursor is attached to the part of the trackball that protrudes above the surface of the window.

This is the most common way to manipulate objects.

Middle Mouse with Ctrl key held down

Using the **Ctrl** key with the middle mouse button allows you to constrain rotations to one of the major screen axes: X, Y, or Z. As you move the mouse, the object will rotate around the axis with the largest degree of change for that particular motion. Moving the cursor vertically in the center of the screen will rotate around X, horizontally in the middle of the screen rotates around Y. Strict Z rotations are difficult to achieve but can be obtained with some practice. Use **Ctrl-**middle mouse button and move the mouse cursor in a circle around the object.

Middle Mouse with SHIFT button held down

A dragging action scales the object: dragging downward or to the left makes the object smaller; dragging upward or to the right makes the object larger.

Right Mouse

A dragging action translates the selected object in the plane of the window (that is, moves the object left-right and/or up-down).

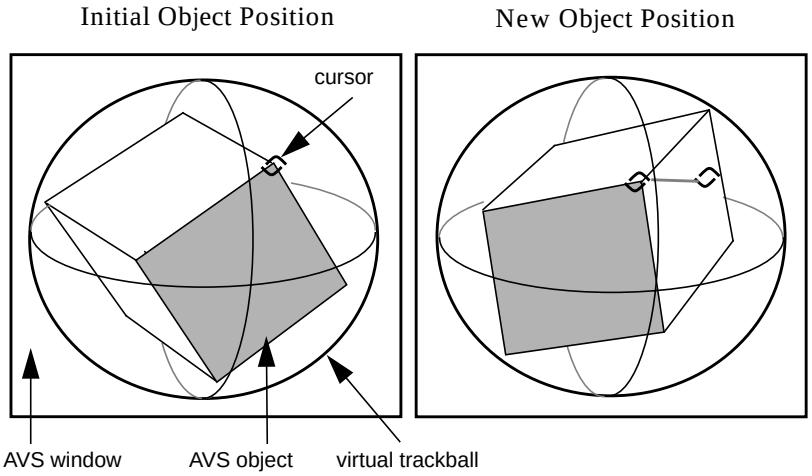


Figure 5-7 Rotating an Object with the Virtual Trackball

Right Mouse with SHIFT button held down

A dragging action translates the selected object perpendicular to the plane of the window (that is, moves the object in-out). Dragging upward or to the right moves the object away from your eye; dragging downward or to the left moves the object toward your eye.

Eventually, this translation may cause the object to cross the front or back clipping plane. This causes the object to (partially) disappear.

Note: You will not see the size of the object change when you transform it in the Z plane unless you turn on Perspective under the Cameras menu.

Right Mouse with Ctrl key held down

Using the **Ctrl** key with the right mouse button allows you to constrain translations to one of the major screen axes: X or Y. The axis with the largest change for a given motion will be the chosen axis. Horizontal motion translates in X, vertical motion translates in Y.

If you release the mouse button while the mouse is still moving (a kind of "flinging" motion), then the object will move continuously until you press any mouse button in the display window. This is called **track rolling**.

Transforming Lights

To transform lights, click the **Transform Light** button (or, equivalently, press function key **F2**). This sets the mouse buttons to perform the following functions:

Left Mouse

Still selects the current object. This button *always* selects objects, no matter what the Transform Selection is. To select the current light, click one of the

numbered boxes in the lighting panel (which appears when the Menu Selection is **Lights**).

Middle Mouse
A dragging action rotates the position (point light) or direction (directional light) of the current light using the "trackball" paradigm (see above).

Middle Mouse with Ctrl key held down
Constrains light rotations to the X, Y, or Z axis.

Middle Mouse with SHIFT button held down
Scales the **Show Lights** representation of the light source. For point and spot lights, this also translates the light source itself.

Right Mouse
A dragging action translates the selected light in the plane of the window. With (bi-)directional lights, this changes the position of the symbol that represents the light source (**Show Lights**), but has no effect on the light source itself.

Right Mouse with SHIFT button held down
(Applies to point and spot lights only) A dragging action translates the selected light perpendicular to the plane of the window.

If the scene is not drawn in perspective (see "Cameras" section below), this will have no effect.

Right Mouse with Ctrl key held down
Constrains light translations to one of the major screen axes: X or Y. The axis with the largest change for a given motion will be the chosen axis. Horizontal motion translates in X, vertical motion translates in Y.

Transforming Cameras

To transform cameras, click the **Transform Camera** button (or, equivalently, press function key **F3**). This sets the mouse buttons to perform the following functions:

Left Mouse
Still selects the current object. This button *always* selects objects, no matter what the Transform Selection is. To select the current camera (view), just click in the desired window. The current camera window will be surrounded with a red border.

Middle Mouse
A dragging action rotates the position of the current camera (the camera in the current window) using the "trackball" paradigm (see above).

Note that the object seems to rotate, rather than the camera.

Middle Mouse with SHIFT button held down
Scales the 3D view volume for the camera. Only objects within this view volume appear in the window.

Middle Mouse with Ctrl key held down
Constrains camera rotations to the X, Y, or Z axis.

Right Mouse
A dragging action translates the camera in the plane of the window.

Right Mouse with SHIFT button held down
Translates the camera perpendicular to the plane of the window. If the scene is not drawn in perspective, this will have no effect.

Right Mouse with Ctrl key held down
Constrains camera translations to one of the major screen axes: X or Y. The axis with the largest change for a given motion will be the chosen axis. Horizontal motion translates in X, vertical motion translates in Y.

The **Transform Camera** button has a "dimple." Pressing this dimple brings up the Camera Options panel. The Camera Options panel provides fine-grained control over the way that a camera portrays a scene including the camera's position, the direction it is viewing, the degree of perspective applied to the scene, depth cueing, and the position of clipping planes. The Camera Options panel is described in detail later in this chapter under the "Cameras" section.

Transforming Texture Maps

To transform the way that an image is texture-mapped to the surface of an object, click the **Transform Map** button (or, equivalently, press function key **F4**). (This selection only appears on systems with support for 2D and/or 3D texture mapping.) This sets the mouse buttons to perform the following functions:

Left Mouse
Still selects the current object. This button *always* selects objects, no matter what the Transform Selection is.

Middle Mouse
A dragging action rotates the position of the current texture (the texture-map image, if any, that is associated with the current object). using the "trackball" paradigm (see above).

Middle Mouse with SHIFT button held down
Scales the texture map.

Middle Mouse with Ctrl key held down
Constrains texture map rotations to the X, Y, or Z axis.

Right Mouse
A dragging action translates the texture map in the plane of the window.

Right Mouse with SHIFT button held down
Translates the texture map perpendicular to the plane of the window.

Right Mouse with Ctrl key held down
Constrains texture map translations to one of the major screen axes: X or Y. The axis with the largest change for a given motion will be the chosen axis. Horizontal motion translates in X, vertical motion translates in Y.

Transforming Labels

Labels are *objects*. For this reason, there is no **Transform Label** button on the Transformation Selection menu. See "Transforming Objects" above.

Precise Transformations

The **Transform Selection** menu bar has a "dimple." Clicking on this dimple produces the **Transformation Options** panel (Figure 5-8).

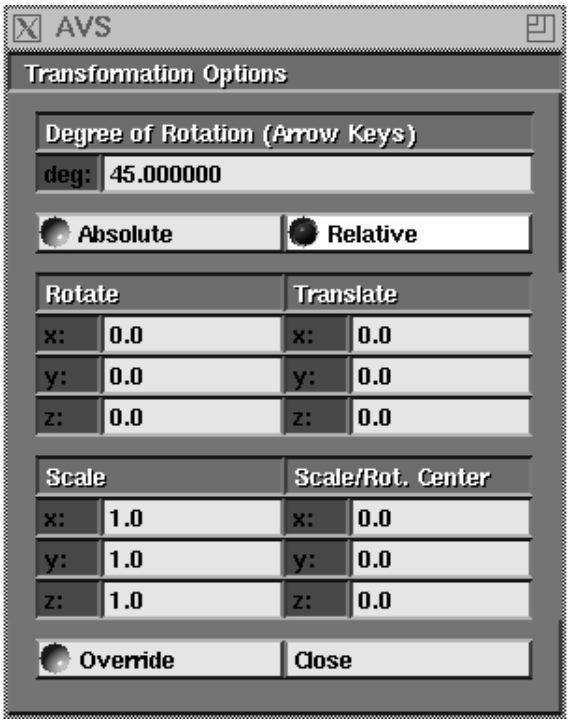


Figure 5-8 Tranformation Options Panel

The Transformation Options Panel allows you to transform objects, hierarchies of objects, lights, and cameras by precise values typed in to the **Trans-**

formation Options panel. The movements can be relative to the current position, or to an absolute position.

Degree of Rotation: Arrow Keys

You can rotate objects, hierarchies of objects, lights, and the camera with the keyboard's arrow keys. The degree of rotation is set with the typein at the top of the **Transformation Options** panel. The default is 45.0 degrees. The minimum is 0.0 degrees and the maximum is 360.0 degrees. Change the default value by moving the mouse cursor into the typein area and type a new value. As with all typeins, **Ctrl-U** deletes the entire line, and **Backspace** deletes the previous character. You do not have to press **Enter** for the new value to take effect. (However, you *do* have to move the mouse cursor into the display window before the arrow keys work.)

Left and Right Arrow Keys
The left and right arrow keys rotate the current transformable about its Y axis.

Up and Down Arrow Keys
The up and down arrow keys rotate the current transformable about its X axis.

Shift Left and Right Arrow Keys
Pressing the Shift key together with the left and right arrow keys rotates the current transformable about its Z axis.

Note: If your window manager has appropriated the arrow keys for its own purposes, the functions just described may not work.

Transformation Options panel typeins do not have to be finished with the **Enter** key. Moving the mouse cursor out of the typein window will cause the transformation to occur. Repeatedly pressing the **Enter** key repeatedly applies the transformation.

Absolute and Relative

These two switches select between transforming the current transformable **Relative** to its current position or state, or to an **Absolute** position or state. The default is **Relative**.

Absolute sets the object's transformation matrix to be specifIed values. **Relative** appends the object's transformation matrix with the values specified.

When an object is selected:

- **Translate** happens with respect to the object's *parent* coordinate system, not within its own coordinate system. With individual objects this usually means, translate the object in the Top level object's coordinate system. For the Top level object, it means translate the Top level object within the world coordinate system.

- **Rotate**. Objects are rotated with respect to their center point, in their own coordinate system. Thus, if the Top level object's Z axis is sticking straight out of the screen, and the object's Z axis is pointing up, the object will rotate around the Z axis that is pointing up. Rotations also happen in a particular order, first X, then Y, then Z. The rotation transformation is not transitive.
- **Scale**. Objects are scaled in their own coordinate system with respect to their center point. Thus, if you scale an object to be twice its original size, then select the Top level object and scale it to be twice its original size, both believe they are twice their original size. The object does not think it is four times its original size, although that is how it appears.
- **Scale/Rotate Center**. This defines the center point for the object. Objects are rotated and scaled with respect to their center point. The center point of an object is within its own coordinate system, not that of its parent.

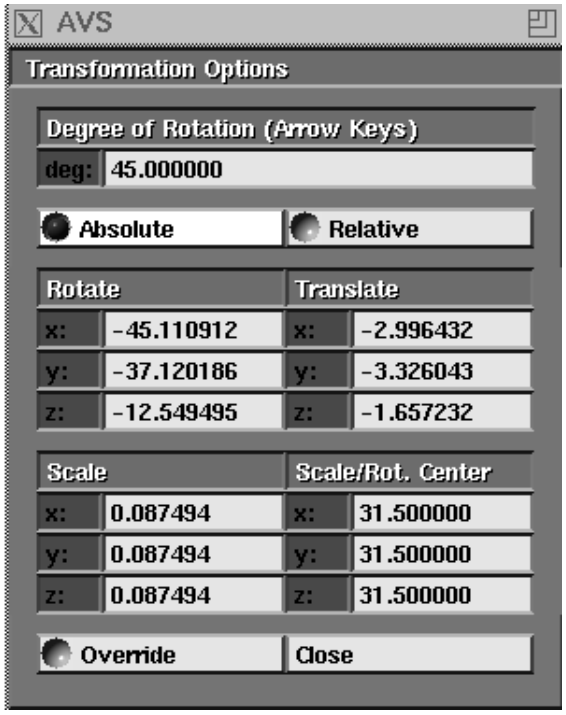


Figure 5-9 Absolute Transformation Values

Absolute

When you first toggle **Absolute**, it reports the current state of the current transformable: where the object is, how it is rotated, scaled, and where its center is (Figure 5-9).You can then type in new values for all these cells, *except* Rotate. You cannot do absolute rotations.

Note: The Absolute display of the Transformation Options panel is not updated automatically if you perform a transformation with the mouse or arrow keys, or if you change the current transformable. To see the

new absolute position, click on Relative then back to Absolute and the new information will be displayed.

On occasion you may see the string "Not Avail" instead of the absolute positions. It is possible for an object to get into a state where the equations that calculate its position, rotation, and scale are undefined (not available). This might occur, for example, if you rescaled an object in the X direction, but not in any other.

The meaning of the Absolute values for cameras are not intuitive. They are not "where the camera is in space and how big it is." Rather, Scale shows how much world coordinate space had to be scaled down to fit into the camera's view volume, and Translate shows how much it had to be shifted over. The Camera Options panel provides a more intuitive interface for camera transformations.

Relative

Relative repositions or rescales the current transformable as a delta value from its current location or scale. You can type in either positive or negative values.

Override

Override disables a module's control over the current object. Some objects in the Geometry Viewer such as the slice planes of the **arbitrary slicer** module, the **volume bounds** object and **isosurface** objects, and the probes of the **probe** and **hedgehog** modules, are being controlled by the module that produces them, not by the Geometry Viewer, through the invisible "upstream data" connection described in the "Advanced Network Editor" chapter. Switching on **Override** for the current object overrides the module's control of the object.

Bounding Box

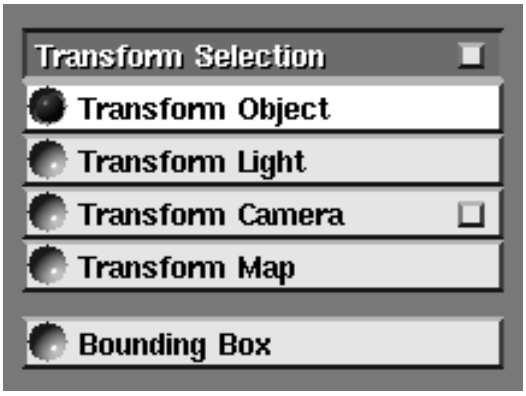


Figure 5-10 Bounding Box Button

Switching on **Bounding Box** makes transformations (rotate, translate, scale) performed with the mouse behave differently.

Normally, when you transform an object with the mouse, the system does its best to update the rendering of the object continuously, in "real time" as it tracks the mouse. So, as you rotate a teapot through an arc from A to E, the system tries to keep the picture "live." In practice, what you get are several intermediate views B, C, D as the teapot rotates from A to E. The more horsepower your system has, the smoother and more continuous the rendering. The more complex the scene is (objects with many surfaces, lights), the slower the rendering.

Bounding Box disables this resource-expensive effort at real time rendering. With **Bounding Box** turned on, when you place the mouse over the current transformable and press the middle or right button, a white wireframe box enclosing the volume of the object appears. (If the current transformable is a light, the lines representing the light change color.) As you move the mouse, the bounding wireframe box moves—*the object does not*. You move the bounding box to the destination position / rotation / scale, then let go of the mouse button. Only then is the object rendered at its new location / scale. It started at A, it ended at E, and positions A and E were all you saw— there were no B, C, D intermediate renderings of the object.

Bounding Box is particularly useful if you are running AVS on a comparatively low performance graphics system, when you are using the software renderer, or as a remote X client from an "X terminal" or workstation. It will make the interaction much faster, if less animated. It is also useful on faster systems trying to render very complex objects. The interaction may be further speeded by using **Bounding Box** in concert with **Freeze Camera**. Performance aside, some people like bounding box because the box gives them a more accurate image of the object's orientation in space.

Toggling **Bounding Box** affects all view windows on the screen.

There is a bounding box for the Top object and all objects in the scene.

You can have **Bounding Box** turned on by default for all your AVS sessions. In your *.avsrc* or *.avsrc.X* file, add the following line:

```
BoundingBox      1
```

Current Object Area

Current Object Indicator

This small window (Figure 5-11) shows a miniature graphical representation of the current object. Clicking any mouse button repeatedly in the Current Object Indicator window cycles through the list of objects for the current scene.

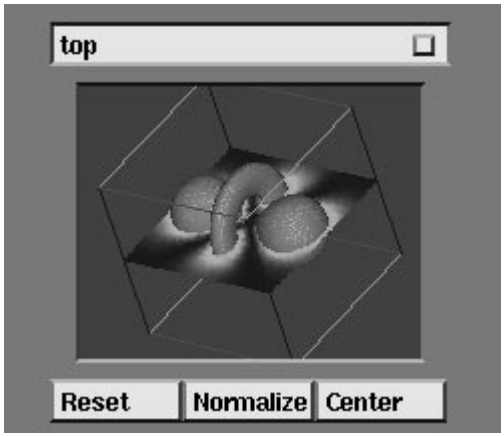


Figure 5-11 Current Object Selection Area

Current Object Browser

The Current Object Namebar shows the name of the current object. Clicking on this dimple produces the **Current Object Browser** (Figure 5-12). This is an alternate way to select the current object. It is intended primarily for the cases, such as the sample file *geometry/jet.geom*, where there are too many objects in a single level hierarchy for it to be reasonable to select them by clicking on them in the display window, or to cycle through them in the miniature current Object Indicator window.

It also supports current object selection in multi-level object hierarchies.

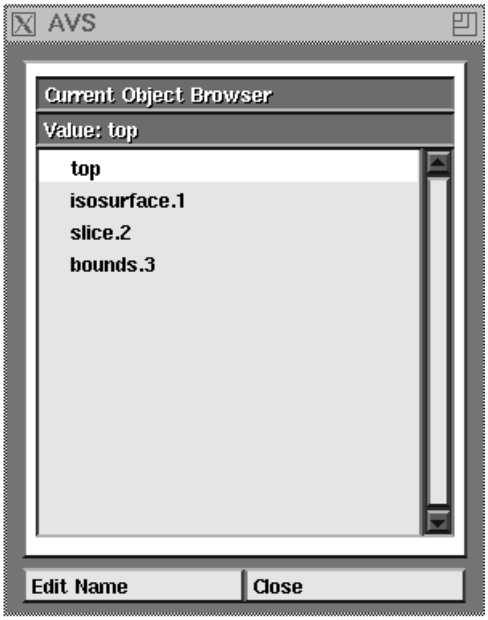


Figure 5-12 Current Object Browser

The names of the objects appear in a browser window very like the file browser widget. The current object is highlighted. Clicking any mouse button on any object name will make it the current object.

Renaming Objects with the Current Object Browser

The **Current Object Browser** has an **Edit Name** button at the bottom of its panel. Pressing this button brings up a typein panel that lets you name the current object. Note that object names *are* case-sensitive.

When you first read in an object, it comes with a name that appears in the Current Object namebar. This name was set by the program that created the *geom* object. By convention, modules name objects after themselves. The Geometry Viewer appends a sequence number to the object name, *name.n*. This is to distinguish between two objects from two modules with the same name, e.g. two **arbitrary slicer**s.

There is a good reason to rename objects. The issue arises when you are using AVS modules that produce geometries and you are just using the Geometry Viewer to examine the results. As the AVS module produces successive geometries, it repeatedly gives them the same name and the Geometry Viewer gives them the same sequence. Each new geometry replaces its predecessor. You will not be able to deal with these geometries as separate objects unless you rename them.

For example, you might be using the **arbitrary slicer** module in the Network Editor to cut several slices at different angles through a 3D volume of data, (perhaps a density image of a cranium) and display them composited together.

To create the composite slices, you would position the slice plane at the first desired angle, then "freeze" it by using the renaming option, creating a new object. Move the slice plane to the second desired angle, freeze it by renaming it, and so on until the whole composite picture has been constructed.

Additional Transformations

Just below the Current Object Indicator are three additional buttons:

Reset
: Restores the current transformable (object, light, or camera) to its original position when it first entered the Geometry Viewer. **Reset** maintains the color, surface properties, center point, and rendering mode of the object.

Normalize
: Scales the current object so that it fills its view window.

Center
: The **Center** button does *not* center an object within the display viewport. Rather, along with the **Scale/Rot. Center** typeins on the Transformation

Options panel, it allows you to control the **center of rotation** and scaling of individual objects and the entire Top level hierarchy of objects.

When an object or a collection of objects within a hierarchy (usually Top) is rotated (middle mouse button) or scaled (shift middle mouse button), it is always rotated or scaled about its **center**.

Center can be in one of four places:

- The *libgeom* library that modules use to create objects contains calls that allow a programmer to define an explicit **center** for the object. If the programmer defined an explicit center for the object, then that is where its center will be.

  The visual clue that an object or object hierarchy's center point is not its origin is that when you rotate the object with the middle mouse button, it does not rotate about what appears to be its physical center, but about some other point in space. This point may even be outside the physical bounds (extent) of the object.
- The center is at 0, 0, 0 in an object's own coordinate system unless the programmer defined an explicit center for the object as noted above.
- Whichever of the previous two cases was true, pressing the new **Center** button recalculates the center point for an object or a hierarchy of objects, overriding the previous setting.

  For individual objects, the center becomes:

```
center X coord = ((max X extent - min X extent)/2) + min X extent
center Y coord = ((max Y extent - min Y extent)/2) + min Y extent
center Z coord = ((max Z extent - min Z extent)/2) + min Z extent
```

  For collections of objects in hierarchies, the center point is calculated the same way, but using the minimum and maximum X, Y, and Z values of the smallest rectangular volume enclosing all of the objects within the hierarchy.
- Center can be set to any point in the object's coordinate system that is typed into the Transformation Options panel.

Reset by itself does not reset the center point of objects or hierarchies of objects. You can keep track of centers and reset them "manually" using the **Scale/Rot. Center** typein on the Transformation Options panel described above.

Function Key Usage

The choices in the Transform Selection areas can be made by pressing function keys instead of using the mouse. This can save you the "overhead" of moving the mouse cursor back and forth between the view window and the Transform Selection Area. The mouse cursor must be inside the view window.

F1

Selects **Transform Object**, "attaching" the mouse to the (composite) object shown in the Current Object Indicator window.

F2

Selects **Transform Light**, "attaching" the mouse to the current light, as indicated on the lighting panel under the **Lights** menu selection.

F3

Selects **Transform Camera**, "attaching" the mouse to the camera in the current window. If you move to a different window, the mouse automatically switches to the camera in that window.

F4

Selects **Transform Map**, "attaching" the mouse to the grid that shows how the current texture is aligned with its object.

F5

Toggles the state of the **Bounding Box** button.

F6

Cycles the current object, as shown in the Current Object Indicator window. This is the same as clicking the mouse in the Current Object Indicator window.

F7

Performs a **Reset**, returning the current object, light, or camera to its original position and orientation.

F8

Performs a **Normalize**, resizing and positioning the current object so that it fills the current window.

F9

Performs a **Center**, changing the center of rotation of the current object to be in the center of its extent.

Note: If your window manager has appropriated the arrow or function keys for its own purposes, the functions just described may not work. If this is the case, you can hold down the **Shift** key in combination with the function or arrow key to get the key to work for AVS.

Dial Box Usage

Figure 5-13 shows how the dialbox can be used as an alternative to the mouse for moving objects, lights, and cameras. To enable the dial box, use the **-dials** *devicespec* option on the AVS command line.

Spaceball Usage

To enable the spaceball, use the **-spaceball** *devicespec* option on the AVS command line. The spaceball should beep as AVS starts up.

rotate around X-axis	translate along X-axis
rotate around Y-axis	translate along Y-axis
rotate around Z-axis	translate along Z-axis
scale uniformly	*not used*

Figure 5-13 Dialbox Usage

The spaceball transforms an object, light, or camera as follows:

- Pulling up on the ball translates the object upward; pushing down translates the object downward.
- Pulling to the left or right on the ball translates the object accordingly.
- Twists in various directions produce rotations.
- Pulling the ball towards you makes the object larger; pushing the ball away scales the object down.

The spaceball's buttons have the following definitions:

Button 1
Allow / disallow translation portion of transformation. A short beep is sounded.

Button 2
Allow / disallow rotation portion of transformation. A short beep is sounded.

Button 3
Allow / disallow scale portion of transformation. A short beep is sounded.

Button 4
Unused.

Button 5
Decrease sensitivity of translate / zoom by factor of two. A beep is sounded.

Button 6
Increase sensitivity of translate / zoom by factor of two. Two short beeps are sounded.

Button 7
Reset sensitivity to initial settings. A short beep is sounded.

Button 8
Generate or stop drift.

Occasionally, you may lose the object of interest outside the view window. When this occurs, press the spaceball's button to reset the transformation matrix to its initial value.

Geometry Viewer Menu Reference

The Geometry Viewer control panel provides access to most functions for creating 3D "scenes", including a combination of objects, lights, and cameras. The Menu Selection area also provides functions for moving data between disk storage and Geometry Viewer windows.

The following top level menu choices are always visible in the Menu Selection area:

Objects
Lights
Cameras
Labels
Action

One of these choices is selected at any particular moment. For instance, when you start the Geometry Viewer, **Objects** is selected automatically. The area below this top level menu changes, depending on which choice is currently selected.

Objects

Selecting **Objects** causes the Menu Selection area to appear (Figure 5-14).

Read Object

This function allows you to retrieve one or more objects from disk files, placing them in the current window. As you select each object, it becomes the current object, as shown by the Current Object Indicator in the upper part of the control panel.

When you select **Read Object**, a small window (the File Browser) filled with filenames from the current directory appears near the control panel. See Figure 5-15.

The File Browser is "sticky" — it remains onscreen until you explicitly remove it by clicking on **Close**. This makes it convenient to retrieve multiple objects consecutively. You can also cancel **Read Object** by clicking on **Close** before you've read any objects at all.

The entries on the File Browser are color-coded: black entries are files that contain Geometry Viewer objects; red entries are subdirectories (the topmost

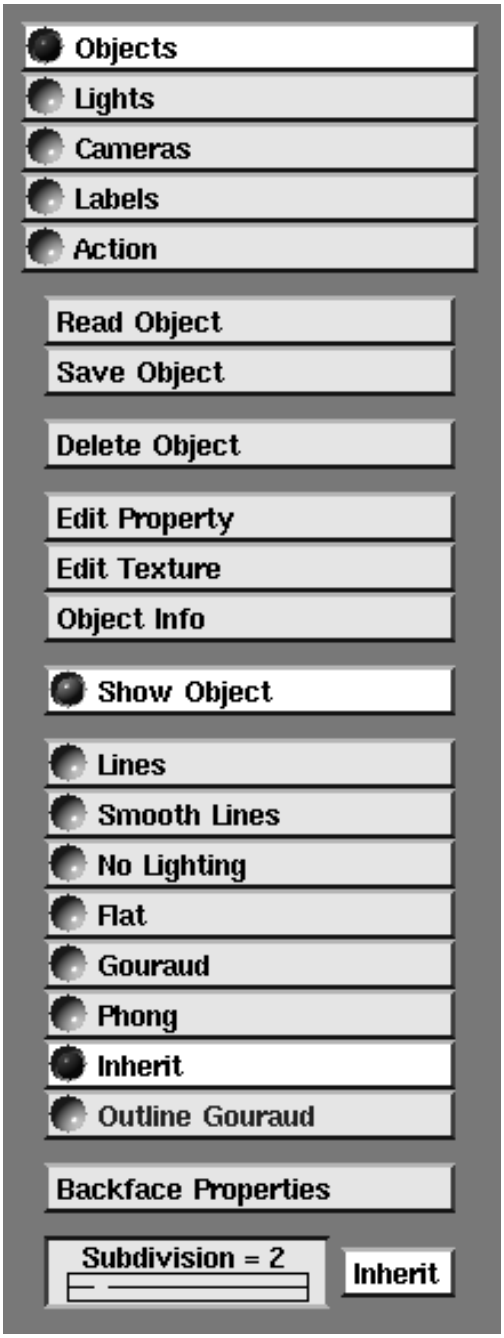


Figure 5-14 Objects Menu Selections

red entry is the parent directory). To select one of the entries, click on it with any mouse button.

Since a directory might contain a large number of entries, the File Browser has a scroll bar along its right edge. Clicking inside the scroll bar makes additional entries appear:

Figure 5-15 The File Browser

- The left mouse button scrolls upward.
- The effect of the middle button depends on exactly where the cursor is:

  In the arrow box at the top. Click to scroll the list to the very top.

  In the elevator shaft. Click and hold down the button to grab the elevator bar. Moving the bar up or down causes the list to scroll accordingly.

  In the arrow box at the bottom. Click to scroll the list to the very bottom.
- The right mouse button scrolls downward.

Selecting an object adds it to the current window. You can then use the mouse to move, rotate, and resize the object.

Selecting one of the red (directory) entries changes the working directory. The names of the Geometry Viewer object files in that directory are displayed, along with the names of any subdirectories.

You can also change the working directory by clicking on **New Dir** at the bottom of the File Browser. A window pops up so that you can type the name of another directory (Figure 5-16). (If you change your mind, click **Cancel** with the mouse.) Be sure the mouse cursor is in the one-line text-entry area before you start typing the directory name.

Similarly, you can click the **New File** button to enter the full or partial pathname of a file. Be sure to include the filename extension.

When typing a filename or directory name, you can use the **Backspace** key to erase the last character. Pressing **Ctrl-U** erases the entire line you've typed.

Figure 5-16 Entering a Fileneme

You can type a full pathname (starting with /) or a pathname relative to the current directory. the name of the current directory is displayed above the text-entry area. For instance, to go two levels up the directory hierarchy, you would enter **../..** as the new directory.

To finish entering the new directory name, press the **Enter** key or click **OK** with the mouse.

Save Object

This function saves the current object (shown by the Current Object Indicator) in a *.scr* file using the Geometry Viewer CLI commands. This can be a *composite object*, consisting of two or more of the simple objects defined in *.geom* files. Any properties you have assigned with the Edit Property window are also saved, as are the rendering method(s) for the simple object(s).

If the object's geometry has been modified either by a module or by the geometry viewer user interface since the object was last saved, the **Save Object** function will also create a *.geom* file that defines the geometry of the object. In this case, you will need to enter two filenames, one for the *.geom* file and one for the *.scr* file. Be sure to read the prompt on the dialog box as it will indicate which file you are saving.

Note: You must use **Save Scene** to save exactly what you see in a view window. Using **Save Object** saves just the object, its properties, and the position and orientation of the object in space. It does not save light or camera positions.)

Note: The *.scr* file contains references to one or more *.geom* files. That is, the *.scr* file does not contain copies of geometries, but merely contains pointers to them. For this reason, be careful not to disturb *.geom* files that store the "building blocks" for your objects.

You don't need to type the *.scr* extension when you enter a filename—AVS adds this extension automatically (unless you type it yourself). To finish entering the filename, press the **Enter** key or click **OK** with the mouse.

Click on **Cancel** to cancel the save operation.

After you've saved an object, its filename appears in the File Browser. You can later bring the object back into the same window, or a different one, using **Read Object**.

Delete Object

This function removes the current object (shown by the Current Object Indicator) from the current window. It also removes the object from all other windows that show the same scene.

Since there is no way to "undo" deleting an object, you may want to perform a **Save Object** before deleting something that might be useful later on.

Edit Property

This function allows you to change the reflectance properties of the current object. The way in which an object in the real world reflects light depends on the characteristics of its surface: color, material (e.g. plastic, metal, fabric), smoothness, etc. From an intuitive point of view, then, this function allows you to specify the material from which the object is constructed.

Note: Some properties, such as transparency, require special graphics hardware/software support that may not be present on your platform's hardware renderer. In this case, switch to the software renderer under the **Cameras** menu.

When you select **Edit Property**, a window appears next to the control panel. The Edit Property window contains sliders that control the various surface properties (Figure 5-17).

When the window first appears, the sliders show the current settings for the current object. As you move the sliders, the image of the object changes as soon as you release the mouse button.

Note: You can move a slider with any mouse button. You can either drag a slider by holding down the mouse button, or just click once at the spot where you'd like the slider to move.

Like the File Browser, the Edit Property window is "sticky" — it remains onscreen until you explicitly remove it by clicking on **Close**. This makes it convenient to change several properties of an object, or to change the properties of several different objects.

The sliders in the Edit Property window are as follows:

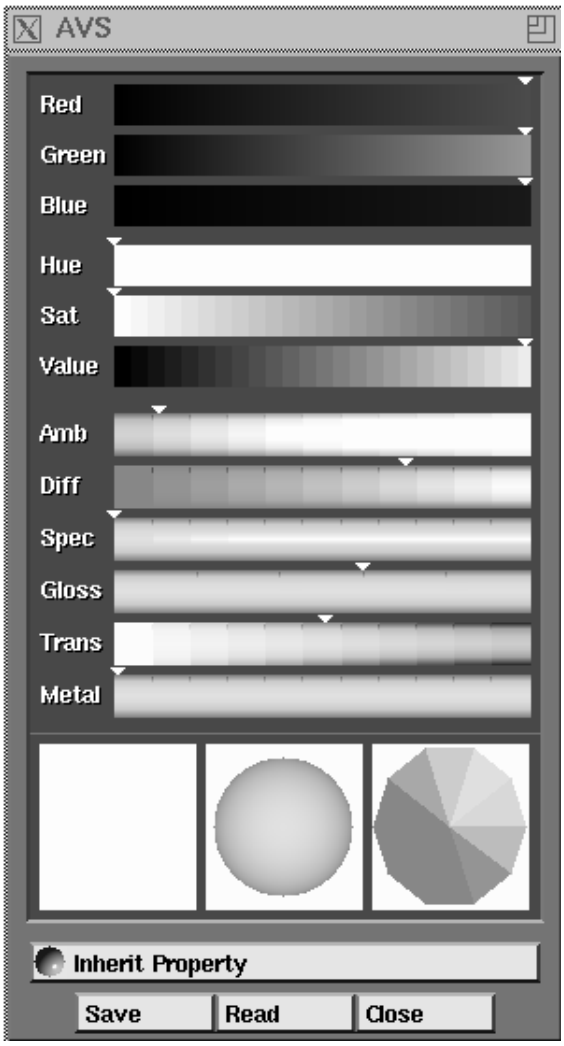


Figure 5-17 The Edit Property Window

RGB Color

The top three sliders control the object's color by adjusting the amount of red, green, and blue. To make an object white, move all three sliders all the way to the right. To make an object black, move all three sliders all the way to the left.

HSV Color

The next three sliders provide an alternative way to specify the object's color: hue-saturation-value.

The hue, saturation and brightness (HSB) color space can be thought of as an inverted cone:

- The **hue** axis runs circularly around the cone. Example hue values and corresponding hues are given below:

  0.00 = red
  0.16 = yellow

0.33 = green
0.50 = cyan
0.66 = blue
0.83 = magenta

- The **saturation** axis runs from the center of the cone (white) to its perimeter (fully saturated color). Example saturation values are:

  0.00 = white
  0.50 = partially saturated **hue**
  1.00 = fully saturated **hue**

- The **brightness** axis runs from the tip of the cone (black) to the base (white). Example brightness values are:

  0.00 = black
  0.50 = partially darkened **hue**
  1.00 = full intensity **hue**

  Note that the RGB slider set and the HSV slider set provide two ways of controlling the same property — the object's surface color. Whenever you make an RGB change, the HSV sliders automatically adjust to reflect the change, and vice-versa.

Figure 5-18 Teapot with Metallic Surface and Specular Highlights

Ambient Light Reflectance

The proportion of the available ambient light that the object reflects. Ambient light is non-directional, affecting all parts of all surfaces equally.

This setting determines how much ambient light the object reflects. To control what ambient light there *is* in the scene, select the **AM** light on the lighting panel. This is described under the top level menu choice **Lights**.

Diffuse Light Reflectance

The proportion of the available *non*-ambient light that the object reflects equally in all directions. Non-ambient light emanates from directional and point light sources, which you specify with the lighting panel.

This setting is used in the calculations for Flat, Gouraud, and Phong shading.

Specular Highlight Intensity/Gloss/Metal

Specular highlights of a particular color and brightness are created when the direction of incoming light (from a directional or point light source) is "sufficiently close" to the viewing direction.

The **Intensity** determines the brightness of such highlights. It corresponds to the specular coefficient in the lighting calculations.

The **Gloss** setting determines what "sufficiently close" means. The greater the sharpness, the smaller (more focused) the size of the specular highlight. This setting corresponds to the specular exponent in the lighting calculations.

The **Metal** setting specifies the color of the specular highlight. AVS constrains the color to be somewhere between the color of the light source (leftmost) and the color of the object (rightmost).

Transparency

This setting controls the degree to which you can see through the front of an object, allowing you to see the back of the object and other objects behind it.

The Edit Property window also contains these buttons:

- The **Save** and **Read** buttons enable you to maintain a library of properties settings on disk. Each time you **Save**, AVS creates a file containing the current settings of all the sliders. It prompts you to enter a filename, and automatically adds the filename extension *.prop* to the name you enter. Be sure the mouse cursor is in the one-line text-entry area before you start typing the filename. There are a wide variety of sample properties in the default directory.
- The **Inherit** button replaces the current slider settings with those of the parent of the current object. The current settings are not lost, however. To Srestore them, just click on **Inherit** again. For longer-term storage of properties settings, use the **Save** and **Read** buttons.

Edit Texture

Note: Not all platforms' hardware renderers support 2D and/or 3D texture mapping. The supposedly texture-mapped objects will appear a featureless white. To get texture mapping, switch to the software renderer under the **Cameras** submenu.

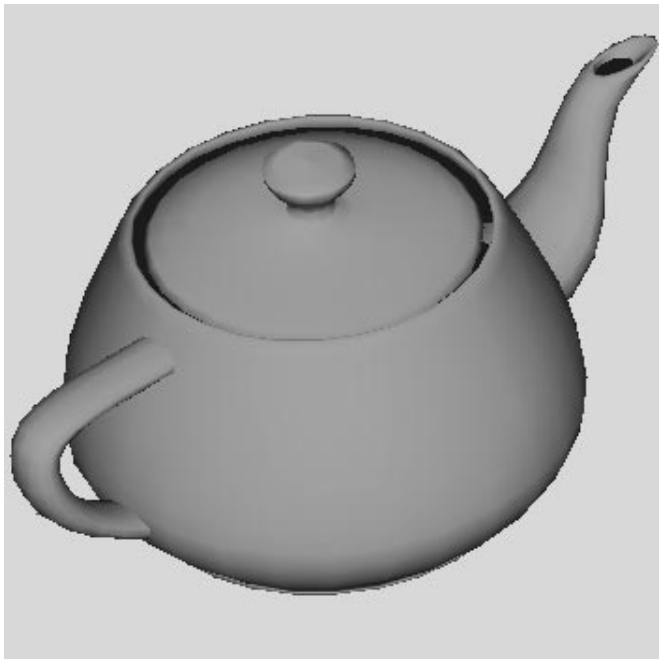


Figure 5-19 Teapot with Clay Surface

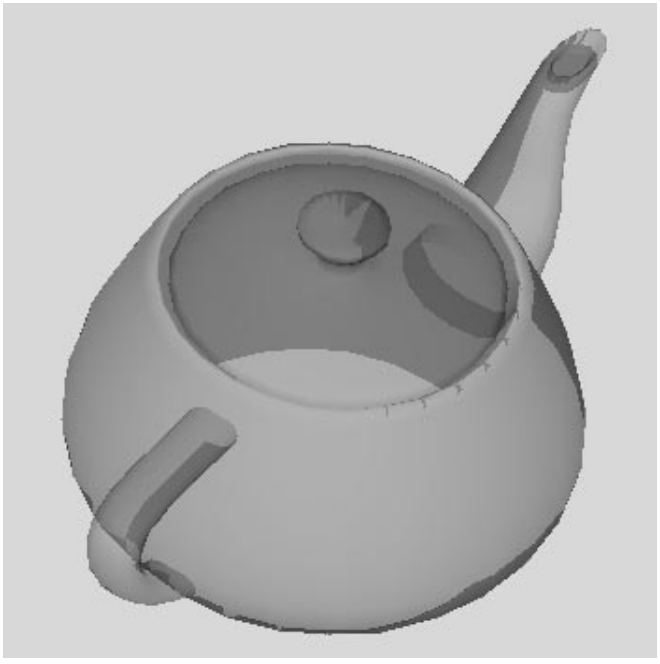


Figure 5-20 Teapot with Semi-Transparent Surface

2D texture mapping is the mapping of a two-dimensional image to the surfaces of three-dimensional geometry. The process begins with a pixmap containing the two-dimensional image (the texture) and a geometry object to which

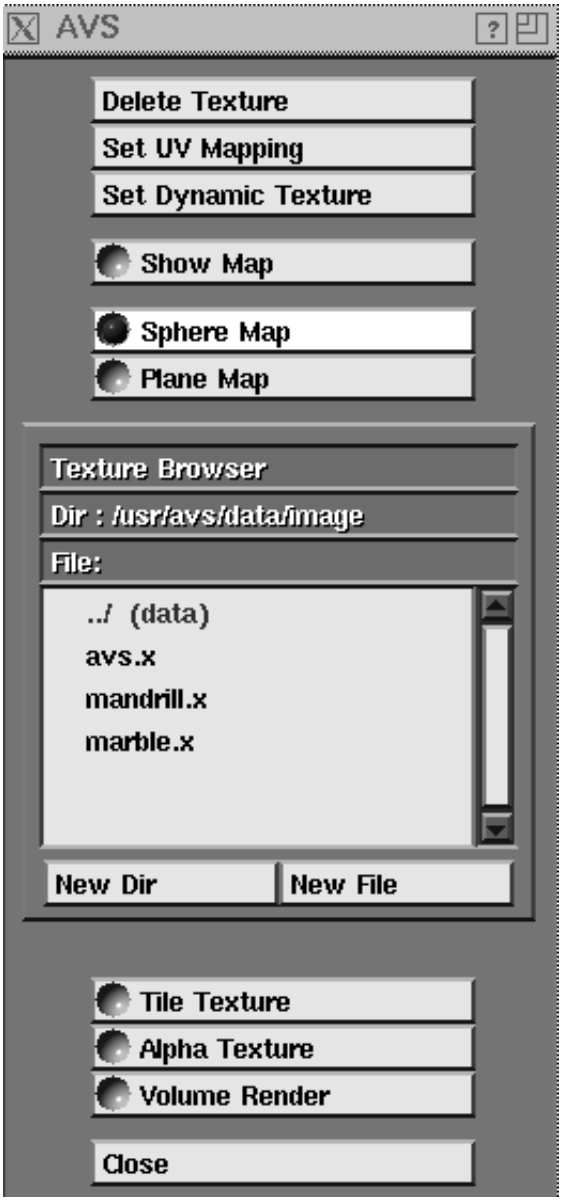


Figure 5-21 Edit Texture Panel

the image is to be mapped. Each vertex of the object is associated with a point in the texture. When the object is rendered, that point in the texture appears on the surface of the object at the associated vertex. AVS uses the underlying graphics subsystem to map the remainder of the texture to the object surface by means of linear interpolation between vertices.

Note: The texture to geometry mapping can also be defined with *libgeom* calls that provide explicit UV(W) coordinates. Otherwise, the Geometry Viewer's interactive technique described below can be used.

Steps in Using AVS Texture Mapping

Here's a procedure for doing 2D texture mapping:

1. Choose an object to which the texture is to be applied. Make that object the current object by clicking on it with the left mouse button.
2. In the **Object** menu, click on **Edit Texture** to bring up a window of choices.
3. Decide which texture mapping method you want to use: *sphere* mapping or *plane* mapping. Click the choice you want if it is not already selected.

   In sphere mapping, the object's vertices are projected onto a sphere, and the texture is effectively wrapped onto the same sphere. The width of the texture is spread along the equator; the top and bottom of the texture are compressed at the two poles.

   In plane mapping, the object's vertices are projected onto a plane that effectively contains the texture.
4. Click on **Set UV Mapping** to have AVS establish the mapping between the object's vertices and the texture. Whenever you change the mapping type or you transform the texture map (e.g. rotate it), you must click this button again to update the mapping. AVS does this automatically if you try to apply a texture to an object that does not have UV mapping information.
5. Specify a filename using the File Browser. You can use only image files in the AVS *image* format, which have the filename extension *.x*.

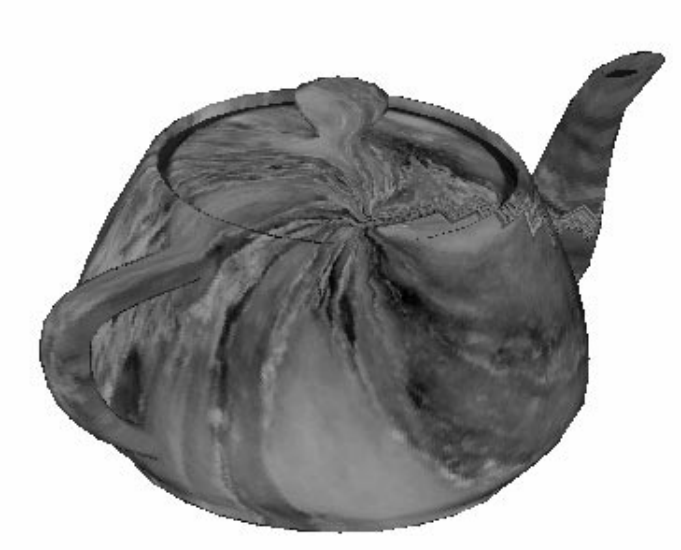


Figure 5-22 Teapot Texture-Mapped with marble.x

This procedure causes the texture to appear on the object. Whenever you transform the object (translate, rotate, scale), the texture transforms accord-

ingly. You can also transform the texture mapping itself (e.g. change the orientation of the plane onto which the object's vertices are projected). To do so, select **Transform Map** in the Geometry Viewer control panel at the left edge of the screen, and use the mouse to transform the map. After you transform the map, click **Set UV Mapping** again to update the mapping of the object's vertices to the texture.

For more on object and texture transformations, see "Transformations and the Transform Selection Area" above.

The Dynamic Texture

The **Set Dynamic Texture** button is for use when the Geometry Viewer is included in an AVS network, as the **geometry viewer** module. This module takes an image as an optional input through its multi-colored field input port. Clicking **Set Dynamic Texture** causes this image to be selected as the texture to be mapped to the current object. This same input port is used to perform 3D texture mapping (e.g., as used by the **brick** module).

There are several different options for performing texture mapping. Not all will be available or meaningful on each platform.

Filter Texture
: Anti-aliases the texture by resampling to remove artifacts that occur when the "pixels" of the texture map have become obvious due to expansion or shrinkage.

Tile Texture
: When uv(w) values exceed the 0-1 range, different renderers produce different visual artifacts. On some renderers, values outside the range are clamped to 0 or 1, producing a smear at the edge of the image that continues to infinity. With the software renderer, values outside 0 to 1 are not texture mapped at all. With **Tile Texture** turned on, values outside 0 to 1 are wrapped, thus continuing the texture map without obvious visual artifacts.

Alpha Texture
: Use the alpha value in the texture pixel to control transparency. The software renderer supports this option. The **volume render** module makes use of this option.

Volume Render
: The **volume render** module uses this texture mapping option. It causes the **geometry viewer** module to treat the 3D texture map entering its leftmost input port as data for volume rendering.

Object Info

Clicking this button displays a window of information pertaining to the current object:

- The object's Geometry Viewer name

- Number of child objects
- Number of triangles in the object
- Number of lines in the object
- Number of triangle strips in the object
- Number of polylines in the object
- Number of disjoint lines in the object
- Number of spheres in the object
- Additional object data: vertex normals, vertex colors

Show Object/Hide Object

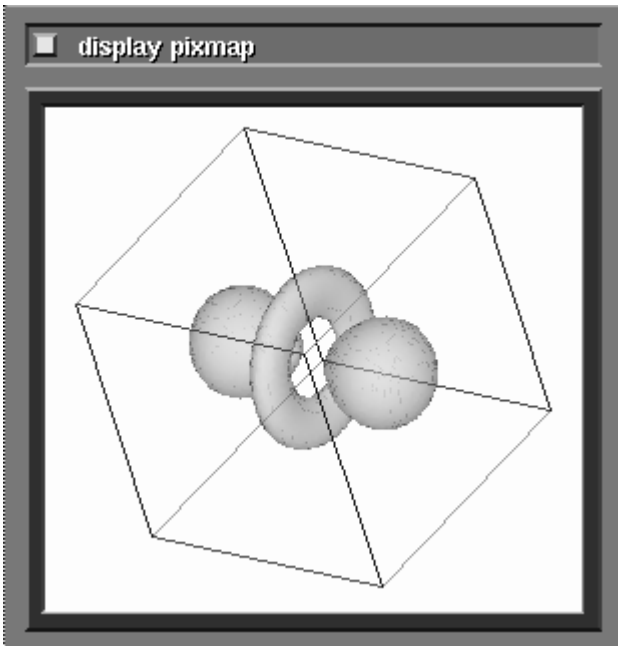


Figure 5-23 Slice Plane Object Hidden

The functions cause the current object to disappear from the current window (Hide) or reappear there (Show). The object also disappears or reappears in all other windows that show the same scene.

A hidden object is still part of its scene. If you perform a **Save Scene** (described under **Cameras**), the hidden object is saved along with all the visible ones. You can later perform a **Read Scene** followed by a **Show Object** to bring the object back onscreen.

The following menu items change the rendering method used to draw the current object.

Points

The object is drawn as a set of points in space, one for each vertex. This feature is renderer-dependent.

Lines

The object is drawn as a wire-frame, using non-anti-aliased lines. (Not all objects have wire-frame representations.)

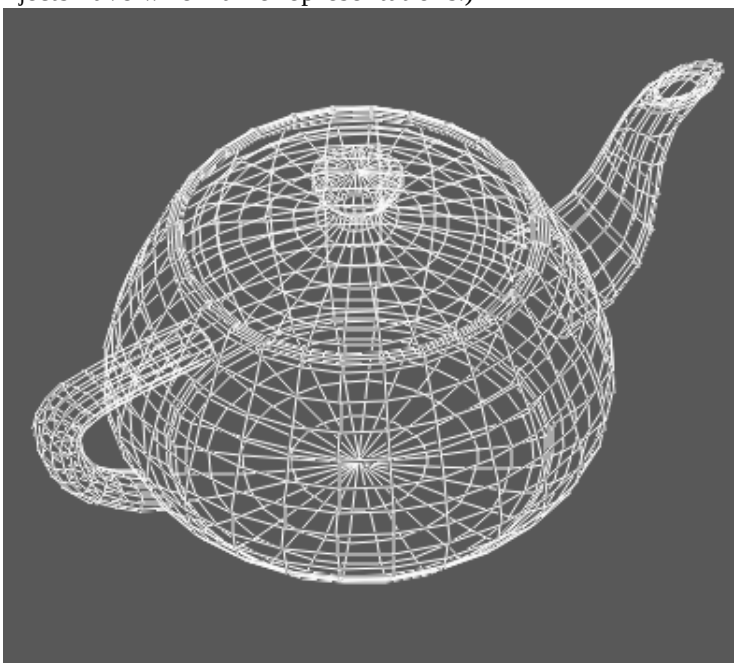


Figure 5-24 Teapot Rendered as Smooth Lines

Figure 5-25 Teapot Rendered with Flat Shading

Smooth Lines

The object is drawn as a wire-frame, using anti-aliased lines. (This feature is renderer-dependent.) See Figure 5-24.

No Lighting

The object is drawn using filled polygons, using no lighting or shading at all. The only color (or colors) used is the color of the object itself.

Flat Shading

The object is flat shaded. See Figure 5-25.

Gouraud Shading

The object is drawn using Gouraud shading. Gouraud shading blends the colors of a polygon's vertices across the polygon face, simulating smooth shading. (This feature is renderer-dependent. The software renderer supports Gouraud shading.)

Outline Gouraud

Outline Gouraud shows an object rendered with Gouraud shading, with the lines of its polygon representation superimposed over it. This allows you to see both the object's actual polygon structure and its shaded representation at the same time. (This feature is renderer-dependent. The software renderer supports **Outline Gouraud.**)

Phong Shading

The object is drawn using Phong shading. This produces a more realistic rendering than either Flat or Gouraud shading. (Phong shading is renderer-dependent.)

Inherit

Causes the current object to inherit the rendering mode of its parent object. Clicking again restores the previous explicit setting of the rendering mode.

Backface Properties

The **Backface Properties** button controls the way in which polygons that are facing away from the viewer are drawn. The direction in which a polygon is facing is determined by the order of its vertices or the direction of its vertex normals. If an object is completely enclosed, like the example file *dodec.geom*, you can "cull" away the back-facing polygons without seeing any changes in the image.

Press and hold down the mouse on this button and you will see a pop-up menu that contains several choices for the backface properties. The current backface property will be highlighted. Release the mouse button over one of the choices to select it for the current object.

Different renderers will support different culling options.

Normal
This mode causes the object's backfaces to be drawn in the normal backface mode for the specific renderer. Some renderers "flip the normals" when a polygon is backfacing so that the backside of the polygon is lit in the same way as the front face. Other renderers light the backface of the polygon with the ambient intensity as the normal rendering mode.

Cull Back
This backface mode causes the render to not draw polygons that are backfacing.

Cull Front
Some renderers support this rendering mode in which front faces are not drawn. This mode is of limited utility and is only useful when the renderer does not accurately determine front versus backfacing polygons.

Flip Normals
Some renderers support both the mode where backfacing polygons are lit and the mode where backfacing polygons are colored with only the ambient intensity. If this is the case, the normal mode will be where backfacing polygons are lit with ambient intensity. In this case, the GEOM_BACKFACE_FLIP mode can be used to cause the normals to be flipped and the backfacing polygons to be lit like front faces. Using bi-directional light sources is a partial workaround for systems that do not support the GEOM_BACKFACE_FLIP rendering attribute.

Inherit
This mode causes the specified object to inherit the backface property of its parent object. This is the default backface property for a newly created object.

Subdivision

Spheres can be rendered in two ways:

- If your hardware has support for sphere rendering, then spheres will be rendered as true spheres. All the hardware requires is a centerpoint and a radius to make a spherical object appear in the output window. The software renderer emmulates true sphere rendering.
- Without hardware sphere rendering, spheres are rendered by approximating them with some number of polygons.

Even when there is support for true sphere rendering, it can be disabled by toggling **Polygonal Spheres** under the **Cameras** submenu.

The **Subdivision** slider bar controls how many polygons are used to render a sphere. It takes many polygons to produce a picture that looks like a smooth sphere. On systems without software/hardware support for sphere rendering, spheres can be slow to render. The problem is aggravated if you are using a **bubbleviz/scatter dots** network or a ball-and-stick representation of a complex molecule that produces *many* spheres. A low **Subdivision** value such as 1 produces an 8-polygon diamond instead of a sphere, which renders more quickly.

The **Subdivision** control has no effect when true sphere rendering is present and **Polygonal Spheres** is not selected under the **Cameras** submenu.

Inherit

Inherit controls whether an object will inherit the **Subdivision** property of its parent object. It is ON by default.

Lights

Selecting **Lights** causes the Menu Selection area to appear (Figure 5-26).

The grid of numbers in the figure is a "lighting panel". The number of lights available is renderer-specific, but is usually either 16 or 8. The figure shows 16 lights, and that will be the assumption used in the rest of this section.

In any scene, you can define up to 15 directional or point lights. In addition, you can specify the **ambient** light, indicated by **AM** on the lighting panel.

The original window of every scene is created with the following initial lights:

- World space is flooded with a uniform, non-directional **ambient** white light. The ambient light source ensures that an object will always be visible even if other lights are not illuminating its surfaces. You can turn ambient light off for very dramatic lighting effects.

  The ambient light has no location. When deciding how to portray an object as lit, the value of the ambient light source is always added to the other factors determining the object's lit appearance. Because the ambient light is non-directional and from no particular source, it does not change the appearance of an object based upon the object's normals. What this means is that, if the ambient light source is the only light turned on, the object will appear as a featureless gray profile of itself.
- Directional light #1, with white color. The direction of the light is parallel to your line of sight, as if a white sun were directly behind you.

The rest of the lights are OFF.

Lights, by default, are white in color, with a brightness factor of 1. To darken a light, change its color with the RGB or HSV slider controls at the bottom of the Lights submenu. The distance of a light from an object does not affect its

Figure 5-26 Lights Menu Selection

intensity; there is no inverse square law at work. To make a scene brighter, turn on more lights.

To create an additional light, click the number on the lighting panel. Then, click **Light On** to turn on the light. If you wish, click **Point** to make the light a point light source instead of a directional light source.

At any particular moment, one light in the scene is the "current light". The number of the current light is always highlighted on the lighting panel. All the lights that are currently on are indicated by green numbers on the lighting panel.

Light On/Light Off

Turns the current light on or off.

Directional/Point/Bi-Directional/Spot

Selects the type of the current light. In the following descriptions provide both intuitive and more formal descriptions of the light types.

- **Directional**: (default light type) A light source whose rays all point in the same direction (are parallel). The sun is the canonical directional light source. Initially, a directional light is computationally located at infinity along the positive world coordinate Z axis. The initial directional vector for the light is said to be 0,0,-1, meaning that the light rays are pointing towards the negative Z direction, perpendicular to the XY planes. When you make a directional light source visible, its object representation appears to be at 0,0,1.
- **Bi-Directional**: A pair of directional light sources that point in exactly opposite directions. See Figure 5-27. This type of light can be used to "correct" the lighting of an object whose faces have been carelessly defined, so that the normals of some faces point outward and the normals of other faces point inward.

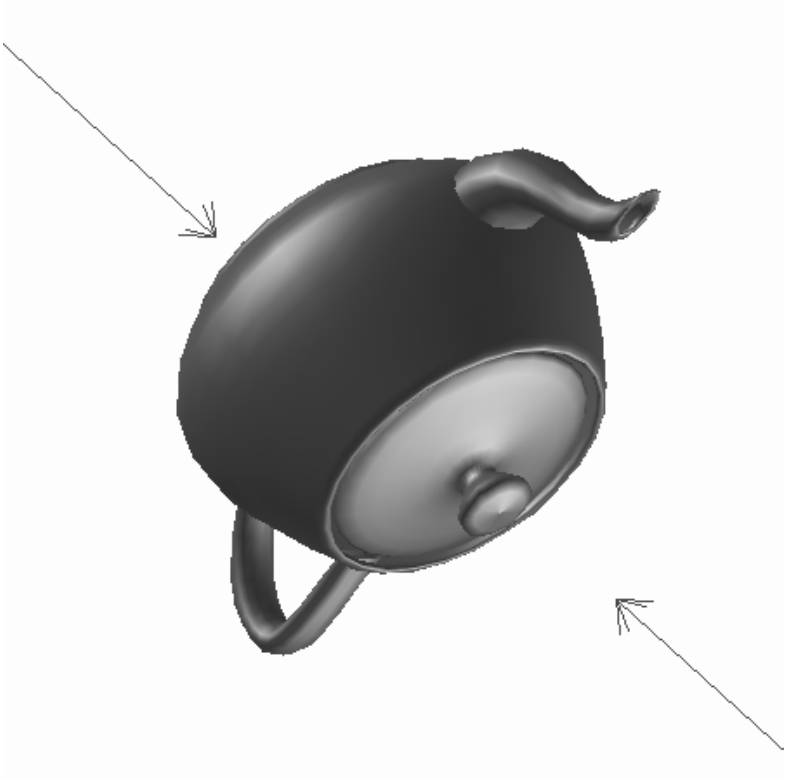


Figure 5-27 Bi-Directional Lights

A bi-directional light is actually implemented as two lights — it occupies two positions, *n* and *n*+8, in the lighting panel. For example, if you make light #5 bi-directional, then light #13 is used as the second light. Accordingly, only lights #1 - #7 can be specified as bi-directional. The two light sources are located at infinity along the positive and negative world coordinate Z axis. When you make a bi-directional light visible, its two repre-

sentations appear at 0,0,1 and 0,0,-1 in world space. Similarly, its lighting vector is 0,0,-1 and 0,0,+1.

- **Point**: A light source whose rays emanate in all directions from a particular point. A bare light bulb is the canonical point light source. Point lights sources have location in world space (initially at 0,0,1), but no direction. A light source can be inside an object.
- **Spot:** A light source whose rays emanate from a particular point and are restricted to a 90 degree cone. A flashlight is the canonical spot light source. Spot light sources have location (initially 0,0,0), and direction (a vector of 0,0,-1 meaning that it is initially pointing toward the -Z axis). (Light sources can be inside of objects.) The cone angle (90 degrees) is not adjustable. The intensity of the light thrown by the cone will fall off from the center to the edge. The degree of fall-off is dependent upon the underlying graphics subsystem lighting model.

Show Lights

Displays a vector symbol for each light source, indicating its (position and) direction. (Figure 5-28.) The size of the symbol indicates the light source's orientation vis-a-vis the view plane (the plane of the display screen). The symbol's color is the same as the light source color, and it is depth-cued (if this function is supported) to help indicate its distance from the view plane.

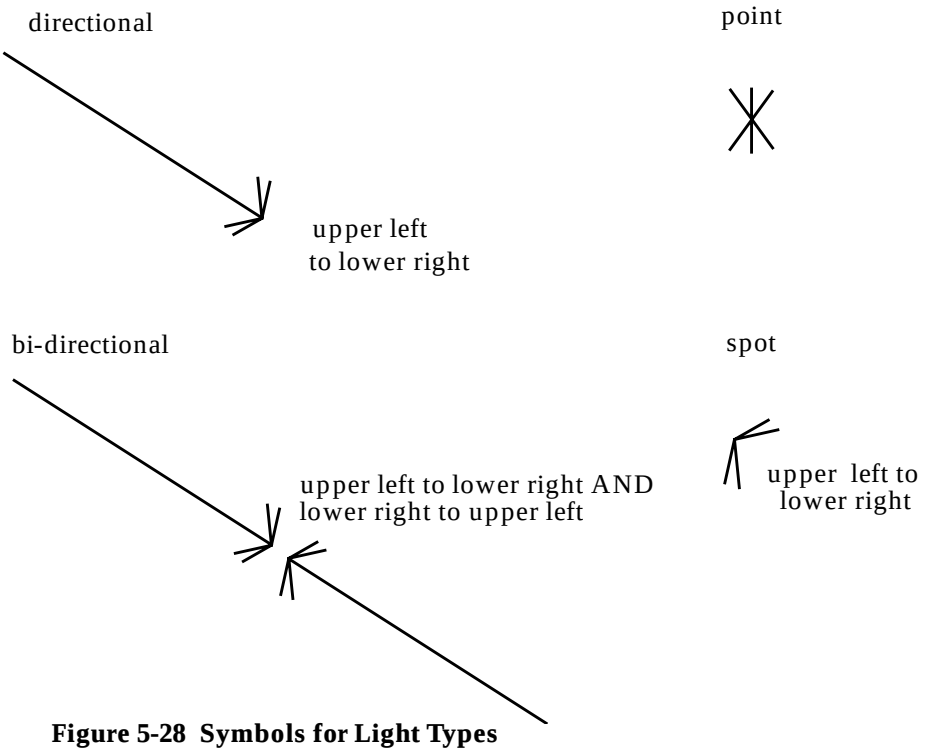


Figure 5-28 Symbols for Light Types

To move (the direction of) a light, make sure that **Lights** is selected in the Transform Selection part of the control panel. Then use the mouse to rotate and/or translate the light. See the **Transform Selection** section for details. It often helps to turn on **Perspective** under the **Cameras** submenu as you try to position lights with the mouse button transformations.

Color of Light

At the bottom of the **Lights** menu selection area, there are RGB and HSV sliders for setting the color of the light source. See "Edit Property" above for an explanation of how to use the sliders.

The default color of all lights is white (brightness value of 1). Changing a light's color also darkens the light.

Cameras

Selecting **Cameras** causes the Menu Selection area to appear (Figure 5-29).

Cameras Defined

You look upon a world space through one or more **cameras**. The camera creates a **view** or **view window** on the screen. You can create multiple views of the same world space by clicking on **Create Camera** under the this submenu. Cameras are initially "located" at about (0,0,100) in world space, looking toward the origin along the Z axis.

The camera sees a **view volume**. *Initially*, the camera's view volume sees a rectangular subset of world space extending from +5 to -5 X, +5 to -5 Y, and +100 to -100 Z in world coordinates.

The view volume travels with the camera as it is moved about world space. As such, it has its own coordinate system, sometimes called **camera coordinates**. The camera is always looking from the positive Z axis of *camera coordinates* toward the negative Z.

Although one can translate the camera, scale the camera (like zooming in and out), and rotate the camera, in some sense this is a misnomer. What is really happening is that world space is being rotated, scaled, and translated in order to fit within the camera's view volume.

You can change the shape of the camera's view volume. By default, the view volume is an oblong box extending from the camera's location to infinity along the Z axis.

- You truncate this box along the camera's Z axis by creating two **clipping** planes. Turn on **Front/Back Clipping** under the **Cameras** submenu. Objects in front of or behind the two clipping planes will not be rendered in the view window. The location of the front and back clipping planes can be adjusted with the Camera Options panel.

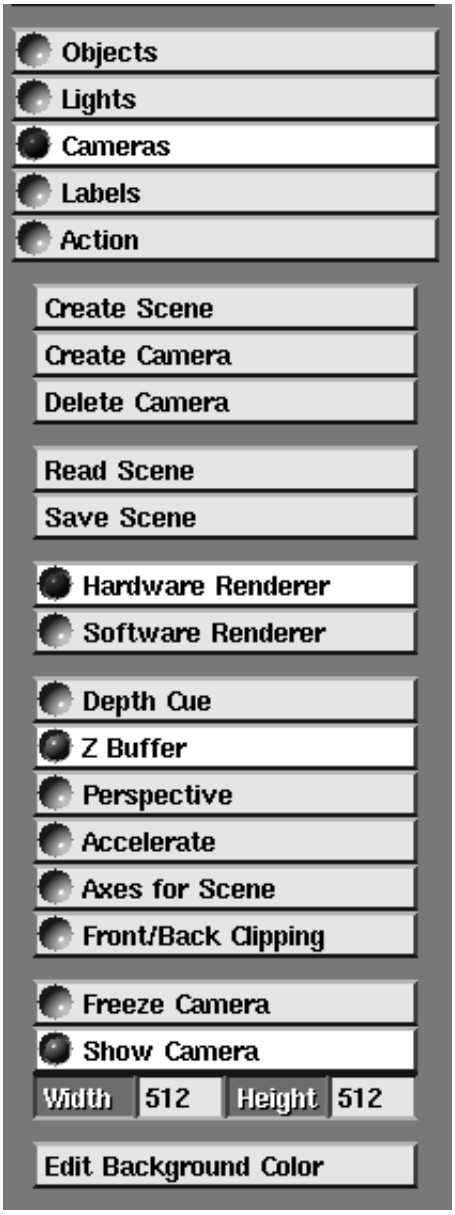


Figure 5-29 Cameras Menu Selections

- You can change the box from a rectangle to a frustrum (a squared-off cone) by turning on **Perspective**. With **Perspective** on, the portions of objects in front of the view volume's Z=0 plane are uniformly exaggerated in size as they approach the camera, and uniformly diminished behind the Z=0 plane. The angle of the side of the frustrum is 45 degrees. **Perspective** is adjustable with the Camera Options panel.

  Perspective projection gives a more real world rendering of objects. You may need to turn on **Perspective** for your eye to be able to interpret a scene.

Cameras also control different aspects of how the object is rendered, such as depth-cueing and Z-buffering (removing hidden surfaces). All of these properties can be controlled on a per-camera basis.

There is one last point: The camera view volume's Z=0 plane can intersect world space from any angle. This intersection is called **screen space** or **screen coordinates**. When you translate objects and lights using "direct manipulation" movements with the workstation's mouse buttons, the transformables *move with respect to the camera's view volume*, not world, Top, or object-level coordinates. By and large, this is a distinction that you can ignore. You will usually only notice it when you use direct manipulation on objects such as the **arbitrary slicer** module's slice planes or the **probe** module's pointer. When you move the slice plane, for example, in the Z direction, the slice plane moves straight toward or away from the camera (camera's Z axis), not along the object's Z axis.

The **Cameras** submenu selections allow you to create additional windows to display the current collection of objects (that is, different *cameras* for the current *scene*). You can also create entirely new scenes, with different sets of objects. See Figure 5-30.

Create Scene

Creates a new, empty window. The new window becomes the "current window", as indicated by the bright red border.

Create Camera

Creates a new window that contains the same object(s) as the current window. This is not a new scene, but an additional window on the same scene. Each such window can have its own camera position, and its own settings for the camera parameters: depth cue, Z buffer, and Accelerate.

When you make a change in one window, all the windows on the same scene are affected simultaneously. This includes rotating or moving an object, changing an objects surface properties, changing the color or position of a light, and so on.

See **Freeze Camera** for a way to suppress this synchronization of windows on the same scene.

Note: In general, the new window is a different size from the original, so the images of the objects are scaled appropriately. The new window becomes the "current window", as indicated by the bright red border.

Delete Camera

Deletes the current window. If you delete the last camera of a particular scene, then the scene itself is deleted, too.

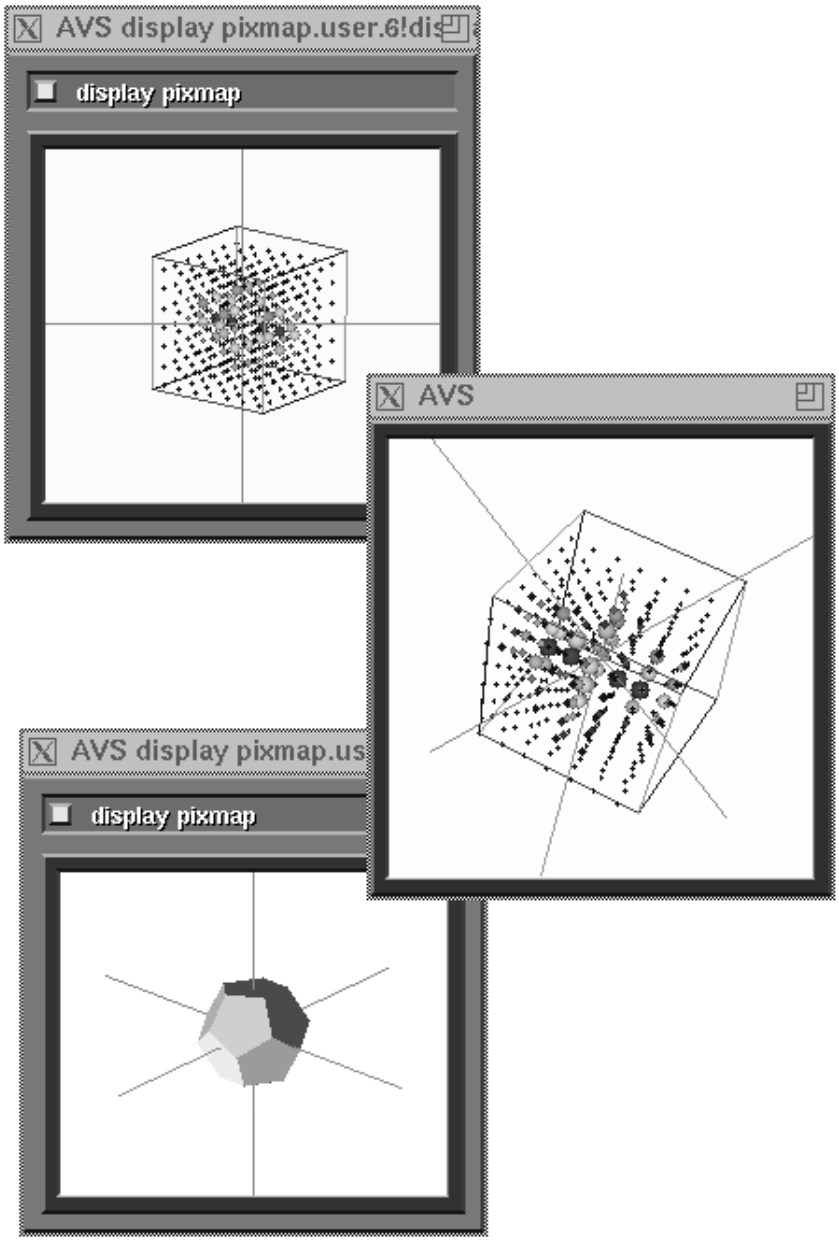


Figure 5-30 Two Scenes and Three Cameras

Read Scene/Save Scene

These functions allow you to maintain a disk library of scenes. Each scene consists of one or more windows. Selecting **Save Scene** stores the current state of all of the scene's windows in a file. AVS prompts you to enter a filename, and automatically adds the filename extension *.scr* to the name you enter. Be sure the mouse cursor is in the one-line text-entry area before you start typing the filename.

Selecting **Read Scene** brings back all of the scene's windows to the screen.

The format of the file created by **Save Scene** is a CLI script containing Geometry Viewer CLI commands. See the section "Geometry Viewer Commands" section of the "Command Language Interpreter" chapter of the *AVS Developer's Guide* for more information on these commands.

Note: The saved *.scr* files contain references to objects and geometries, rather than copies of them. For this reason, be careful not to disturb *.geom* files that store the "building blocks" for your objects.

Note: You must use **Save Scene** to save exactly what you see in a view window. Using **Save Object** saves just the object, its properties, and the position and orientation of the object in space. It does not save light or camera positions.

Hardware Renderer/Software Renderer

Selects which renderer will be used to draw the scene for the current camera. (See the discussion on "Renderers" near the beginning of this chapter.) It is possible to have a hardware renderer drawing the contents of one camera's window, and the software renderer drawing the contents of a second camera's windows. This second camera may be viewing the same scene of objects, but with different rendering options, such as transparency, in effect. If the platform only supports software rendering, or if **-nohw** or **NoHW** was specified at startup, then **Hardware Renderer** will be shaded out.

Some platforms may support additional renderer options.

Depth Cue

Note: This control is renderer-dependent. The software renderer supports depth cueing of lines. Systems may depth cue lines, polygons, and/or spheres.

This setting causes lines to "fade away" as they get more distant from the viewing position, enhancing the illusion of three-dimensional depth in the scene window. The Camera Options panel described below can control the locations where the depth-cueing effect begins and ends, and the amount of fading applied.

Z Buffer

This setting causes AVS to take into account the fact that some objects may block your view of other objects. By performing some extra up-front calculations, AVS can save time overall by drawing only the portion of each object that currently is visible (not obscured by other objects).

This function is not supported on all renderers. On some systems, Z buffering applies only to lines, not to surfaces.

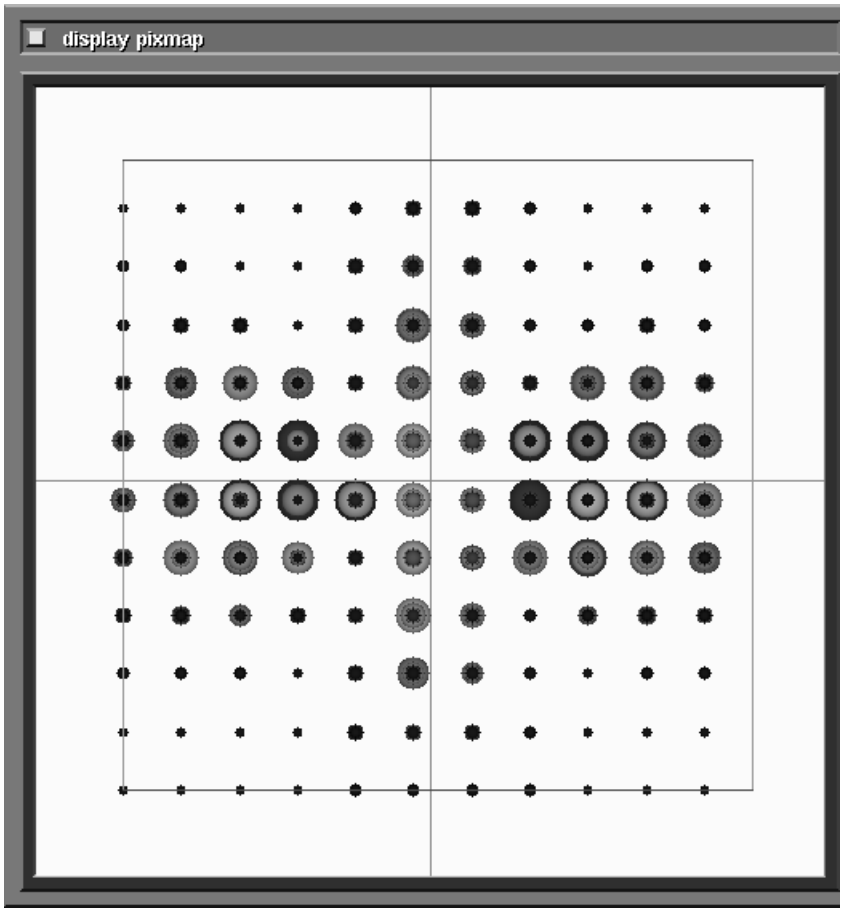


Figure 5-31 Bubbleviz hydrogen.dat without Perspective

Perspective

This setting causes the current window to use a **perspective** viewing projection (the default is to use a **parallel** projection). See Figure 5-31 and Figure 5-32. The difference becomes most apparent when you scale the view volume. (Select **Transform Camera**. Then use the middle mouse button together with the SHIFT key to change the size of the view volume. For more on transformations, see the section "Transformations and the Transform Selection Area" above.)

The degree of **Perspective** exaggeration (45 degrees) is adjustable with the Camera Options panel. **Perspective** also affects the apparent "location" of the camera. The camera seems to be much closer to the object, and its clipping planes are similarly shifted inwards.

The way to have a camera seem to zoom inside an object is to turn **Perspective** on, then translate the camera or object with the shift-middle mouse button.

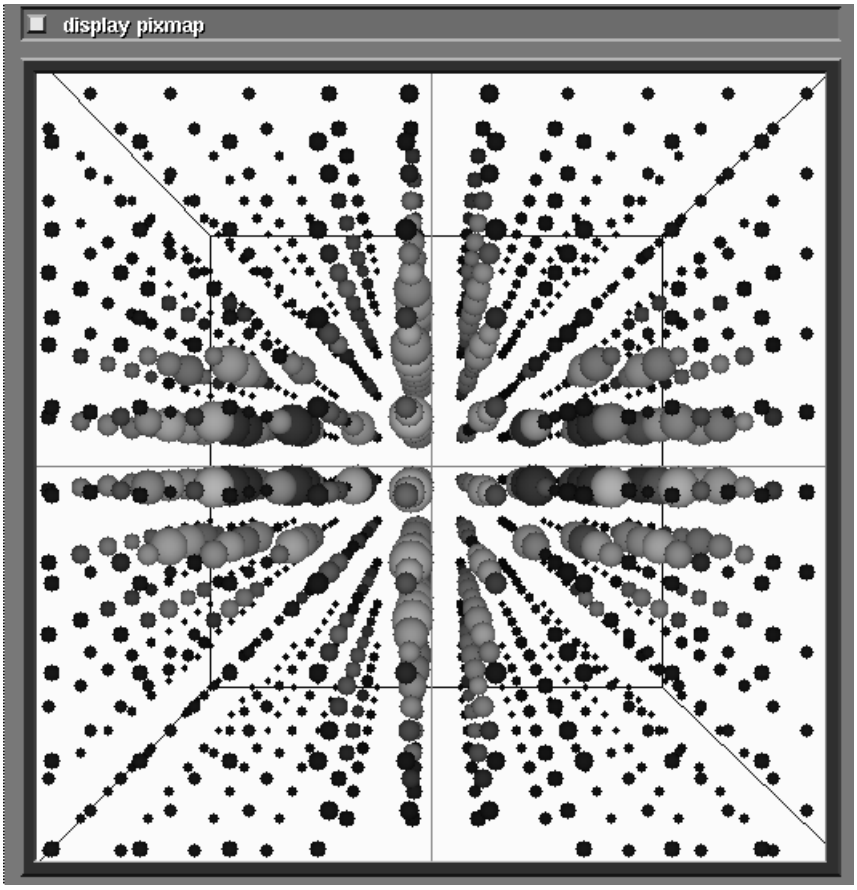


Figure 5-32 Identical Scene with Perspective

Accelerate

Note: This control only appears on systems that are able to save and restore objects in an offscreen Z-buffer.

This choice is most useful when you are manipulating one object in a complex scene. All objects (except the current object) are rendered once into offscreen memory. As you manipulate the current object, portions of the offscreen memory are copied to the screen, as needed, instead of being re-rendered.

Axes for Scene

This choice toggles display of XYZ axes in the current scene. The right-hand coordinate system indicated by these axes is the "world coordinate system" for the scene. The axes extend from +5 to -5 in XYZ.

Front/Back Clipping

This choice toggles the use of front and back clipping planes. When clipping is enabled, objects disappear as they move either very close to the eyepoint or

very far away. When clipping is disabled, the front and back clipping planes still exist, but they are so distant that in most cases no front/back clipping takes place.

The actual "location" of the clipping planes depends upon how the camera has been scaled, and whether **Perspective** is turned on or not. Clipping is defined in the camera's view volume, not world coordinates. When a camera is first created, the clipping planes are at Z=100 and Z=-100 in world space when clipping is turned off; and at about Z=5 and Z=-5 in world space when clipping is turned on. If you also turn on **Perspective**, the clipping planes move inward to about Z=10 and Z=-10 with clipping off, and to about Z=3 and Z=3 with clipping turned on. Such statements rapidly lose meaning as you begin to transform objects, the camera, and the Top level coordinate system. Suffice to say that clipping off puts the clip plane very near the camera itself, and clipping on puts it out in front of the camera.

The location of the clipping planes can be controlled precisely with the Camera Options panel.

Double Buffer

Note: This control only appears on systems that allow the user to control whether object are rendered directly into the window or into an offscreen pixmap.

When **Double Buffer** is on, objects are first rendered into an offscreen pixmap, the transferred to the display window. If it is off, objects are rendered directly to the display window.

On some platforms, it may be necessary to turn **Double Buffer** off in order to produce the highest color resolution that the display is capable of. These systems are dividing the color planes (24/12 or 8/4) between the buffers.

Sort Transparency

This feature is only supported on some renderers. Most transparency algorithms have problems rendering scenes that contain overlapping transparent surfaces. The **Sort Transparency** button can be used to correct the artifacts caused by these problems.

Global Antialiasing

This feature is only supported on some renderers. When this feature is enabled for a camera, an antialiasing technique is applied to remove the jagged edges effectively increasing the resolution of your image.

Polygonal Spheres

Some renderers support true hardware sphere rendering. The software renderer emmulates true sphere rendering. These renderers can be directed to disable true sphere rendering and use sphere rendering by polygonal surface

approximation instead by toggling **Polygonal Spheres**. The number of polygonal surfaces that will be used is controlled by the **Subdivision** slider on the **Objects** submenu.

Freeze Camera

This function can be used to implement a different style of interaction that is useful when the scene takes so long to render that even the use of **Bounding Box** mode is too slow. When you select **Freeze Camera** for a camera, the camera will not be updated until one of three conditions occurs:

- the camera is exposed by raising it or moving another window across it
- the **Freeze Camera** button is turned off for the camera
- you click the mouse in the red border of the camera

This mode is most useful when **Bounding Box** mode is also enabled. You can rotate, scale and translate several objects using the bounding box extent information to get an idea of the position, size and orientation of the object, then hit the border of the window to cause this window to refresh.

Show Camera

This function can be used to control the visibility of the camera window. It is particularly useful when you are using the **geometry viewer** module's image output port in a network in which you do not want to see the results directly on the screen. For example, you either want to see the results after a filtering operation, or want to simply process the resulting image further.

Note: the software renderer should be used for any cameras with **Show Camera** turned off. Most hardware based renderers are not capable of producing an image with an invisible image window, or will not render the portion of an image that is off (or larger) than the display screen.

Camera Width/Height Typeins

These typeins on the camera menu allow the user to specify or determine the width and height of the current camera window. In particular, the user can enter new values to specify a new size for the window. For example, to increase the resolution of an image for animation or higher quality PostScript output, you may set the size of the camera's window to be larger than the display screen. When doing this, also use the **Show Camera** and **Software Renderer** options to force rendering, since most hardware renderers will not render off-screen or obscured portions of their windows.

Be careful not to choose a size that is too large for the memory configuration of your system. The software renderer requires at least six bytes per image pixel.

Edit Background Color

At the bottom of the **Cameras** submenu, the **Edit Background Color** button will pop-up an color editor with RGB and HSV sliders for setting the background color of the current window. See "Edit Property" above for an explanation of how to use the sliders.

The default background color for all windows is black. You can set the background colors (as well as the location and size) for a sequence of windows by specifying a defaults file with the command-line option **avs -geometry -defaults**. See "Command Line Options in the "Starting AVS" chapter for details.

Camera Options Panel

The Camera Options panel (see Figure 5-33) allows the user to specify or determine position and depth cueing attributes of the current camera. It is raised by clicking on the dimple next to the **Transform Camera** button.

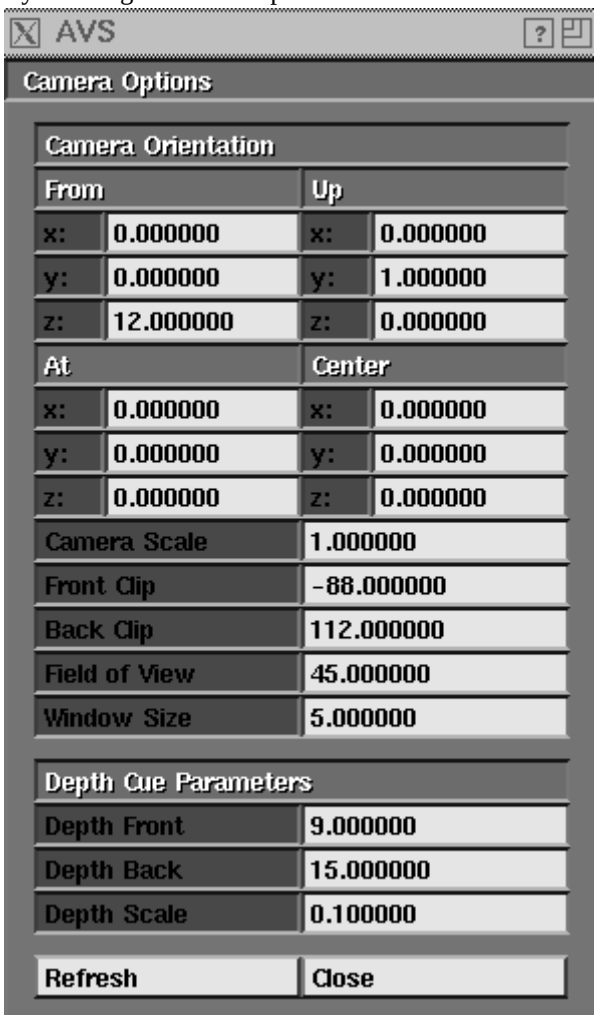


Figure 5-33 Camera Options Panel

The **Camera Options** panel displays the current values for the camera position parameters:

From

This point contains the X, Y, Z position of the camera in world coordinates. If the **Perspective** button is enabled, it is the "eye" position of the camera. In both perspective and parallel projections, it defines the base point for the front and back clipping planes and the depth cueing parameters. (See Figure 5-34 and Figure 5-35.)

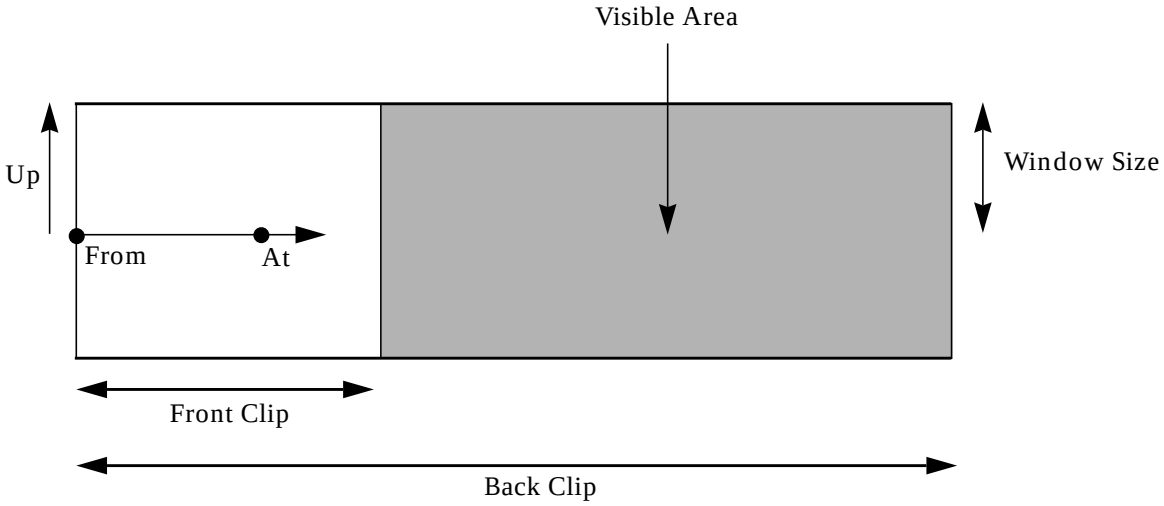


Figure 5-34 Parallel Camera and Camera Options

At

This point is used in conjunction with the **From** point to determine the view direction or Z axis of the camera. A line in world coordinates that connects **From** to **At** would be drawn as an end-on line directly in the middle of the camera window.

Up

This is a direction vector that determines the direction of the vertical axis of the camera. A line that is parallel to the **Up** vector in world coordinates will become a vertical line in the window. It is important that this direction vector be perpendicular to **From** - **At** (the view direction) or the camera will "shear" the objects.

Center

This is a point that determines the rotation center of the camera.

Camera Scale

This is a scale term that is applied to camera. This value is changed when you scale the camera. Since the scale is applied before the projection parameters, you will need to scale the front clip, back clip, depth front, and depth back values by this number if it is not 1.0 in order to get values in world coordinates.

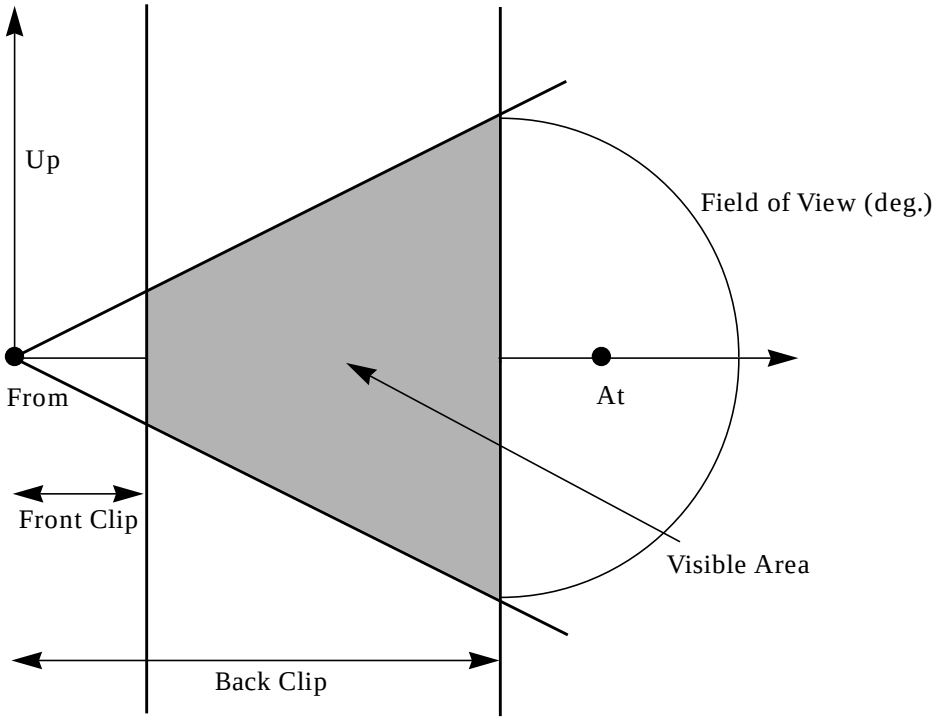


Figure 5-35 Perspective Camera and Camera Options

Front Clip

This is a distance value in world coordinates that specifies the distance from the **From** point along the view direction (towards **At**) before the front clip plane. This distance must be greater than zero for perspective projections but can be negative for parallel projections.

Back Clip

This is a distance value in world coordinates that specifies the distance from the **From** point along the view direction (towards **At**) before the back clip plane. This distance must be greater than **Front Clip** for both parallel and perspective cases.

Field of View

This value specifies the angle formed by the edges of the view volume for perspective projections. It must be a value between 0 and 180 degrees. This attribute is not used for parallel projections.

Window Size

This is a scale factor for parallel projections that determines the range of viewing. It is defined in world coordinates.

The **Camera Options** panel also controls the depth cueing parameters:

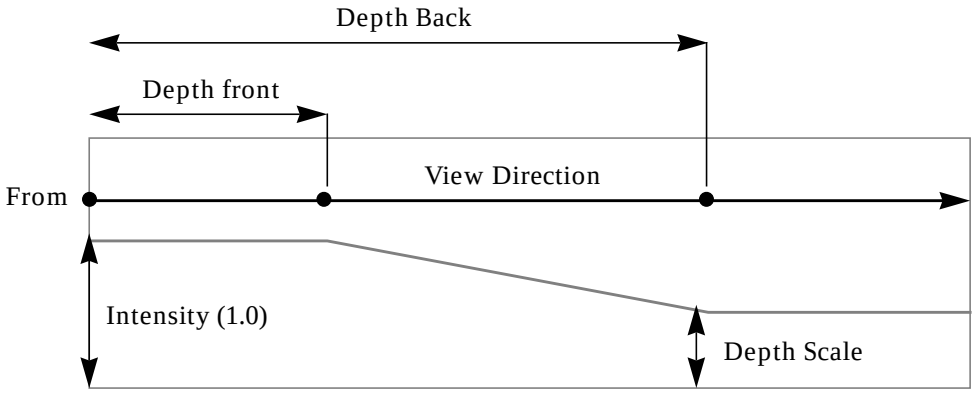


Figure 5-36 Depth Cueing Parameters

Depth Front
This parameter specifies the distance in front of the **From** point in the viewing direction (towards **At**) where depth cueing begins. (See Figure 5-36.) Any objects that are closer to the **From** point than this distance will be drawn at full intensity.

Depth Back
This parameter specifies the distance in front of the **From** point in the viewing direction (towards **At**) where the maximum depth scale is applied.

Depth Scale
This value is the factor that is applied to locations farther from the viewer than the **Depth Plane**.

It is usually the case that **Depth Front** is larger than **Front Clip** and **Depth Back** is smaller than **Back Clip**. Also, **Depth Front** should always be smaller than **Depth Back**.

Labels

The **Labels** menu selection (Figure 5-37) provides access to the Geometry Viewer's annotation text facility. You can attach one or more **labels** to any object. Each label consists of a single line of text. As you manipulate the object—move it, resize it, temporarily hide it, permanently delete it, etc. — the object's label(s) react accordingly.

You have considerable typographic control, with a wide range of fonts, type styles, sizes, and colors to choose from. You can also control the position of each label relative to its associated object; one alternative is to have the label become a **title**, which always appears at the same location in the window, no matter how the object is transformed.

Creating Labels

To create a label, first make sure the object to be labeled is the current object. If necessary, click on the object with the left mouse button. Then, click the **Labels** menu selection to bring up the Labels submenu.

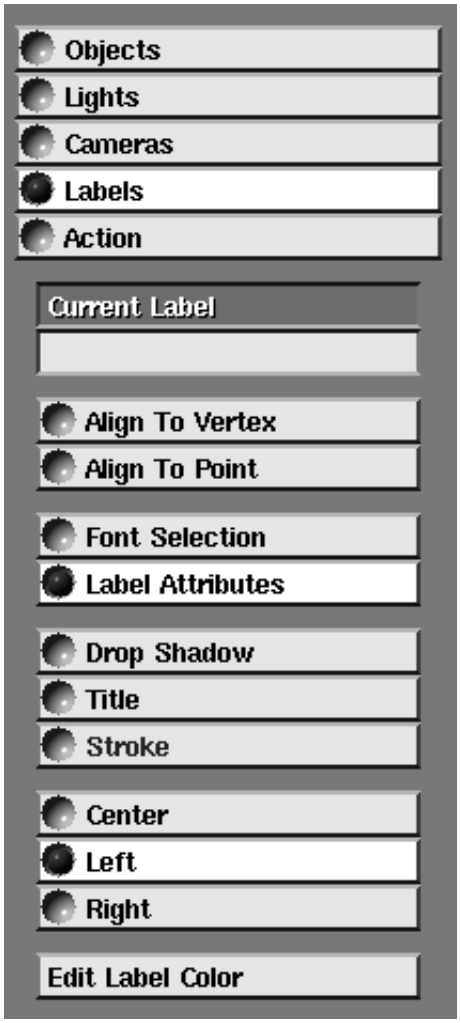


Figure 5-37 Labels Menu Selections

Place the cursor in the empty box below **Current Label**, and type any string of printable characters. Use **Backspace** (erase last character) and **Ctrl-U** (erase entire line) to make corrections.

Press **Return** or move the cursor out of the typein area when you've finished the label. When you do so, the label appears centered on the current object, surrounded by a red box.

Note: In some cases, part or all of the label may be obscured by the object itself. The red box, however, will always be visible. If you have problems with label visibility, turn off Z buffering under the **Cameras** menu selection.

To create additional labels for the same object, select the object again by clicking on it with the left mouse button. This clears the **Current Label** box. (In addition, you may want to check that the Current Object Indicator shows the object and its name.) As before, type in a text string and press **Return**.

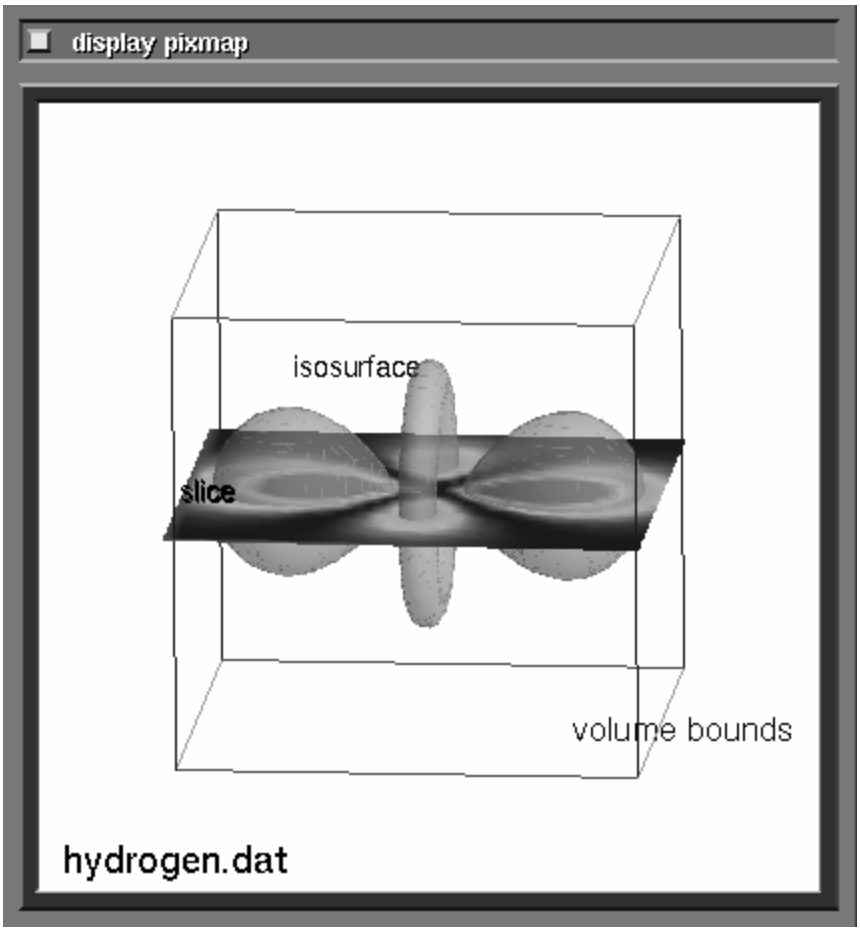


Figure 5-38 Scene with Three Labels and a Title

Labeling the Top Level Object

Labels you create for the Top level object apply to the entire scene — they will appear in every window you create for the scene using **Create Camera**. The "Transformations and the Transform Selection Area" section above describes the ways in which you can select the Top level object.

Picking and Moving a Label

Each of an object's labels is "attached" to a particular point in the object's coordinate system. Initially, this *base point* is the center of the object (that is, the origin of the coordinate system). You can move an existing label so that its base point is at a different X-Y-Z location:

- **Moving within the X-Y plane:** Click and hold down the left mouse button on the label. The red box reappears to confirm that the label has been

"picked". Drag the cursor to any other location, then release the button. This moves the base point parallel to the plane of the display screen.

- **Moving in the Z direction:** Hold down the SHIFT key, then use the left mouse button as described above. This moves the label perpendicular to the plane of the display screen. Note that this does not change the size of the label (but see "Changing Label Attributes" below).

The label's new location is still defined in terms of the object's coordinate system—you have simply changed the coordinates of the base point. As you move, resize, or rotate the object, it remains attached to its base point, and so moves around the display window.

Occasionally, a label may become obscured by the surface of an object. For example, it might come to be inside an isosurface. To access such a label, change the representation of the obscuring object to **Lines** or **Points** under the **Objects** submenu.

Align to Vertex/Align to Point

If you want to attach a label to one of the object's vertices, you needn't worry about separate movements in the X-Y plane and the Z direction. Just click the **Align to Vertex** selection, then drag the label using the left mouse button. Before you release the mouse button, make sure the cursor is on (or very near) a vertex. This causes the vertex to become the label's new X-Y-Z base point.

Align to Point aligns the label to the closest point on the closest adjacent polygon, rather than to a vertex.

Making a Label Into a Title

It is sometimes desirable to have one or more labels that are associated with an object, but which don't move around the screen as the object is transformed. Such labels are called **titles**. For instance, you might want a title string for an object to appear in the upper left corner of the window whenever the object is displayed. You can change any regular label into a title label by clicking the **Title** selection.

A title label "lives" in the window's X-Y coordinate system, rather than the object's X-Y-Z system. You can change the position of a title label using the left mouse button.

Editing/Deleting a Label

To change the text of a label, first click on the label with the left mouse button to make it appear in the **Current Label** box. Them move the cursor into the box and type the changes. As when you first create a title, **Backspace** erases the last character and **Ctrl-U** erases the entire label, deleting it from the scene.

Changing Label Attributes: Label Menu Selections

The annotation text facility includes a two-level function menu, which allows you to customize the appearance of each label. The Top level choices, **Font Selection** and **Label Attributes** are always visible. The submenu for whichever of these choices is currently selected appears below.

Font Selection Submenu

The submenu for **Font Selection** includes a list of fonts similar to the following. (The actual list of font names you will see varies from system to system.)

Courier
Helvetica
Schoolbook
Times
Charter
Symbol
Selects the font to be used for the label.

Bold

Italic
Selects the type style. You can click both of these choices to produce a bold-italic label. (Not all systems support **Bold** and/or **Italic** fonts.)

Label Height
Selects the point size of the label. Labels do *not* scale continuously; instead, AVS makes best use of the available X Window System fonts. As you move the slider to indicate a larger or smaller size (using any mouse button, by clicking or by dragging), the label size changes when a different font provides the closest fit.

The red box around the label *does* scale continuously to indicate the label's height at the requested size, whether or not a font of that size is available.

Label Attributes Submenu

The submenu for **Label Attributes** includes the following choices:

Drop Shadow
Creates a one-pixel drop shadow for the label. This can improve label readability. (Not all systems display this option.)

Title
Makes the current label into a title, whose position is defined in terms of window coordinates, rather than in relation to the object's 3D location. See "Making a Label Into a Title" above.

Stroke
(This control does not appear on all systems.) Some systems have an alternative means of rendering text via **Strokes**. This control turns on this feature.

Center
Left
Right
Specifies which part of the label is placed on the *base point*. Initially, it is the bottom left. The alternatives are the center of the label and its lower right corner.

Edit Label Color
An RGB-HSV color editor, similar to ones used elsewhere by the Geometry Viewer, allows you to specify the color of the label.

Action

The **Action** menu selection allows you to define "animations", which take the form of a sequence of geometries. You can append new frames to the end of the sequence, delete any frame within the sequence, and play back the sequence in a variety of ways. You can either define the sequence of geometries as a "cycle" in the Geometry Viewer Script Language (see the "Geometry Viewer Script Language" appendix for details), or you can create it dynamically by storing away geometries that are created from the network.

The **Action** submenu will only display two buttons if the current object does not have any flip-book geometry defined for it. These two buttons, **Store Frames** and **Append Frame**, are used to create flip-book animations from objects that are generated by modules. They are described later.

When the currently selected object does have a sequence of geometries defined for it, you will see the following menu choices.

Playing Back the Frames

The following functions provide a variety of ways of viewing the frames in an animation sequence:

Step Forward
Displays the next object in the cycle. When the end of the cycle is reached, you automatically wrap around to the first object.

Step Backward
Displays the previous object in the cycle. When the beginning of the cycle is reached, you automatically "wrap around" to the last object.

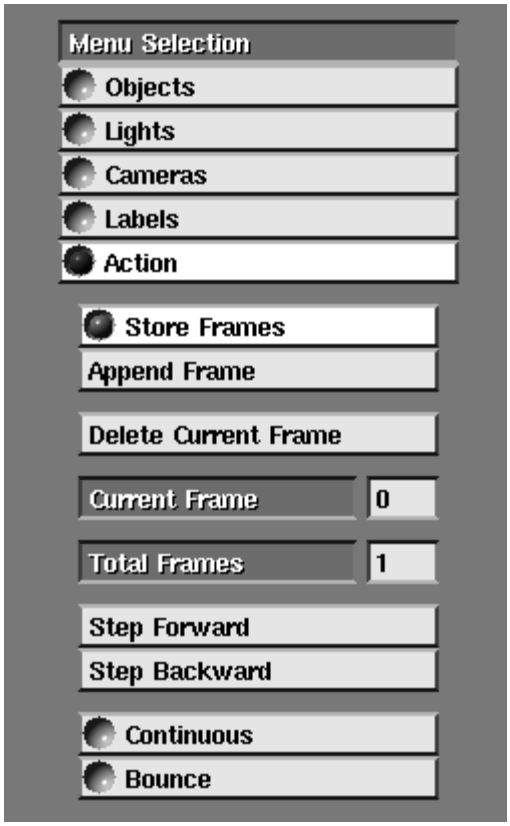


Figure 5-39 Action Menu Selections

Continuous
Continuously cycles forward through all the objects in the cycle. At the end of the cycle, the animation wraps around to the beginning automatically. To stop the animation, click **Continuous** again.

Bounce
Continuously cycles through the images, but alternates between going forward (beginning to end) and backward (end to beginning). To stop the animation, click **Bounce** again.

Some systems have a sample Action animation that you can play back in the file */usr/avs/data/geometry/math.obj*, which references a series of geometry files in the *math* subdirectory. Note that each frame is a distinct geometry. To play this sequence, use **Read Object** on *math.obj*. Then, make sure it is the current object. Bring up the **Action** submenu. The controls listed above should appear. If they do not, it is probably because the animation cycle is not the current object. Press any of the above controls to play the animation.

Adding Frames

There are two modes for adding new frames to the end of the sequence:

Store Frames

If you turn on this toggle switch, every new geometry sent to the **geometry viewer** module corresponding to the current scene for the current object will be appended to the frame sequence. The **Current Frame** and **Total Frames** counters are updated automatically.

Note: Main memory must be allocated for each frame. Make sure that your system has sufficient memory to accommodate all the frames.

Append Frame

This is a command, rather than a toggle switch. If **Store Frames** is turned off, you can click this button to add a frame to the sequence. The currently-displayed geometry is *not* added—rather, the next time a new geometry is sent to the **geometry viewer** module, it will also be added to the sequence.

This feature has some restrictions. Each time you read a new geometry, it replaces the existing geometry and the old object is discarded. (Reason: when a new geometry is added to a sequence, it must have the same name as the object being animated; that is, it must be a modification of the current object.) This means that you can't create an animation simply by clicking on a series of different names in the File Browser. You *can* create this kind of animation using the Script Language, however.

A frame contains a geometry definition only, not such attributes as the transformation, surface color, or material properties. Likewise, lighting information is *not* captured in a frame. This means, for instance, that you can't create an animation that shows an object going through a rotation sequence. (The rotation is a transformation attribute, not part of the object's geometry.)

Animating More Than One Object

It is fairly common to want to animate more than one geometric flipbook at the same time. To do this, create each object's animation sequence independently either using the **Store Frames**, **Append Frame**, or the Geometry Viewer Script Language. Make sure that each object has the same number of frames in its animation and that each object starts out on the same frame number if the animations are to be kept in sync.

Now set the current object to be the object called **top** (or any other object that contains all of the objects to be animated as children). When you press the **Action** command, you will see that you have access to the menu buttons: **Step Forward**, **Step Backward**, **Continuous**, **Bounce** and **Delete Current Frame**. When any one of these buttons is pressed, it will perform the function for each of the child objects that have a defined animation. For example, if you choose the **Step Forward** button, each of your several objects will advance a frame. If you choose **Continuous**, all of the objects will begin continuously animating at the same time.

You cannot select **Store Frames** or **Append Frame** on a parent object and have that object automatically store the frame for each individual child. You must create the animations of the leaf objects individually.

Deleting Frames

Clicking the **Delete Current Frame** button deletes one frame. (There is no way to delete a range of frames.) Typically, the current frame is the one most recently added to the sequence, but you can make any frame current. Use **Step Forward** or **Step Backward** to move to a particular frame. Alternatively, go to the **Current Frame** box, use **Backspace** or **Ctrl-U** to erase the number already there, type a new number, and press **Return**.

Geometry Viewer Command Language Interpreter

It is possible to drive the Geometry Viewer with the AVS Command Language Interpreter (CLI) rather than the X display interface. The commands can be either typed in interactively from a terminal emulator window while AVS is running, they can be read from a script file, or they can be sent from a user-written module.

Note: The Geometry Viewer Command Language Interpreter is **not** the same thing as the Geometry Viewer Script Language.

This opens many possibilities:

You can create scripts that animate the Geometry Viewer. not just the network-produced images within it, producing demonstration, illustration and test scripts.

To run AVS with the Command Language Interpreter (CLI) active, type this:

```
avs  -cli  other-options
```

This starts AVS as usual, but also starts the CLI command line interpreter in the invoking window. (You might have to press carriage return to get the **avs>** prompt.)

To get a list of the Geometry Viewer CLI commands, type the following:

```
avs>  help Geometry
```

This produces a list of Geometry Viewer CLI commands. To get help on an individual commands, type "help" plus the command name:

```
avs> help geom_set_matrix
geom_set_matrix Sets a transformation for an object, camera, or light.
    Usage: geom_set_matrix {-object<name>}{-camera{1-n}}...
                .
                .
                .
avs>
```

Many of the AVS Demo scripts in */usr/avs/demosuite/General/Geom* contain Geometry Viewer CLI commands. When you save an object or a scene with

Save Object or **Save Scene**, the resulting *.scr* file is written out in Geometry Viewer CLI commands.

The Command Language Interpreter and the Geometry Viewer set of CLI commands are documented in detail in the "Command Language Interpreter" chapter of the *AVS Developer's Guide*.

High Quality Image Output

One often wishes to capture the contents of the Geometry Viewer window in a PostScript file for presentation purposes (documents, lecture slides, etc.) There are several techniques to do this using the **geometry viewer** module's image output port.

image to postscript module

Connect the image output port to the **image to postscript** module to convert the contents of the current scene window into a monochrome or color PostScript file. You should usually be using the software renderer when attempting this, because the software renderer will output a 24-plane true color image, even on 8-plane systems. If memory space permits, you can improve the quality of the output image by enlarging it significantly with the **Camera Width/Height** typeins. Again, the software renderer should be used to force output of obscured or off-screen areas.

See the **image to postscript** man page for information on image scaling and orientation.

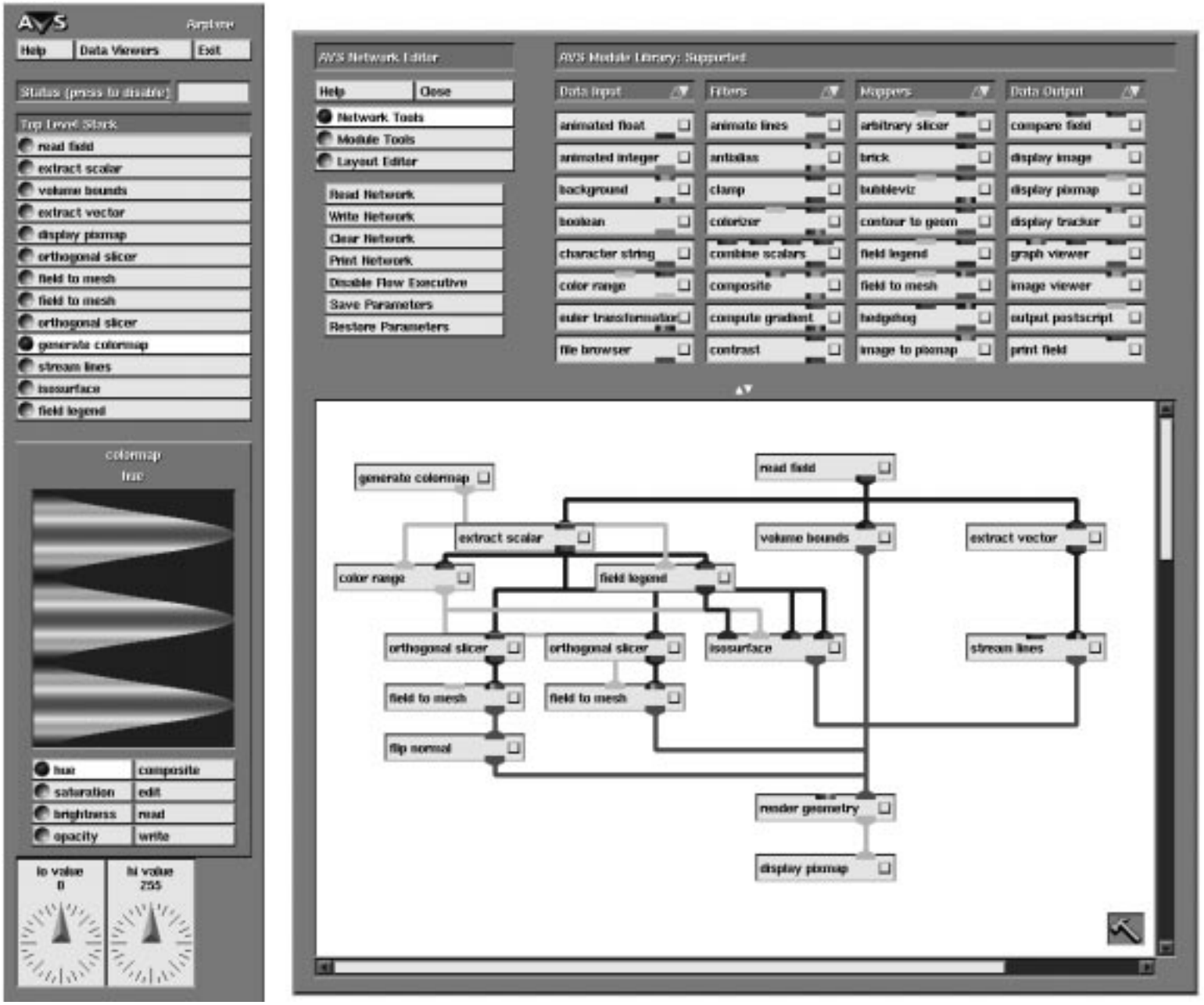
geom_save_postscript CLI command

The main drawback to using the **image to postscript** module is that it outputs at the resolution of the screen (approximately 100 dpi), not at the resolution of many PostScript printing devices (300 dpi and greater). Enlarging the image as described above is an effort to overcome this basic fact.

The **geom_save_postscript** CLI command will output the contents of the Geometry Viewer scene window using PostScript text and line-drawing primitives where possible, improving the quality of labels and lines in the final image. (There are no PostScript primitives for shaded polygons.) You must be using the software renderer.

At present, **geom_save_postscript** has no Geometry Viewer button that represents it; it must be typed to the CLI monitor. See the discussion on **geom_save_postscript** in the "Geometry Viewer" section of the "Command Language Interpreter" chapter of the *AVS Developer's Guide* for particulars.

The image output port is also used with the optional AVS Animation Application, if present, to send images to the various Animation image sequence storage and processing modules.


AVS
Runtime
Help
Data Viewers
Exit
Status (press to disable)
Top Level Stack
read field
extract scalar
volume bounds
extract vector
display pixmap
orthogonal slicer
field to mesh
field to mesh
orthogonal slicer
generate colormap
stream lines
isosurface
field legend
colormap
hue
hue
saturation
brightness
opacity
composite
edit
read
write
lo value
0
hi value
255
AVS Network Editor
Help
Close
Network Tools
Module Tools
Layout Editor
Read Network
Write Network
Clear Network
Print Network
Disable Flow Executive
Save Parameters
Restore Parameters
AVS Module Library: Supported
Data Input
animated float
animated integer
background
boolean
character string
color range
euler transformator
file browser
Filters
animate lines
antialias
clamp
colorizer
combine scalars
composite
compute gradient
contrast
Mappers
arbitrary slicer
brick
bubbleviz
contour to geom
field legend
field to mesh
hedgehog
image to pixmap
Data Output
compare field
display image
display pixmap
display tracker
graph viewer
image viewer
output postscript
print field
generate colormap
read field
extract scalar
volume bounds
extract vector
color range
field legend
orthogonal slicer
orthogonal slicer
isosurface
stream lines
field to mesh
field to mesh
flip normal
render geometry
display pixmap

CHAPTER 6

NETWORK EDITOR SUBSYSTEM

Introduction

AVS's **Network Editor** subsystem is a visual programming interface for creating, testing, and revising AVS visualization networks. It supports these major features:

- You can save the visualization networks that you create as a visualization application.
- The applications can be made accessible from the main menu's **AVS Applications** submenu. (The mechanism for doing this is described at the end of the "Starting AVS" chapter.)

This chapter explains the basics operations necessary to use the Network Editor. The "Advanced Network Editor" chapter later in this *User's Guide* describes these additional Network Editor features:

- You can use the Network Editor's Layout Editor to redesign the user interface to a network, so that others can perform visualization tasks without having to be knowledgeable about network construction.
- Networks can contain a mixture of modules that execute locally, and modules that execute on a remote host that runs AVS . The remote host can be "heterogeneous," that is, of a different hardware type than your workstation.
- Where multiple processors are available, modules can execute in parallel.
- The user can see and control which process a module executes in.
- Sets of modules can be interactively grouped together to form one **macro** module. Macro modules behave in the same way as individual modules. The macro module can be added to the module Palette, saved, read in, and made part of a module library.
- You can create libraries of modules to support an individualized repertoire of visualization functions.

The Network Editor can also be driven by the AVS Command Language Interpreter (CLI), either by typing Network Editor CLI commands to the CLI prompt, by reading a CLI script file, or from a user-written module that sends CLI commands to the Network Editor via the AVS kernel. Furthermore, the Network Editor supports a **journaling** feature. You can record your Network Editor interactions into an ASCII file, which you can then edit to produce a demonstration script that can be later replayed. This is how the **AVS Demo**

suite scripts described below were created. See the "Command Language Interpreter" chapter in the *AVS Developer's Guide* for information on this functionality.

Starting the Network Editor

To start the Network Editor subsystem, click the **Network Editor** button on the AVS main menu. If you're not currently at the main menu, you can return there by clicking the **Close** button at the top of the current subsystem's main control panel.

A **Network Control Panel** window appears along the left edge of the screen. Initially, this window is empty, since no network is currently active.

The remainder of the screen is used for the **Network Construction** window, which is divided into an upper part and a lower part. The upper part is shared by the **Network Editor Menu** and the **Module Palette**. The lower part is the **Workspace** in which you build networks (Figure 6-1).

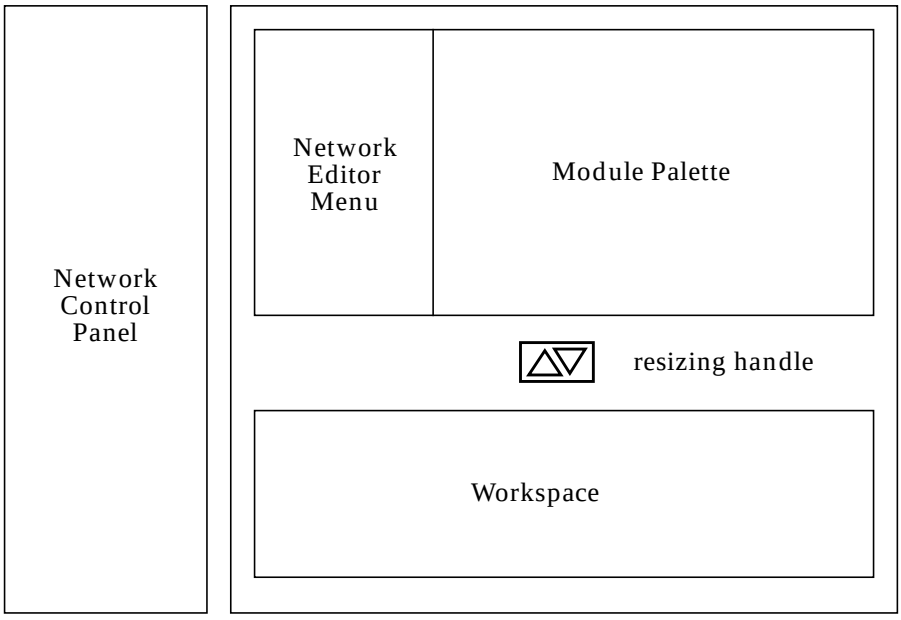


Figure 6-1 Network Control Panel and Network Construction Window

Getting Help

On-line help for the Network Editor is available via two **Help** buttons. The **Help** button in the **Network Construction** window brings up a Help Panel. The Help Panel has two features:

- It contains a scrolling browser of Network Editor usage topics. Clicking on one of these topics makes the help text appear in the Help window.
- It contains a **Help Demos** button. Clicking on this produces a browser of automatic scripts that illustrate various visualization modules performing in networks. The scripts load a sample network, and run some interesting data and parameter settings through it. When the script is finished, it leaves the network in the Network Editor so that you can experiment with it yourself. The next script read in will clear the previous network. You must get rid of the last network manually, either by hitting **Clear Network** on the Network Editor's panel, or by simply **Exit**ing the Network Editor.

  While the script runs, you can cause it to **Pause**, **Continue**, or **Abort**.

  See the "Using On-Line Help" section in the "Starting AVS" chapter for details on using on-line help.

The various AVS **Demo** suite scripts, accessible from the main **Applications** menu, also illustrate the Network Editor in use. Like the **Help Demos**, these demonstration scripts leave the networks in the Network Editor so that you can experiment with them further. The **Demo** suite is a good way to learn which modules are connected together in which patterns to produce useful visualizations of different types of data.

To get help on individual modules, their purpose, ports, and parameters, click on the module icon's "dimple" with the middle or right mouse button. This brings up the **Module Editor** panel. Then click on **Show Module Documentation**. This will display the module's "man page" description that includes a sample network.

The **Help** button in the **Network Control Panel** window will probably be of limited use—when you click it, AVS tries to locate a help file named after the currently-active network. No such help files are supplied with this release.

Closing the Network Editor

The Network Editor subsystem has two top-level windows, which can be closed separately:

- Click the **Exit** button at the top of the Network Control Panel window to exit the Network Editor and return to the AVS main menu. Unlike the other AVS subsystems, this is a true exit. Work will be lost if you do not save it. The Network Editor raises a dialog box on the screen that gives you a chance to cancel the Exit so that you can save your work (e.g., **Write Network**) before it exits. Any current work is lost, so be sure to save your work first.
- Click the **Close** button at the top of the Network Construction Window to close that window without destroying any work. This button is useful when you finish building a network and want more screen space for executing the network (e.g. for manipulation of the network's display windows).

Clicking this button causes a **Display Network Editor** button to appear at the top of the **Network Control Panel** window. This allows you to reopen the Network Construction Window at a later time.

Switching Subsystems

In many situations, you'll want to switch to the other AVS subsystems without losing your current Network Editor work. For example, if you create a network that displays a geometry, you may want to modify the rendering method or the lighting with the Geometry Viewer. There are two ways to go directly from the Network Editor to the other subsystems:

- Position the mouse cursor over the **Data Viewers** button at the top of the Network Control Panel. Press *and hold down* any mouse button. A pop-up menu appears listing the other subsystems. Roll the mouse cursor down to the subsystem you wish to start, then release the mouse button. The control panel for the selected subsystem will appear *on top* of the Network Control Panel. You can return to the Network Editor by clicking the **Close** button at the top of each subsystem's control panel, or simply use your window manager to move the control panel to another part of the screen.
- If a network includes the **geometry viewer**, **image viewer**, or **graph viewer** modules, use the left mouse button to click the small square box in the icon for the module.

Status Widget

At the top of the Network Control Panel is a Status widget.

Figure 6-2 Status Widget

This device shows which module is executing, and gives an approximate idea of its progress. If the widget shows 10%, it means that the module is receiving data. 90% means it is done processing the data and is outputting the result. Anything in between means that the module is executing. Individual modules can use this Status widget to give more precise statements of progress. The **read field** module does this. Not all modules take advantage of this.

Overview of Network Editor Usage

In general, creating a network includes these steps:

- Using the mouse to pull modules from the Palette into the Workspace, and/or reading already-existing networks from disk storage.

- Using the mouse to connect the modules' input and output ports. The connections define the network by specifying the flow of data among the modules.
- Adjusting the modules' input parameters using the widgets in the Network Control Panel.

These steps are described more fully in the sections that follow.

Using the Module Palette and the Workspace

When you first start AVS, the Network Editor loads its Module Palette (Figure 6-3) with one or more sets of AVS modules.

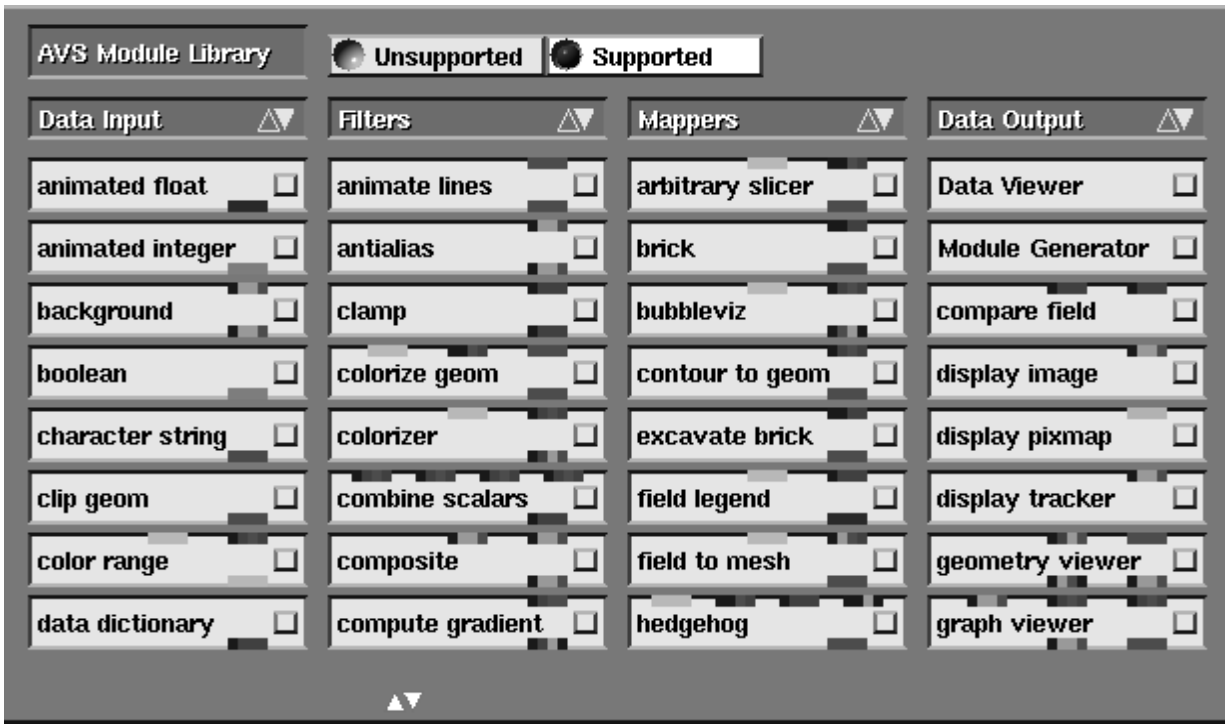


Figure 6-3 Module Palette

AVS uses its system-default startup file in */usr/avs/runtime/avsrc* to determine which modules to load. This startup file contains the following line:

```
ModuleLibraries    /usr/avs/unsupp_mods/Unsupported /usr/avs/avs_library/Supported
```

This instructs AVS to load two **Module Library** files: *Unsupported* and *Supported*. A module library is an ASCII file containing lines that describe the names of modules and where to find them. It is used to quickly construct the icons in the Module Palette and make them known to the AVS kernel without actually loading the binary module files.

The *last* module library specified is the set that will be showing when you enter the Network Editor; thus the system default *.avsrc* file lists the Supported library file last.

Supported and unsupported modules are documented in the *AVS Module Reference Manual*.

All loaded module libraries appear as icons across the top of the module Palette. You switch between the module libraries by clicking on their icons.

If you have a *.avsrc* or *.avsrc.X* file, then you can override the default list of module libraries in the system *avsrc* file. Place a **ModuleLibraries** line in your personal *.avsrc* or *.avsrc.X* file. AVS then loads modules from the library files specified. The last library listed shows as the default set in the Module Palette. Note that **ModuleLibraries** needs full file specifications, and that all of the libraries listed must be on one (perhaps very long) line.

You can also load any library by using the *-library* option on the AVS command line, or by loading the library interactively with the **Read Module Library** button under the Network Editor's **Module Tools** menu.

You can create your own module library files. These may contain your own modules, and / or be a subset of the AVS supported and unsupported modules containing only the modules you regularly use. This is discussed in the "Constructing a Module Library" section in the "Advanced Network Editor" chapter.

Module Types

The Module Palette includes an icon for each of AVS's computational modules. The modules are partitioned into four functional categories:

Data Input Modules

These modules introduce new data into an AVS network. Some modules (e.g. **read field**) read a data file from disk storage. Other modules (e.g **generate colormap**) create data according to the settings of their input parameters.

Filter Modules

These modules transform a numerical data set into another numerical data set. They perform such actions as sampling, subsetting, establishing threshold values, applying a linear transformation, etc.

Mapper Modules

These modules perform the "visualization" step—converting a numerical data set to a description of one or more displayable objects, either geometries or images. For instance, the **field to mesh** module creates a 2D surface in 3D space. It does so by interpreting each scalar value of a 2D array as the height of a point above a base plane. The collection of points defines (an approximation to) a 2D surface above the plane.

Data Output Modules

These modules produce the final output of the visualization process. In most cases, this is an on-screen image, displayed in its own window. Some modules store image data in image files for later display, or in PostScript files for printing.

Module Input/Output Ports

Each module icon shows the module's name, along with **input ports** and **output ports** to indicate the types of data that the module handles (Figure 6-4). The ports are color-coded to indicate the type of data that can pass through the port.

Figure 6-4 Module Icon

You need not memorize the color-coding scheme—AVS allows you to connect ports only if their data types are compatible. You can also display the ports' data types by clicking the small square **Module Editor** button on the module icon (the "dimple") with the middle or right mouse button. This pops up the **Module Editor** window (Figure 6-8), which displays helpful information about the module: a capsule description, the data type of each input and output port, a list of the input parameters, and which module procss and group it is running in. If you need further information on the module, click the **Show Module Documentation** button in the Module Editor window to display the entire manual page for the module in a help browser window.

The next few sections describe how to work with module icons using the mouse. For quick reference, here's a listing of how the mouse buttons work in this context:

- **Left Mouse Button**: Move one or more icons.
- **Middle Mouse Button**: Establish a connection between two icons.
- **Right Mouse Button**: Break an existing connection between two icons.

Finding the Module You Want

The Network Editor's standard module library includes more than 140 modules. This number is large enough so that you may not immediately see the module you're looking for at any given moment. And in some cases, there are too many modules in a particular category to fit in the vertical space allotted to the Palette. The following sections describe AVS's several facilities for handling such situations.

Scrolling a Module List

The icons in each category are listed alphabetically. If AVS cannot simultaneously display them all in the allotted space, it adds a scroll widget to the category's title bar:

Figure 6-5 Scroll Icon for a Module Category

One or both of the arrows are lit at any moment, indicating which way(s) the list can be scrolled. Clicking the left mouse button in the title bar scrolls toward the top of the list; clicking the right mouse button scrolls toward the bottom.

Incremental Search Through a Module Category

Each module category is organized alphabetically by module name. At any time (even when all the module icons in a category are visible), you can perform an incremental search through the names:

1. Put the cursor in the title bar of the category to be searched.
2. Type any character in the module's name. The list automatically scrolls so that the first icon containing that letter is at the top of the list
3. Now, there are two ways to continue searching:
   - Type the next letter in the module's name. The next icon containing the pair of letters scrolls to the top.

     For instance, to search for the **colorize** module, you might type "c" followed by "o", or you might type "i" followed by "z".
   - Press RETURN to continue the search on the current basis—that is, search for the *next* icon containing the letter you typed.
4. Any time the cursor is in the title bar of a category, you can press **Backspace** to scroll the category back to the top.
5. You can repeat the preceding step as many times as you like, either adding characters to the search string, or pressing **Return** to continue the search for the same string.

There is no need to explicitly end the search. Whenever you're finished searching, just stop typing. Similarly, nothing special happens if a search string fails to match any module—the list simply doesn't scroll.

Making the Module Palette Larger

In some cases, you may find it desireable to change the vertical space allotment for the Module Palette. Increasing it can reduce the need for scrolling the module lists. There is a "handle" marked with scroll arrows at the top of the Workspace area (see Figure 6-1). When you place the cursor on this handle, a message appears alongside it, explaining how to use it: grab the handle with any mouse button and move it downward or upward. That is, click and hold down the mouse button, drag the mouse, then release the button. This changes the partitioning of the Network Construction window between the upper area (Palette / Menu) and the lower area (Workspace). The **ModulePanelHeight** *.avsrc* keyword controls the same feature.

Moving Icons into the Workspace: Left Button

Use any mouse button to drag a module icon from the Palette to the Workspace. As you do so, the module's **control panel**—the set of widgets that control the input parameters—appears in the Network Control Panel window at the left side of the screen. (See Figure 6-6.)

Note: Do not drag a module from the Palette directly to the the Hammer in the Workspace in one continuous motion. This deletes the module from the Palette. If you do this by mistake, you can get a new copy of the module using the **Module Tools** submenu's **Read Module** button.

At the top of the Network Control Panel is a choice menu ("radio button" menu) labeled "Top Level Stack", which lists all the modules currently in the Workspace. At any moment, one module's control panel is visible. Click in the menu to bring any other module's control panel in view.

You can also bring up the control panel of any module in the Workspace by clicking the small square on the module icon (the "dimple") with the left mouse button.

It is useful to select control panels by clicking on the module icon's dimple with the left mouse button:

- When there are multiple instances of the same module. If you click on the module's dimple, there is no doubt which module's controls are displayed.
- If one of the modules is the **geometry viewer**, **image viewer**, or **graph viewer**, this is a quick way to bring up their respective control panels.

  When the **geometry viewer**, **image viewer**, and **graph viewer** modules are dragged into the Workspace, their control panels do **not** appear, nor are their names added to the Network Control Panel menu. The control panel for these modules are the entire Geometry Viewer, Image Viewer, and Graph Viewer subsystems, described in their own chapters.

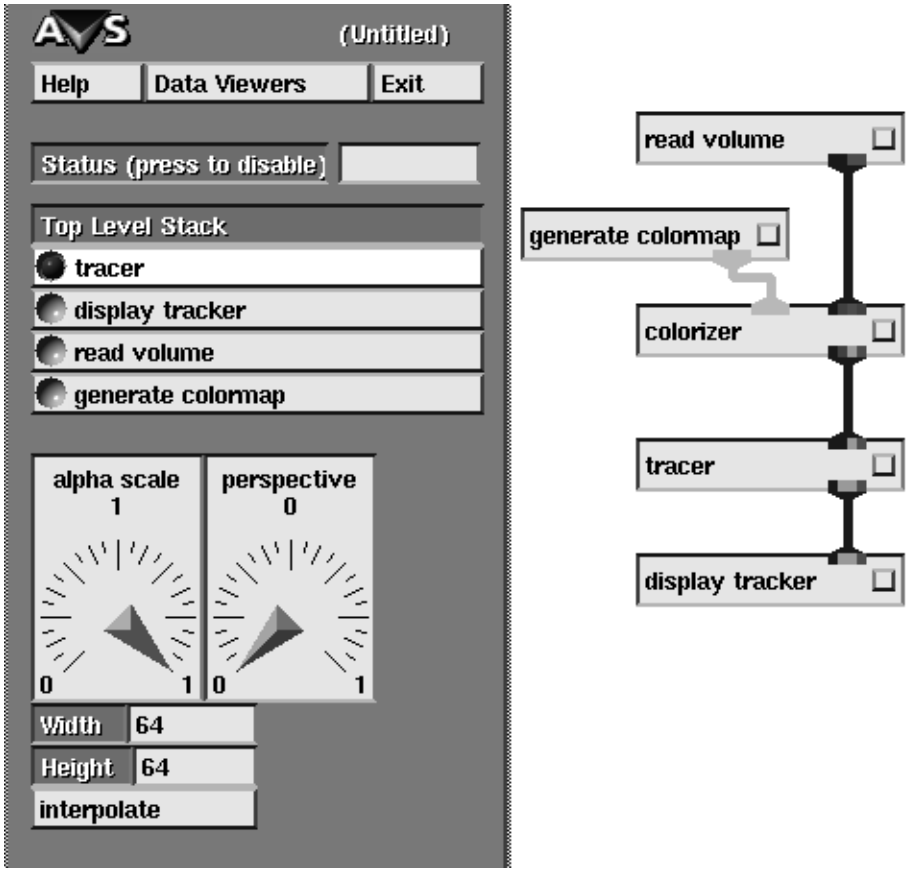


Figure 6-6 Network Editor Control Panel and Network

The appearance of the Network Editor's control panel can be changed. If your networks contain so many modules that the supplied control panel is not long enough to hold all of the radio buttons, you can change the radio buttons into a scrolling browser similar to a file browser (see the **StackSelector** keyword description in the "Starting AVS" chapter). You can also use the Layout Editor to make more module control panels visible simultaneously.

When you click the small square on these modules' icon with the left mouse button, the respective Viewer control panel appears, obscuring the Network Control Panel. To make it disappear, click the **Close** button at the top or use the left mouse button to click the small square of some other module icon in the Workspace. Another alternative is to move the Viewer control panel aside, using the X Window System window manager. This allows you to see both the Viewer's control panel and the Network Control Panel at the same time.

Moving Modules within the Workspace

Once a module icon is in the Workspace, you can move it around without breaking connections, again using the left mouse button. You can also drag a rectangular (Figure 6-7) lasso around several icons:

- Click and hold down the left mouse button when it is *not* on a module icon. This places one corner of the lasso.

- Drag the mouse to expand the lasso, fully enclosing one or more module icons.
- Release the button to complete the lasso.
- Press the left button again with the mouse cursor within the lasso area to drag the entire group to a different location in the Workspace.

To remove a lasso, click the left mouse button in the Workspace background.

In the Workspace, the right mouse button moves the entire network.

Deleting Modules from the Workspace

Use the left mouse button to drag a module icon onto the hammer icon in the lower right corner of the Workspace. You can also lasso several icons, drag the entire lasso area so that any part of it touches the hammer, then release the button.

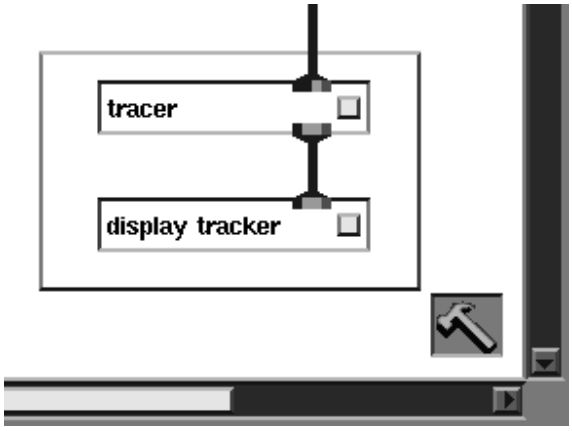


Figure 6-7 Two Lassoed Modules About to Be Deleted

Whenever you delete a module from the Workspace, any connections between its ports and those of other modules are automatically deleted, too. The deleted module's control panel disappears from the Network Control Panel.

"Hammering" an executing module causes that module to terminate execution. (However, see the cautions in the "Cancelling an Operation" section.)

Connecting Modules: Middle Button

The small colored bar(s) at the top edge of a module icon represent the module's *input ports*. (*Data* modules have no input ports, since they introduce new data into a network, rather than process data that is already there.)

Similarly, the colored bar(s) at the bottom edge represent *output ports*. (Many *Data Output* modules have no output port, since they don't pass any data to other modules. Instead, they either display an image on-screen or write data to a disk file or output device.)

The ports are color-coded to represent the type of data that can pass through. (See the "Importing Data Into AVS" chapter for a full discussion of AVS data types.) An output port can only be connected to an input port with a matching color. The color codes are defined under "Port Color-Coding" in the next section.

To make a connection:

1. Click and hold down the middle mouse button on one module's output port. AVS automatically displays thin lines that indicate all the valid connections to other modules' input ports.
2. Drag the mouse toward one of the valid destinations.
3. As soon as AVS highlights the connection you want to make (turns it *white*), release the button to complete the connection. It's not necessary to drag the mouse all the way to the destination.

If you release the mouse button before any of the possible paths is highlighted, no connection is made. You can also avoid making a connection by returning the mouse cursor to the original output port.

The same procedure works for making connections in the opposite direction—start on an input port and connect backward to another module's matching output port.

The details of connecting data ports is covered in the next section.

Disconnecting Modules: Right Button

The process of disconnecting modules is similar to connecting them, except that you use the *right* mouse button instead of the *middle* button. Click and hold down the right mouse button on a connected input (or output) port. Drag the mouse toward the other end of the connection, until the connection to be deleted is highlighted. Then release the button.

When you delete a module from the workspace (see above), all its connections are automatically deleted, too.

Completing a Network

A network is complete when it includes one or more modules that generate data, and one or more modules that display an image (or store data on disk). It can also include any number of modules that perform intermediate processing on the data.

Input/Output Ports

Modules have three kinds of input/output ports: data ports, parameter ports, and upstream data ports. Simple data ports are dicussed in the following sections. Parameter ports and upstream data ports are described in the "Advanced Network Editor" chapter.

Data Ports

Modules receive and transmit *data* through one or more **data ports**. Data are:

- the field, image, volume, unstructured cell, or molecule representations of scientific data;
- the colormaps that modules use to represent data as colors;
- and the pixmaps, images, and geometries that are the viewable constructs of data that modules produce.

Module data ports are always visible on the module icon.

The Module Editor and Parameter Editor Windows

Each module icon has a small square **Module Editor** button ("dimple") at its right edge. You can use the middle or right mouse button to click on this button to bring up the Module Editor window (see Figure 6-8).

The Module Editor window contains detailed information about the module including:

Name
The name of the module. If this module is a macro module, you can edit the name by typing into this field. If the module is not a macro, the name cannot be changed.

Host
This field contains the host name that the module is running on, or the string: (local) if the module is running on the local host.

Group
This field contains the module process group. This field can be used to optimize the process allocation of modules. When a module is created by placing it onto the workspace, in certain circumstances this module is created to be in the same process as an existing module. This can make module start up time faster and reduce data communication overhead between the modules (if shared memory is not available). The module group for the module can be used to override this feature. It is normally not necessary to use this attribute. See the "Advanced Network Editor" chapter later in this *User's Guide* and the "Advanced Topics" section of the

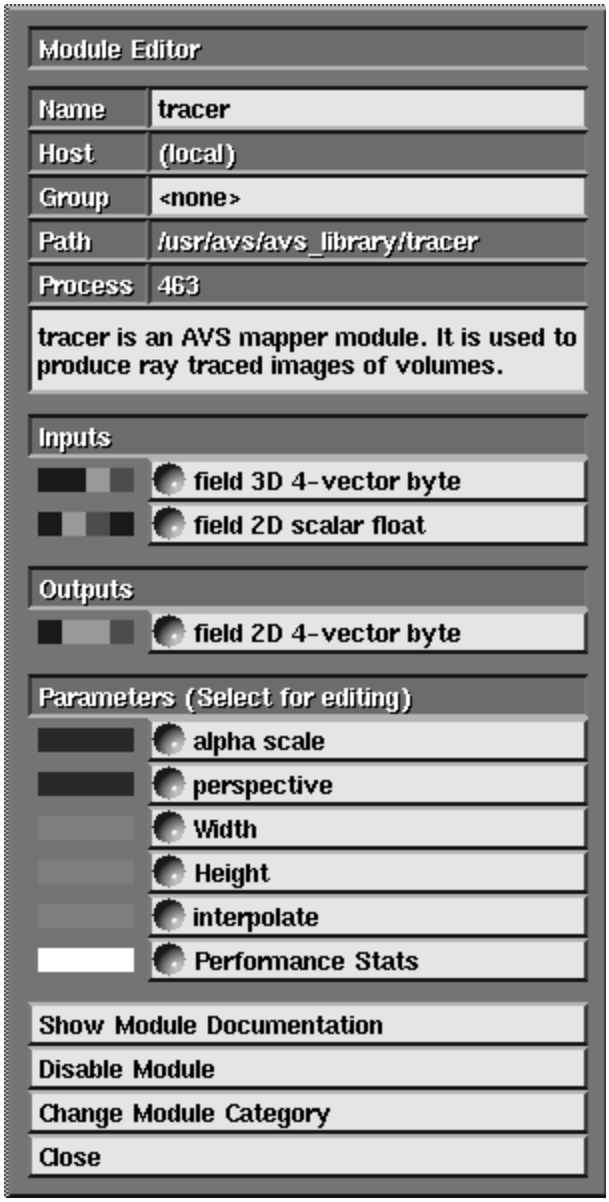


Figure 6-8 The Module Editor

Developer's Guide for more information on how to use this feature. The default is "(none)".

Path

This field contains the pathname of the file that is used to create the module. For a module that is builtin to the AVS kernel, this field is simply the string: "<builtin>".

Process

This field contains the UNIX process identifier for the module. If the module is a remote module, the process id of the process used to start the module is also displayed in this field.

Module Description

This window provides a first level of documentation for the module: a one-sentence summary description, descriptions of the input and output ports, and a listing of the module's input parameters.

Inputs

This section lists the inputs for the module (see the "Port Editor" section below).

Outputs

This section lists the outputs for the module (see the "Port Editor" section below).

Parameters

This section lists the parameters for the module (see the "Parameter Editor" section below).

Show Module Documentation

Displays the compete manual page for the module in a help browser window.

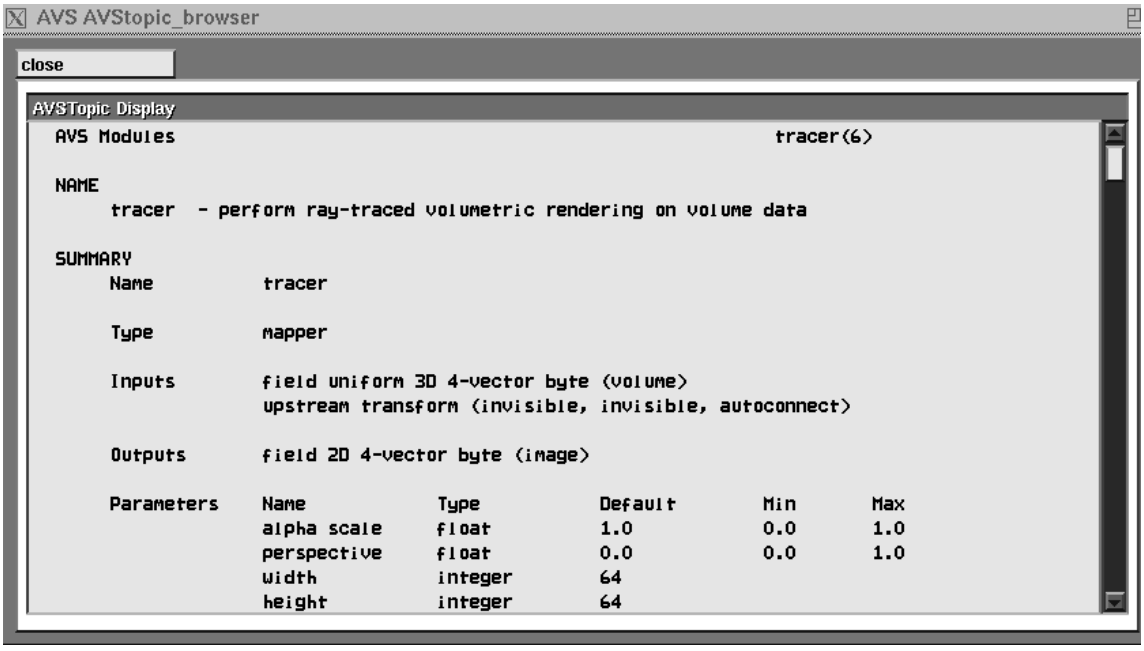


Figure 6-9 Module Help Browser

Disable Module

Temporarily inhibits the module from executing, preventing it from receiving or sending data. This may prevent execution of the entire network if there are no alternate data paths. The module icon turns red to indicate its disabled state.

To reenable the module, click this button again.

Change Module Category

Press down and hold the mouse button when the cursor is on this button and you will see a pop-menu that lists the different module categories that are available for this module. Move the cursor until it is over the new category for this module and then release the mouse button. If the module is in the module library Palette, the module icon will jump to the new category. If the module is a macro module that you are editing, it will appear in the category you selected when you finish editing the macro. If the module is an instanced module, there is no real change.

By default, the module categories are: Data Input, Filters, Mappers, Data Output. The current module categories are a property of the current module library.

This feature is useful when you are customizing a module library. You might, for instance, change the name of the "Data Input" category to "Field Modules" and then rearrange modules to and from this category, saving the results with the **Edit Module Library** panel's **Write Library** function.

Close

This button closes the Module Editor panel.

The Port Editor

If you have opened the Module Editor window from the Workspace (not from the Palette) you can bring up a Port Editor window for any one of the input or output ports of the module. The Port Editor lets you control the visibility of the input or output port on the module with the **Port Visible** button. If a port is not visible , any connections to or from it are not displayed by the network editor. You might want to hide some of the complexity of a network by hiding ports on modules. Occasionally modules will set default port visibility attribute to hide rarely used optional ports from the naive user. Modules that use upstream data (see the "Advanced Network Editor" chapter) often hide the upstream input and output ports to reduce the complex appearance of the network.

The Parameter Editor

Once you have opened the Module Editor window from the Workspace (not from the Palette), you can click on any of the input parameters to open its **Parameter Editor** window. This window allows you to change the control widget that is attached to the input parameter. For instance, you might want a parameter that currently is controlled by a dial to be attached to a type-in in-

stead. This would allow you to enter an exact value, such as 48.2, rather than using the mouse to fine-tune a dial setting.

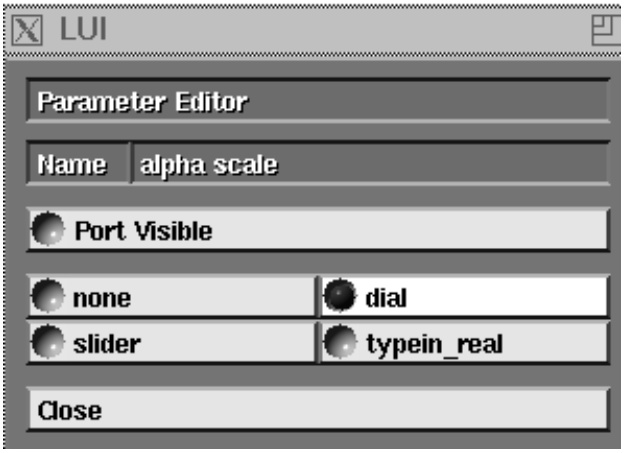


Figure 6-10 Parameter Editor

You also use the Parameter Editor's **Port Visible** button to make the normally-invisible parameter and upstream data ports visible (see the "Advanced Network Editor" chapter).

Port Color-Coding

red = *geometry*

A *geometry* is a geometric description of one or more objects (a "scene"). It can be created by a module or other program using calls to the AVS *libgeom* library (see the *AVS Developer's Guide*). Along with the definitions of the objects in terms of points, lines, triangles, spheres, etc., a geometry can include specifications for vertex and surface colors, lighting, rendering mode, transformations (translation, rotation, scaling), and transparency.

AVS includes conversion utilities that accept data in common formats and produce *geometry* files that can be read into a network with the **read geometry** module.

AVS also includes modules that dynamically convert "raw" data into geometries (e.g. the **field to mesh** module).

yellow = *colormap*

A *colormap* is a table that converts an integer value to a pixel value (i.e. to a color). Typically, you use the generate colormap module to create colormaps dynamically. This module also allows you to maintain a set of on-disk colormaps that you can load during network execution.

multi-color = *field*

A *field* is a very flexible data type, more like a collection of related types. A field is a generalization of the array structure that is used to represent many kinds of scientific data.

orange = *unstructured cell data*
Unstructured cell data (UCD) is commonly used in finite element analysis.

magenta = *molecule data type*
The molecule data type (MDT) stores information about molecules and their component atoms.

"Matching" has a special meaning in the case of the multi-colored field ports. For more information, see "Connecting Field Ports" below.

(For more information, see the "Importing Data Into AVS" chapter.)

Parameter data ports (described in the "Advanced Network Editor" chapter) are color-coded by this scheme:

```
medium purple     =  float
light purple      =  integer
grey blue         =  string
white             =  0 or 1 bit
```

Boolean and tristate parameters are implemented internally as integers. Thus their parameter ports show as light purple.

Connecting Field Ports

The AVS *field* data type is actually a general format—you can think of it as a collection of related subtypes. Fields can differ in their dimensions: 1D, 2D, 3D, etc. Fields can also differ in the type of data that is specified for each point: scalar byte, 4D vector of bytes, scalar float, etc., and in their coordinate systems (uniform, irregular).

The various field subtypes are incompatible: a module that outputs a 2D field cannot be connected to one that expects to input a 3D field; a module that outputs floating-point data cannot be connected to one that expects to input byte data.

AVS includes modules that convert field data from one type to another. For instance, the **field to byte** module accepts any field as input, and outputs a field whose data values are bytes. This may be necessary when you plan to use a module that accepts byte-valued fields only (e.g., **tracer**). Similarly, there are modules for handling dimension-based conversions. The **orthogonal slicer** creates a 2D slice from any 3D field. You can extract one scalar element from a vector field using **extract scalar**; you can assemble a vector field from scalar components using **combine scalar**.

In AVS, many modules have been generalized to accept a wide variety of input types. For example, **colorizer** accepts integer and floating point input values in addition to bytes.

As an aid in matching field subtypes, the color bars for field input and output ports are divided into four parts (Figure 6-11 and Table 6-1.):

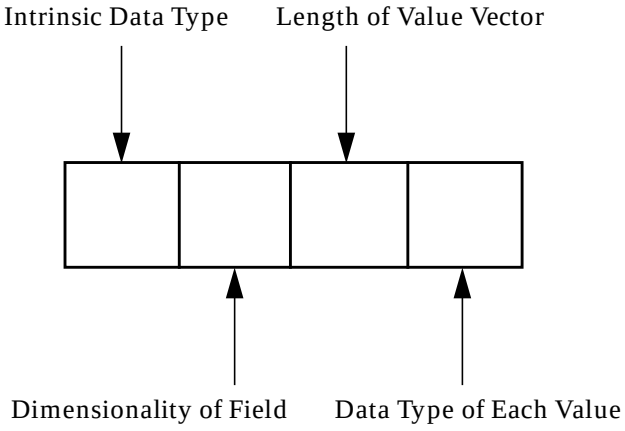


Figure 6-11 Color-Coding for Field Input/Output Ports

Table 6-1 Color-Coding for Field Input/Output Ports

	Color	Meaning
Intrinsic Data Type	blue	field
Dimensionality of Field	red	1-dimensional
	green	2-dimensional
	blue	3-dimensional
	gray	any dimensionality
Length of Value Vector at Each Point in Field	red	1
	green	3
	blue	4
	gray	any vector length
Data Type of Each Value	red	byte
	green	integer
	blue	single-precision floating point
	yellow	double-precision floating point
	gray	any of the above

Note that the color gray is a "wildcard": it indicates that the module is written to handle any of the supported alternatives. For example, if the second part of an input port color bar is gray, the module can accept fields of *any* dimensionality. This means that the color bars for field ports don't have to match *exactly* to be candidates for connection (Figure 6-12.).

AVS checks to see that a field to field connection is at least plausible. For example, if the sending module (A), outputs an unrestricted field, and the re-

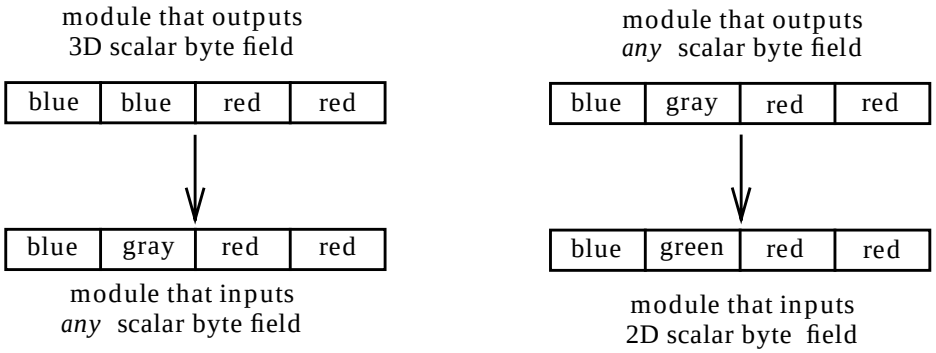


Figure 6-12 Gray Color-Coding as Wildcard Value

ceiving module (B) accepts only uniform scalar fields, it is plausible that A will send B the right thing, so the connection is allowed.

If A actually sends B an irregular vector field when the network executes, then AVS's runtime checking will catch it and send you a notice of the run-time incompatibility.

Controlling the Execution of a Network

As you build a network, its modules start to execute. In most cases, nothing useful will occur until the network is complete and you specify the input data to be visualized. Thereafter, the network re-executes every time ...

- ... you specify a different data set (or the data entering the network changes in some other way)
- ... you change the setting of a module input parameter.

To *stop* a network from executing automatically, press the **Disable Flow Executive** button on the main Network Tools menu (Figure 6-13.) **Disable Flow Executive** suspends network execution, allowing you to adjust several parameters before having the network reexecute.

You make these changes to the network's execution environment using the Network Control Panel window at the left edge of the screen. The Network Control Panel is organized as follows (see Figure 6-14):

- Individual **control widgets** (sometimes simply called **controls** or **widgets**) correspond to the input parameters of the modules in the network.
- Each module's controls are assembled onto a **page**. Each module has its own page, whose size depends on the number of input parameters and the control widgets attached to them.

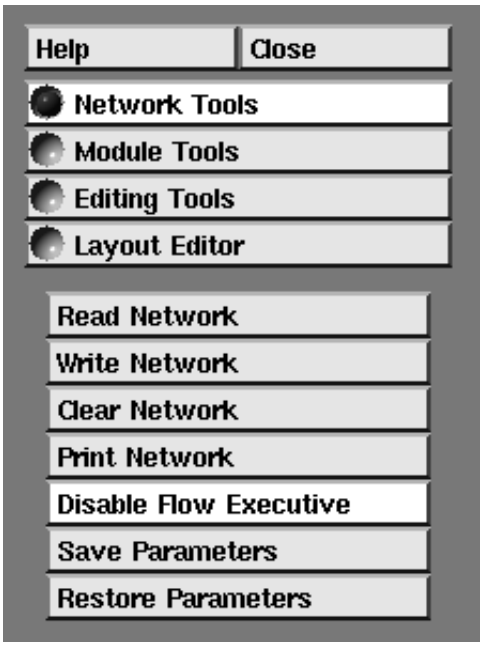


Figure 6-13 Disable Flow Executive Button

- All of the pages are gathered into the Network Control Panel window, which has the form of a **stack**: only one page at a time is visible; you can switch among the pages by clicking in the choice menu at the top of the window. (This menu is automatically created as you add pages to the stack.)
- You can change this default layout with the Layout Editor described later in the "Advanced Network Editor" chapter. If you save a network with a modified layout, the new layout is saved with it.

Cancelling an Operation

In many situations, you can "hammer" a running module. Use the left mouse button to drag the module to the Hammer in the lower right corner of the Workspace. This causes the module's process to exit.

For example, perhaps you were using AVS from a display that does not have hardware sphere rendering and you created a visualization network for a very large dataset that included sphere rendering, *and* you did not remember to use the Geometry Viewer's **Subdivision** control to reduce the number of polygons used to create each sphere. You might sit watching the **scatter dots** module executing for awhile until you began to realize that you will be sitting there for a very long time before you see any picture. Hammer the **scatter dots** module (its connections will be broken automatically), toggle **Disable Flow Executive**, drag down a new copy of the module (and perhaps a **down-size** module to reduce the size of the dataset) and hook it up, set **Subdivision** to 1 in the Geometry Viewer, then turn the Flow Executive back on.

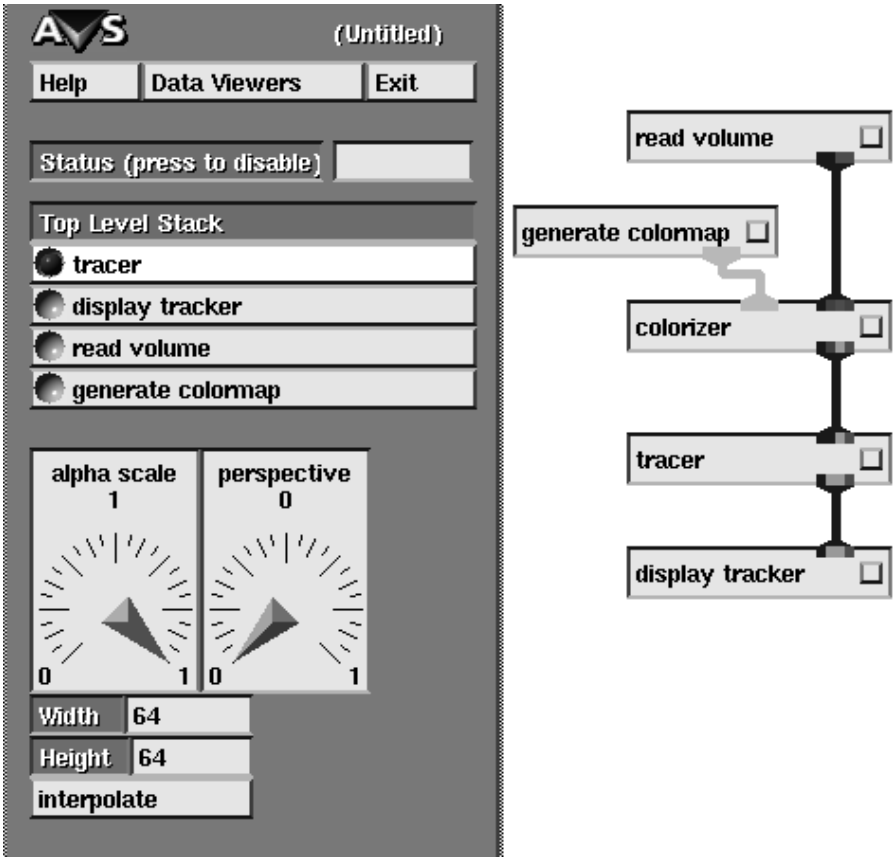


Figure 6-14 Organization of the Network Control Panel

If the module that you hammered was part of a multiple-module set executing as one process, the Network Editor will restart all of the unhammered modules in the process with their parameter settings intact.

You may need to manipulate one of the parameters of a module upstream from the hammered module in order to restart data flow through the network.

There is one restriction: you cannot hammer built-in modules. (These modules are part of the main AVS process.) For a list of built-in modules, see the list in the "Remote Module Execution" section of the "Advanced Network Editor" chapter.

Module Restart Option

If a module dies while executing its module icon will turn black. Modules can die for a variety of reasons, either because of a bug, because they got by AVS's data type checking and are trying to process incompatible data (usually occurs with incompatible dimensions or extents), or the module runs out of memory.

When a module dies, you get a message panel that informs you of the problem and gives you three options:

accept
This means that you accept that the module has died, and that you will deal with this by "hammering" the dead module (discard it by dragging it down to the Hammer icon in the Workspace). To get a new copy of the module, drag its icon down from the Palette and reconnect it.

restart
The module will be restarted with its default parameter settings.

restart same
The module will be restarted with the same parameter setting. The network will not re-execute until you have changed a parameter setting. This gives you a chance to correct a parameter value that may be killing the module.

There is also a **Restart Module** button on the **Module Tools** submenu. This restarts a module with its default parameter settings. It is a way to get a fresh instance of a module without hammering the old copy, then instancing a new version from the palette and reconnecting it to the network.

Note: Multiple modules can be in the same process. If one of these modules dies, all of the modules in the process also die, and turn black. **Restart Module** will restart all current dead modules in sequence to try to get the same modules running in the same process again.

Using Control Widgets

AVS has a variety of control widgets that allow you to specify module input parameters: integers, floating point numbers, text strings, filenames, mutually-exclusive choices, non-mutually-exclusive choices, and colormaps. The following sections describe how to use the various types of control widgets.

Module writers can give control widgets **conditional visibility**. This means that all of a module's widgets may not be visible at a given time, and only become visible if some condition is satisfied.

Using Type-In Controls

Figure 6-15 shows a typical type-in control widget.

To use a type-in, move the mouse cursor into the type-in area, so that it "lights up". Then, type any printable characters, ending with **Return**. The existing value, if any, is not replaced—you must explicitly erase it if you want to enter a completely new value. There are two erasure keys:

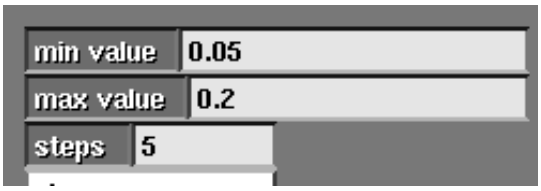


Figure 6-15 Type-In Control Widgets for Animated Float

Backspace
Erases the last character currently in the type-in area.

Ctrl-U
Completely erases the type-in area.

There are two ways to finish the entry. You can press **Return**, or you can simply move the mouse cursor out of the typein area. AVS checks the new value against the parameter's bounds and type. In some cases, the value you type is converted (e.g. decimal value converted to integer, out-of-bounds value converted to allowable maximum), and the result of the conversion is displayed.

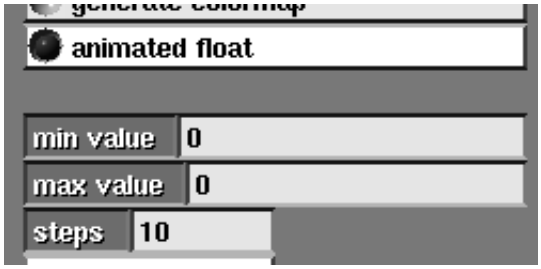


Figure 6-16 Default Values Replaced

Using Dial Controls

Figure 6-17 shows a set of typical dial control widgets.

You can use a dial either by *clicking* or by *dragging*:

- If you click with any mouse button at a location along the edge of the dial, the needle jumps immediately to that location and the current value indicator changes accordingly.
- Alternatively, click and hold down any mouse button near the needle; then use a circular motion to drag the needle either clockwise or counterclockwise. As you do so, the current value indicator changes. You can drag the needle any amount, from just a few degrees to many complete revolutions.

  If a dial's associated parameter has limits, attempting to drag the needle to a value outside the parameter's min-max bounds will fail—either the needle stops moving when you reach the limit, or it "snaps back" to its limit value.

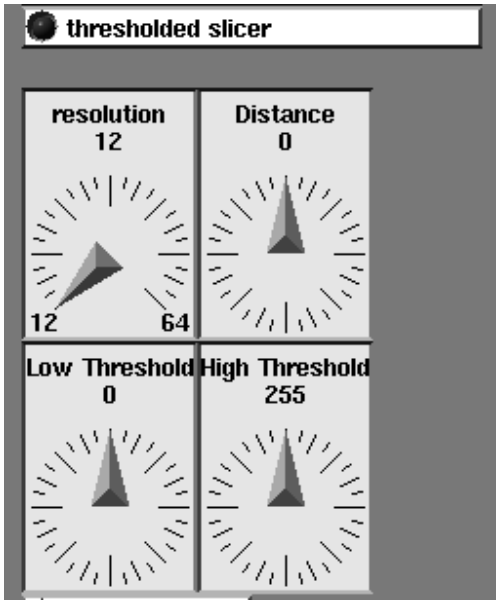


Figure 6-17 Dial Control Widgets

The Dial Editor

Dial control widgets are special in that they have an associated control window called the **Dial Editor** (Figure 6-18). To pop up the Dial Editor, move the cursor to the center of the dial, causing it to be highlighted. Then, click with any mouse button.

You can use this window to specify an exact value for the parameter (which may be easier than trying to move the dial needle by microscopic amounts). You can also change the dial's resolution (how much is once around) by typing in Minumum and Maximum values.

The **Immediate** button is a toggle switch. If you turn this feature on, the dial continuously sends values when you drag the dial needle to a new position. This causes the module that owns the widget to execute continuously, also. (This may not be advisable for compute-intensive networks—changes to the final output window may lag behind the movement of the needle.) In most modules, **Immediate** is off by default.

Some dial widgets are conditionally visible. They may not appear if other modules with overriding parameters are connected to the module.

Using dial widgets is quite intuitive. However, there are some subtleties to their use that become evident when certain data value ranges interact with some modules. These subtleties are explained below. You can probably skip over the remaining paragraphs in this section, and only refer to them if you find your dials behaving in a perplexing manner.

A module can declare a dial to be **bounded** or **unbounded**. It will declare it bounded when it is confident that parameter values will almost always fall

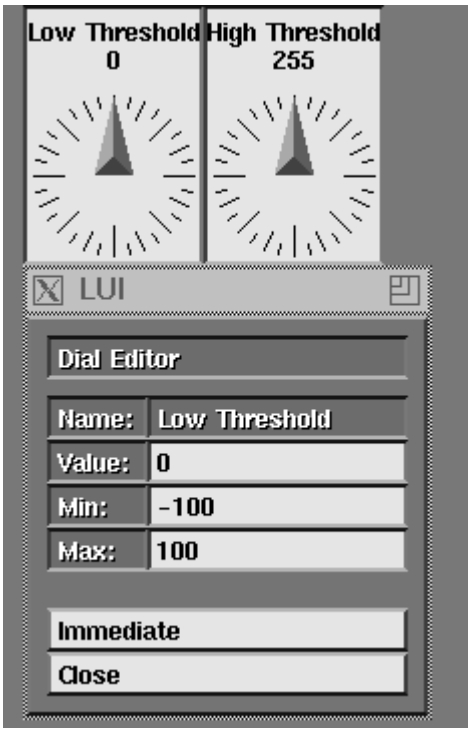


Figure 6-18 Dial Editor

within a certain range. It will declare it unbounded when it cannot predict a parameter range, or when it wishes the parameter range to be data-dependent (for example, based on the minimum and maximum data values in the input field.) You can check the module's man page to see how the module defines each of its dial parameters.

In the "Dials" figure, **resolution** is a bounded dial. The title is specified by the module. The module's declared minimum and maximum allowable values for the dial are shown at the lower left and lower right ends of the dial's scale. The current value for the dial appears at the top, beneath the title. The scale (or resolution) of the dial is max-min / 270 degrees. As you move the dial needle, the current value is updated correspondingly. You cannot move the needle past the minimum / maximum bounds. It will just stop. You cannot change the dial's minimum or maximum bound values. Bounded dials appear, for example, as the radius multiplier factor for **scatter dots** spheres, as the number of sample points for which to generate **streamlines**, or as the rotation of the **arbitrary slicer** plane.

Unbounded dials are also common (**distance**, and **high** and **low threshold** in the "Dials" figure.) Again, the title is specified by the module, and the dial's current value is displayed below the title. When an unbounded dial first appears, its indicator needle is always pointing straight up. The scale (or resolution) of the dial is always, initially, once around is 200. A module has no control over the resolution of unbounded dials; there is no "normalize dial resolution to data value range" function that the module can call.

In theory, you should be able to increment or decrement an unbounded dial's values indefinitely—just keep turning. However, a module may enforce data-dependent minimum or maximum bounds on unbounded dials. If you move the needle past the software-defined minimum or maximum, it will "snap back" to the bound value.

Unbounded dials appear, for example, as the **isosurface** module's level parameter, and as the **thresholded slicer**'s minimum and maximum threshold values.

It sometimes happens that the module defines, for example, the minimum field data value as the bounds for one dial, and the maximum field data value as bounds for another dial. If the input field data happens to be floating point values all falling between 0 and 1, and given that the default dial resolution is once around is 200, the dials become too sensitive to use. Anything you try to set just "snaps back."

In this case, you can use the Dial Editor to either type in specific values, or you can manually change the resolution of the dial from its default Minimum / Maximum of -100 / 100 to 0 and 1.

It also helps to hook up **statistics**, **print field**, and / or **generate histogram/ graph viewer** module(s) to your network. **statistics** tells you field minimum and maximum data values; **print field** reports field minimum and maximum extents; and **generate histogram/graph viewer** plots the distribution of data in a field. All help you know the characteristics of your data and can illuminate dial behavior.

Using Slider Controls

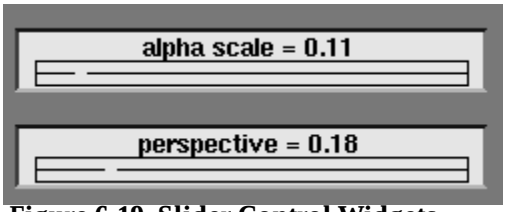


Figure 6-19 Slider Control Widgets

Figure 6-19 shows a typical slider control widget.

As with a dial, you can use a slider either by *clicking* or by *dragging*:

- If you click with any mouse button at a location along the slider, the crosshair jumps immediately to that location and the current value indicator changes accordingly.
- Alternatively, click and hold down any mouse button near the crosshair; then drag it to the left or right. As you do so, the current value indicator changes.

The parameter's minimum and maximum values are displayed only while you are adjusting the slider. Similarly, the slider's name disappears while you are adjusting it.

Using a Set of Choices (Radio Buttons)

Figure 6-20 shows a typical *choice* ("radio buttons") control widget, which allows you to select from a mutually-exclusive set of choices.

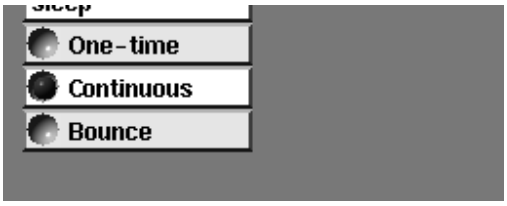


Figure 6-20 Set of Radio Buttons

A red ball and highlighting indicates which one of the choices is currently selected. To select another choice, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button to move the red ball to the new selection.

Using Toggle Controls

Figure 6-21 shows a toggle control widget, in the *off* state.

Figure 6-21 Toggle Control Widget Off

To change the state of a toggle switch, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button.

Figure 6-22 Toggle Widget On

Using Tristate Controls

Tristate control widgets are used for parameters that can assume three values, not just two.

Figure 6-23 Tristate Widget

To change the state, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button. Successive clicks cycle the widget through its three states.

Using Oneshot Controls

Figure 6-24 shows a typical oneshot control widget. This type of control is used to invoke a command, rather than to change the state of a parameter.

Figure 6-24 Oneshot Control Widget

To use a oneshot control, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button to make the box flash.

Using File Browser Controls

Figure 6-25 shows a typical file browser control widget.

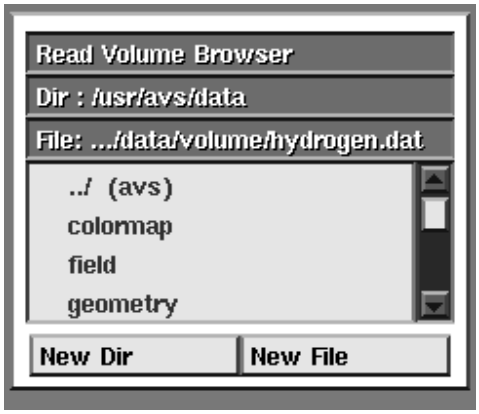


Figure 6-25 File Browser Control Widget

The entries in a file browser are color-coded: black entries are files; red entries are subdirectories (the topmost red entry is usually the parent directory). To select one of the entries, click on it with any mouse button. Selecting a directory entry changes the working directory, causing filenames in that directory to be displayed, along with the names of any subdirectories.

Since a directory might contain a large number of entries, a file browser has a scroll bar along its right edge. Clicking inside the scroll bar makes additional entries appear:

- The left mouse button scrolls upward.
- The effect of the middle button depends on exactly where the cursor is:
  - **In the arrow box at the top**. Click to scroll the list to the very top.

  - **In the elevator shaft**. Click and hold down the button to grab the elevator bar. Moving the bar up or down causes the list to scroll accordingly.
  - **In the arrow box at the bottom**. Click to scroll the list to the very bottom.
- The right mouse button scrolls downward.

A file browser has these buttons at the bottom:

New Dir
Pops up a dialog box in which you can type the name of another directory (full pathname or path relative to the current directory). Be sure the mouse cursor is within the dialog box (but not on the **OK** or **Cancel** button) before you start typing the directory name. When you click the **OK** button in the dialog box (or press the **Return** key), the directory whose name you've typed becomes current, and its filenames are displayed in the browser window.

Should you inadvertently give a filename, it will select that filename as though you had used **New File** *and* select the directory it is in.

Use **Backspace** to erase the last character or **Ctrl-U** to erase the entire name. If you change your mind altogether, click the **Cancel** button.

New File
Pops up a dialog box that works the same way as the **New Dir** box. This allows you to specify the file to be processed, either with a full pathname or a name relative to the current directory. Should you give a directory rather than a filename, it will change to the direcotry.

Close
(not always present) Some file browsers are "sticky"—they pop up in a separate window and remain onscreen until you explicitly remove them by clicking this button.

Other Browsers

These browsers have scrollbars that behave just like the file browser scrollbars. They serve slightly different purposes.

Choice Browser
Choice browsers look just like file browsers, except that the items displayed are not files. The Remote Host Browser described below, for example, allows you to select an alternate host to run modules on. The Help Demo browser is another choice browser.

Text Browsers
Text browsers produce scrolling windows of text. They are read-only; you do not select items. The **Show Module Documentation** window is a text browser.

Using the Colormap Control

Figure 6-26 shows the control widget that generates a colormap. You can also use it to maintain a set of colormaps in disk files.

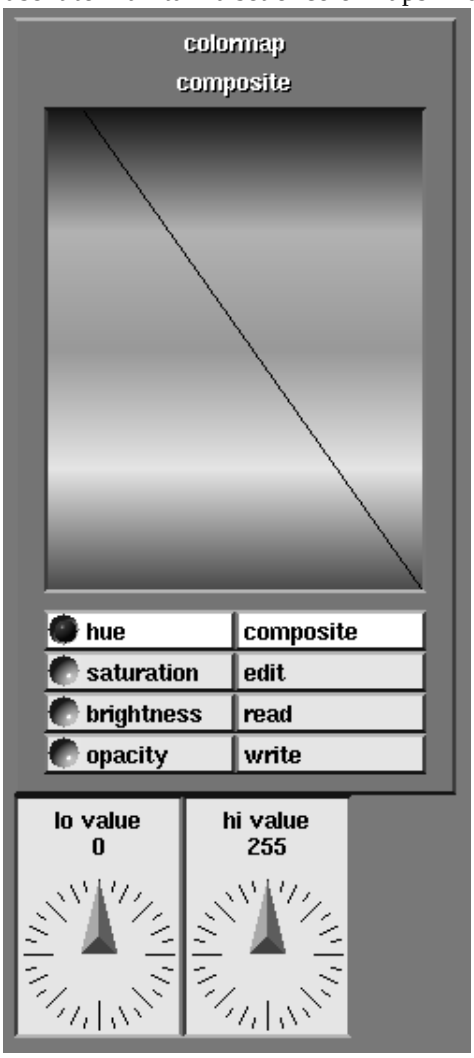


Figure 6-26 Colormap Editor Widget

Translating numbers into colors is an important technique in scientific visualization. The **generate colormap** module's **Colormap Editor** widget is the palette you use to mix the colors. An AVS colormap is used by a number of modules to translate numbers into to pixel values (i.e. colors). A colormap is essentially a 256-line table, each line of which includes four fields: hue, saturation, brightness, and an auxiliary field. The Colormap Editor widget allows you to create a colormap table visually.

The widget has four "pages", one each for hue, saturation, value, and opacity (the auxiliary field). You switch among the pages by clicking the radio buttons in the bottom left part of the control widget.

Each page has the appearance of an area graph. It is actually a set of 256 very thin horizontal bars. On the hue page, the length of the top bar specifies the hue number for line 0 of the table. The color of this bar indicates the hue to which a data value of 0 will be mapped. The next bar specifies both the hue number for line 1 and the hue to which data values of 1 will be mapped; and so on.

Initially, the hue numbers form a linear ramp. Smaller numbers will be mapped into the blue part of the spectrum; larger numbers will be mapped into the red part. To change the set of hue numbers:

- Place the cursor near the top (but within) the square containing the 256 horizontal bars.
- Press any mouse button and drag the cursor downward along the new path. The lengths and colors of the horizontal bars change as you drag downward. The new values are reported at the top of the control widget as you drag.

The Colormap Editor widget's additional controls are described below.

composite

The hue, saturation, and brightness "pages" of the Colormap Editor normally show a graphical representation of *just* the hue, or *just* the saturation, or *just* the brightness. Switching on **composite** displays the entire colormap, with colors that accurately reflect the shade that the combined hue, saturation, and brightness values will produce in the rendered image.

For example, with hue toggled, the default colormap page shows lower data values mapping to a medium clear blue. If you then switch to saturation, you could edit the saturation of this blue to be 50%, by drawing a line down the middle of the editor page, producing a pale clear blue. But when you switch back to the hue page, the color still shows as medium blue.

If you toggle **composite**, the true combined hue, saturation, and brightness shade will appear—the actual pale clear blue.

Scrawled across the actual colormap will be a black line. This black line shows the setting for the hue for that data value (if **hue** is toggled), or for saturation (if **saturation** is toggled), or for the brightness (if **brightness** is toggled).

The composite colormap does not display opacity values; there being nothing "behind" the colormap to obscure or reveal.

edit

The **edit** button raises another control panel (Figure 6-27) with additional colormap editing features. Working from the top down:

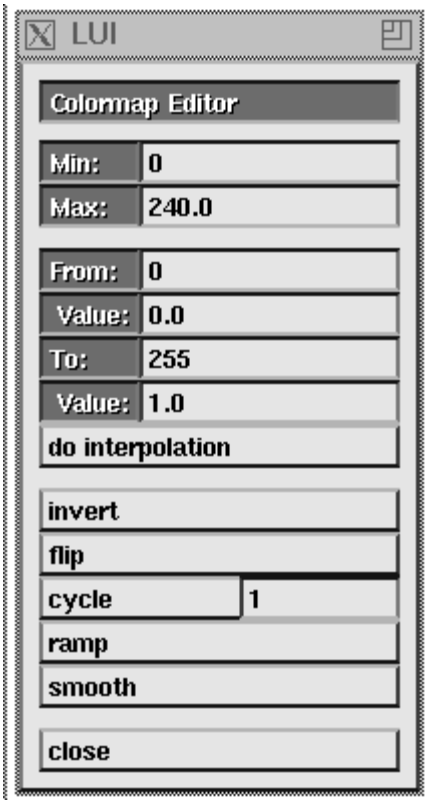


Figure 6-27 Edit Panel

Min
Max

The colormap editor uses the HSV color model. Hue is represented as a circle. The default AVS colormap produces hues from 0 to 240 degrees around this 360 degree circle; i.e., from red to blue. One third of the hue spectrum (the magentas and blue-reds from 240 degrees to 359 degrees) is normally excluded from the colormaps. The **Min/Max** typein controls let you set the beginning and endpoints of the spectrum of hues in the colormap. A **Min/Max** of 100 to 140 would produce 256 fine distinctions in greens. **Min**/**Max** applies only to hues.

From/Value
To/Value

These typein controls give you precise numeric control over which colormap slots map to which explicit hues. Rather than drawing curves in the editor window freehand with a mouse, you type beginning and ending slots in the colormap, and the range of hues, saturation, or brightness that will be associated with them.

To make the typeins take effect, press **do interpolation**.

For example: you have a set of data values ranging from 0 to 200. The values from 160 to 165 are important and you want them to show up in a contrasting color—not the evenly blended shade they would have with the neighboring values. 160 would be the **From** value; 165 would be the

To value. Pick any contrasting hue, saturation, or opacity—say bright red (0.0), and enter it as both the from **Value** and the to **Value**, then press **do interpolate**.

When you have byte data that ranges from 0 to 256, the mapping between the data values and the colormap slots is easy to figure out. You can use the **field legend** module, which takes both a colormap and a field as input, to see how the numeric values are mapping onto the colormap.

invert

Reverses the mapping of the range 0..255 to color values. The 0th color becomes the 255th color, the 1st color becomes the 254th color, etc. Visually, this flips the colormap over a horizontal axis.

flip

flip is like invert; it inverts the colormap for hue, saturation, brightness, or opacity, whichever is selected. But, where **invert** inverts the colormap about the horizontal axis, **flip** flips it about the vertical axis. New value = (Max Colormap Value - Old Value).

cycle

cycle performs a "circular shift" on the colormap. How much the hues, saturation, or brightness moves is set by the typein. The default is 1. Repeatedly pressing cycle repeatedly shifts the colormap.

For example, **cycle** can be used as a sort of pseudo contour generator for volume data. Create a colormap with an opacity like a narrow square wave - most values wholly transparent, with a narrow band wholly opaque (you could use the **From-To** typeins). Setting a cycle step of 10, then repeatedly pressing cycle would show the data as a series of pictures 0-10 data, 10-20 data, 20-30 data, etc.

ramp

ramp sets the hue, saturation, brightness, or opacity values in a colormap to be an even graded interpolation between its minimum and maximum values. This is the default for hue and opacity.

smooth

smooth "smooths out" the distribution of values in the hue, saturation, brightness, or opacity of a colormap. Where the black hairline shows spikey sudden changes, **smooth** produces curves. It performs a Gaussian convolution on the colormap values.

Lo Value/High Value

These dials let you set the minimum and maximum values of your data. These values are included in the **generate colormap** module's output colormap. Modules will use these values to normalize the range of the colormap to the range of the data in the field. You can use these dials when your data's range of values is either not evenly distributed between 0 and 255 (for example, your data is floating-point values from -1 to 1), or much of the data's values lie outside the 0 to 255 range (for example, from 0 to 10,000). You can also

use it if you want to "fan out" the colors representing a narrow range of data. (To see what your data's min / max values are, use the **statistics** module.)

Note: The **color range** module will calculate the minimum and maximum data values in a field and store them in the output colormap. These dials are the manual way of doing the same thing. They are also more flexible. For an explanation of why this is a necessary option, see the **color range** module man page in the *AVS Module Reference Manual*.

Set the **Lo** and **High** values either with the floating-point dials, or by clicking the mouse on the dial's center to bring up a floating-point typein.

Read/Write

These functions allow you to maintain a set of colormaps on disk. Clicking either of these buttons brings up a file browser that allows you to specify a file in which to store the current colormap (**Write**), or from which to reinstate a previously-stored colormap (**Read**).

Organizing a Network's Display Windows

In general, an AVS network produces one or more pictures as its output. (In this section, we use the word "picture" to refer either to an *image*, produced by converting data directly into pixels, or to a *pixmap*, produced by converting data to a geometry which is then rendered.) Each picture is displayed in its own *display window (output window)*, although some pictures may combine data from several data sets. This section describes the way in which AVS creates display windows, and the ways in which you can manipulate these windows.

Whenever you drag a module icon from the Palette to the Workspace, AVS adds the corresponding page of control widgets to the Network Control Panel window. For modules whose output is an on-screen picture, AVS also creates a display window. Initially, this window is empty. When you complete a network and specify all the required input data, a picture appears in this window.

Complex networks may include several modules that produce pictures as output. AVS creates a separate window for each such module.

Picture Size and Window Size

When a picture first appears in a display window, AVS automatically resizes the window to fit the picture. (Since the size of the picture depends on the data being visualized, AVS cannot calculate the appropriate window size before data flows through the network.) If the window size subsequently changes, AVS automatically resizes the picture, if appropriate. In this connection, it is important to keep in mind the difference between the output windows produced by the **display image** and **display pixmap** modules. (The **image view-**

er and **graph viewer** modules have their own behavior, described in their respective chapters.)

Images

An *image* is originally defined in terms of pixels. The only scaling AVS performs on **display image** windows is successive doubling: *x2*, *x4*, *x8*, etc. When you resize an image window, there are several possibilities:

- If you make the window exactly two times as large (or four times, or eight times, etc.), the image is scaled and continues to fill the window exactly.
- If you make the window any other size, the window will no longer fit the image exactly. Only part of the image will be visible; to see more of it, use any mouse button to click-and-drag the image. Alternatively, use the scroll bars that appear along the window edges. They work the same way here as in a File Browser:
  - The left mouse button scrolls upward.
  - The effect of the middle button depends on exactly where the cursor is:
    - **In the arrow box at the top (or left)**. Click to scroll to the very top or left of the image.
    - **In the elevator shaft**. Click and hold down the button to grab the elevator bar. Moving the bar causes the image to scroll accordingly.
    - **In the arrow box at the bottom or right**. Click to scroll to the very bottom or right of the image.
  - The right mouse button scrolls downward.

Pixmaps

In most cases, a *pixmap* is generated by the **render geometry** module. AVS can scale pixmaps continuously. When you resize a pixmap window, the picture always resizes accordingly by re-rendering at the new resolution.

Using the Window Manager

All display windows are initially created as "top-level" X windows. This means that you can manipulate them using the X Window System's window manager program—move, iconify, resize, raise, lower, etc.

If you use the **Edit layout** function of the Network Editor to reorganize a network's control widgets, you may want to include the network's display windows in the reorganization. This topic is discussed in section "Including Display Windows in a Reorganized Layout" below.

Note: Don't use the *xkill*(1) program or any other means to delete an AVS display window or any other AVS window. This will cause the network to hang.

Using a Display Window's Pulldown Menu

Each display window has a pulldown menu that allows you to resize the window's picture without having to use the X Window System window manager. To use the menu, click and hold down any mouse button on the small square at the left side of the windows title bar. Drag the mouse to highlight the desired menu choice, then release the button.

The menu choices vary from module to module. **display pixmap** has only the first two options, **display image** has all of the following menu choices:

Zoom Full Screen
Enlarges the display window to be (approximately) the largest possible square size. An image is scaled up accordingly; a pixmap is re-rendered.

Unzoom
Restores a zoomed window to its former size. If you use the window manager to move and/or resize a zoomed window, AVS continues to remember the previous configuration as the unzoomed position and size. If the window had been moved inside a *page* or *stack* using the Layout Editor (described later), it returns to that location.

Auto-Fit (Turn On/Off)
This toggle switch controls the automatic sizing of display windows to exactly fit the current image size. By default, this feature is enabled.

Scrollbars (Turn On/Off)
When an image is larger than its display window, AVS automatically adds scrollbars unless you turn this toggle switch off. You can configure AVS not to use scrollbars by default: use the **ImageScrollbars** parameter in the AVS startup file (see Chapter 2).

Resize Window to Fit Image
Use this choice when **Auto-Fit** is turned off and an image is too big for its window. (If you don't use this function, you can scroll the image in order to see different parts of it.) This choice is also useful when AVS has refused to scale up an image (as described above), and you want to trim off the unused portion of the display window.

x1, x2, x4, etc.
Scales the image by a power of 2. The "(selected)" annotation indicates the scaling currently in use.

Using the Network Editor Menu System

The Network Editor has a two-level menu of functions that support your work in creating, revising, and executing networks. The top-level menu is always visible in the upper-left part of the Network Construction window (Fig-

ure 6-28). At any moment, one of the menu choices is selected, and the corresponding submenu appears below the main menu.

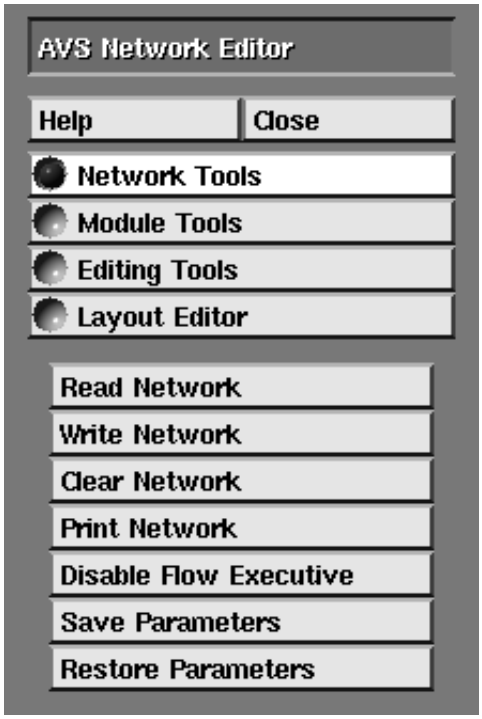


Figure 6-28 Network Editor Main Menu

The following sections describe the functions in the Network Editor submenus. Whatever submenu is currently active, the following two buttons always appear near the upper-left corner of the Network Construction window:

Help

Pops up a help browser window and displays the contents of the *network_editor.txt* help file. This file provides an overview of Network Editor usage. The browser shows the additional help topics that relate to the Network Editor.

Close

Closes the Network Construction Window, but does *not* delete the current network, if any. In fact, if a network is currently executing, it will continue to do so even though the Network Editor windows are closed.

A **Display Network Editor** button automatically appears at the top of the Network Control Panel window, providing a way to reopen the Network Construction Window at a later time.

Network Tools

Read Network

Reads an existing network from disk storage into the workspace. A file browser widget appears to help you specify the file containing the network definition.

If there is already a network (or even just a single module) in the Workspace, you must choose whether to **Clear** the existing network or to **Merge** the new one with the existing one. Merging can cause the module icons to overlap in the Workspace—use the left mouse button to rearrange them afterward.

Write Network

Writes the current network to disk storage. If you have already specified a filename for the current network with **Read Network** or **Write Network**, AVS offers to use the same name. If you choose not to, a file browser appears to help you specify the filename.

Note that performing a **Read Network** with the **Merge** does not change the name of the current network. You must select the **Clear** option to effect the name change or choose a new name when writing the network.

Write Network saves networks in Command Language Interpreter (CLI) instructions (See the "Command Language Interpreter" chapter in the *AVS Developer's Guide*.) It saves the state of the network to the extent that its CLI instructions allow. In general, **Write Network** saves:

- All the modules that have been placed in the workspace, and all of the connections between them. (CLI **net_show** command.)
- The current state of all module parameter settings that have been changed from their default values. This includes the names of files that have been read into the network. (CLI **parm_save** command.)

If you wish to save all parameters, not just those which have changed, you must use the **NetWriteAllParms** keyword in your *.avsrc* file.

- The current state of the layout of the screen, including all changes made to widgets using the **Layout Editor** and the **Parameter Editor** (such as changing dials to sliders), and the current location of all display windows. (CLI **layout** command.)

Write Network *does not* save the current state of any of the viewers (Geometry Viewer, Image Viewer, Graph Viewer). Rather, it gives the viewers a chance to save their own state. Each does so in different measure.

The Geometry Viewer saves its state using the Geometry Viewer CLI commands (see the "Command Language Interpreter" chapter in the *AVS Developer's Guide*). All camera, light, object, and transformation state is saved in this way.

To avoid saving large quantities of geometric data, the Geometry Viewer saves only the attributes of the objects saved in the scene. If there exists an up-to-date description of the geometry of any particular object already in a "geom" file, the Geometry Viewer will try to read the geometry data

from the file when reading in the network. The Geometry Viewer knows that geometry is up-to-date if either it was read in directly from the Geometry Viewer **Read Object** menu or the object's geometry was saved since the last time it was modified with either the **Save Scene** or **Save Object** Geometry Viewer commands.

It is often the case that modules will immediately regenerate the geometry for all of the objects that they've produced in a particular network.

When saving references to AVS executables, **Write Network** always prefaces the binary filename with $PATH. $PATH usually resolves to */usr/avs*. When saving references to data files or network files, **Write Network** by default saves absolute path names to the files. It is possible to get **Write Network** to substitute the $DataDirectory and $NetDir strings (if you have defined them) if you use **NetWriteAbsPath off** in your *.avsrc* file. Pathways to remote modules and remote data directories are always saved as $RemMods and $RemData, no matter how **NetWriteAbsPath** is set.

The network files are editable ASCII CLI files.

Clear Network

Deletes the current network and associated control panels. AVS displays a dialog box to have you confirm the selection.

Print Network

Creates a PostScript file named */tmp/AVSnetPID.ps* (where *PID* is the process number), which shows the layout of the current network. AVS composes a shell command to print the file, then displays a pop-up window showing this command. You must choose whether or not to issue the print command. (If you choose not to print, you may want to copy the PostScript file to another location, using an **xterm** window. The next time you select **Print Network**, the PostScript file will be overwritten.

The default command is *lpr*. You can set this to be something else using the **PrintNetwork** *.avsrc* keyword. (See the "Starting AVS" chapter.)

Disable Flow Executive (toggle)

Modules perform their computations under the control of the Flow Executive, which determines when their output is required by another module and reexecutes them if their most recent computation has become out of date. Disabling the Flow Executive inhibits all network execution. This is useful when you wish to change the values of several parameters, but you don't wish to have the network's modules recompute after each change.

Save Parameters

Saves the current module parameter values for the current network in a parameters file, named */tmp/avs_snapN.PID*. (*PID* is the process number, and *N* an internal sequence number. This is useful if you want to provide yourself a checkpoint, to which you can return later in the same Network Editor session.

Restore Parameters

Resets the network's parameter values to those most recently saved. If appropriate (and if the Flow Executive is not disabled), the network recomputes. You cannot retrieve the parameters of another network, or of the same network from a previous Network Editor session.

Module Tools

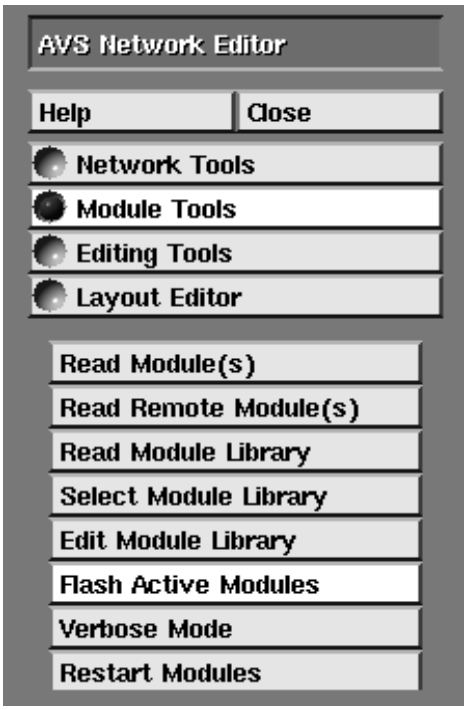


Figure 6-29 Module Tools Menu

Read Module(s)

Adds a module to one of the categories in the Palette. A file browser widget appears to help you specify the module program file. Each module specifies its category; you cannot choose a particular category when invoking this function.

It is possible for a single program file to define several modules. In this case, all the modules defined in the file are added to the Palette. You can also specify a directory, in which case the Network Editor loads *all* the modules defined in module program files within that directory.

Read Remote Module(s)

This brings up a Remote Host Browser that lets you select **another system** in your network from which to read and execute modules. The remote host might be a very powerful number cruncher, and the remote module a simulation that executes on this heterogeneous host sending its data to AVS on your workstation for visualization. This is a large topic covered in

its own "Remote Module Execution" section in the "Advanced Network Editor" chapter.

Read Module Library

Loads the Module Palette with all the modules in a specified *module library*, replacing the existing library in the Palette. A file browser widget appears to help you specify the library file to be read. After the library is loaded, the title bar above the Palette changes to display the name of the new library. The existing module library is still accessible through the **Select Module Library** browser.

A library file names some combination of AVS-supplied modules, user-written modules, and directories that contain modules. For example:

```
builtin         geometry viewer
builtin         read geometry
builtin         display pixmap
file            /usr/johnp/avs_modules/smooth
file            /usr/johnp/avs_modules/rough
directory       /usr/johnp/avs_modules/tools_dir
```

For details on creating module libraries see the "Constructing a Module Library" section in the "Advanced Network Editor" chapter.

Select Module Library

This panel duplicates the function of the module library icons across the top of the module Palette. It invokes a choice browser, allowing you to select one module library from all the ones that have already been selected with **Read Module Library** during the current Network Editor session. The default library—the one automatically loaded at the beginning of the session—is listed, too.

Edit Module Library

A facility for interactively editing and creating module libraries. This is discussed under "Constructing a Module Library" in the "Advanced Network Editor" chapter.

Flash Active Modules (toggle)

If this toggle switch is *on*, module icons are highlighted (displayed with a black background) as the modules execute. Turning this off may speed up the execution of highly interactive networks. It is on by default.

Verbose Mode (toggle)

If this toggle switch is *on*, AVS displays debugging information as the modules execute. The information is sent to the *stderr* of the **avs** command that started the AVS session. Typically, the information is displayed in the terminal emulator window from which you typed the **avs** command.

Restart Modules

Brings up a panel that gives options for restarting modules that have died. Selecting **Restart** restarts all "blacked out" modules with their de-

fault parameter settings. **Restart** also will throw away an instance of a module and instance a fresh copy without requiring you to go through the usual user interface steps to accomplish this. **Restart Same** restarts all blacked-out modules with the same parameter settings. (This may well kill the module(s) again.)

Editing Tools

The Editing Tools menu controls two Network Editor operations: network editing operations such as copying, cutting, and pasting groups of modules; and **macro** module editing operations. The Editing Tools menu and network editing operations are described in the "Advanced Network Editor" chapter.

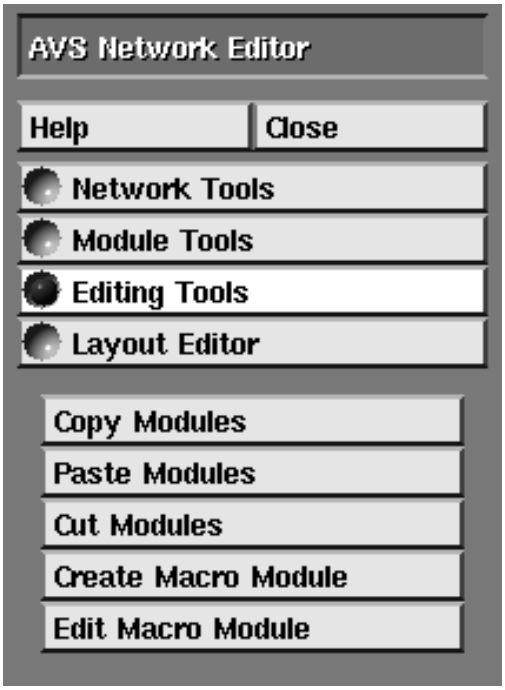
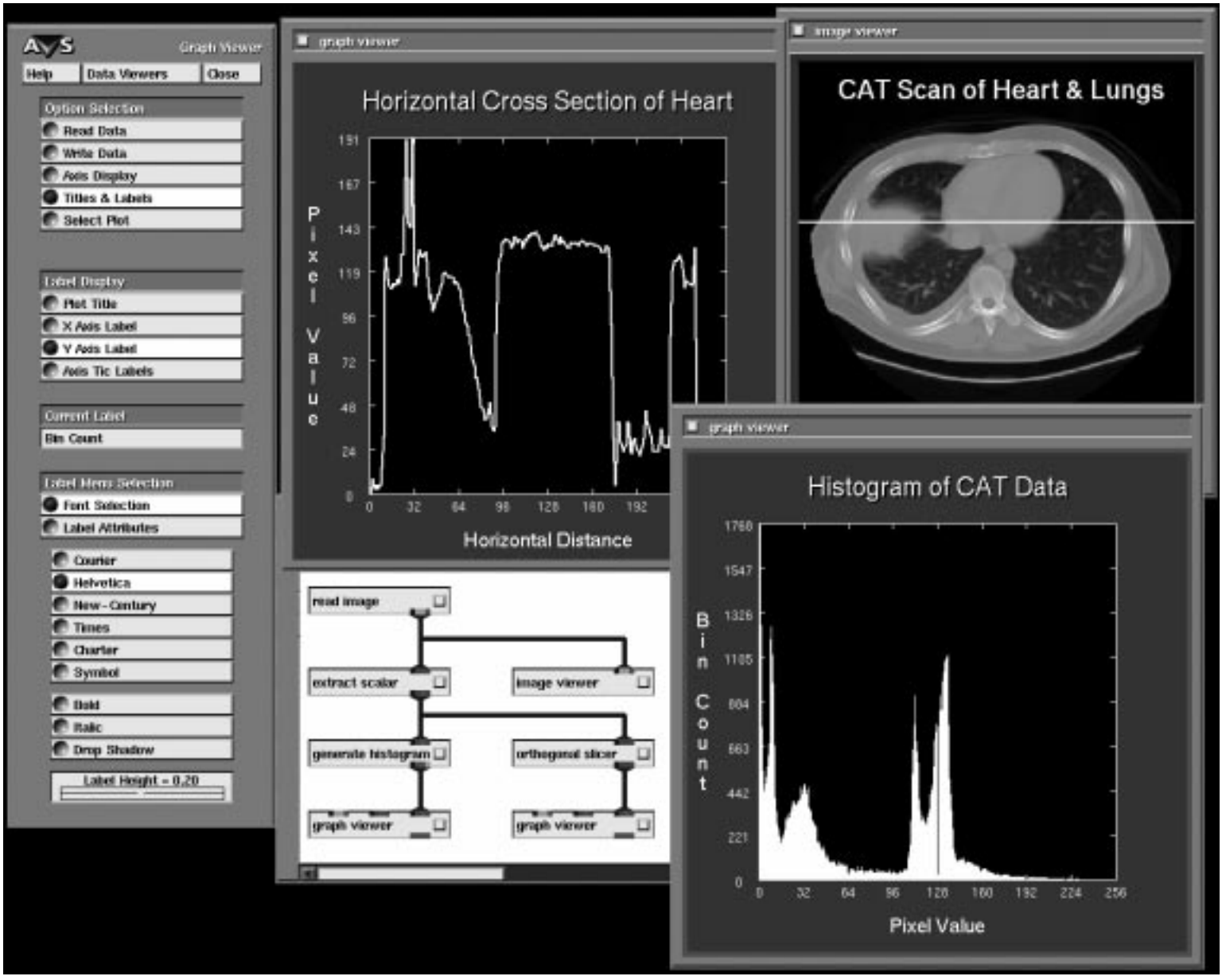


Figure 6-30 Editing Tools Menu

Layout Editor

The AVS **Layout Editor** has been called one of the best, but least-known AVS features. Selecting **Layout Editor** places the Network Editor in a mode that allows you to "redesign" the user interface of an AVS network. You can, for example, place widgets outside the Network Editor control panel to use screen space more effectively. The Layout Editor is described in the "Advanced Network Editor" chapter.

Figure 6-31 Layout Editor Menu


AVS
Graph Viewer
Help
Data Viewers
Close
Option Selection
Read Data
Write Data
Axis Display
Titles & Labels
Select Plot
Label Display
Plot Title
X Axis Label
Y Axis Label
Axis Tic Labels
Current Label
Bin Count
Label Item Selection
Font Selection
Label Attributes
Courier
Helvetica
New-Century
Times
Charter
Symbol
Bold
Italic
Drop Shadow
Label Height = 0.20
graph viewer
Horizontal Cross Section of Heart
Pixel Value
Horizontal Distance
image viewer
CAT Scan of Heart & Lungs
graph viewer
Histogram of CAT Data
Bin Count
Pixel Value
read image
extract scalar
image viewer
generate histogram
orthogonal slicer
graph viewer
graph viewer

CHAPTER 7

GRAPH VIEWER SUBSYSTEM

Introduction

The AVS Graph Viewer subsystem is an interactive tool for creating XY linear or contour plots of data.

XY linear plots can be displayed as a line plot, area plot, scatter plot, or bar plot. Contour plots are displayed as 2D graphs of lines connecting data points with the same **level** (numeric) value. The graphs can be annotated with a title, legends, and axes labels in various font styles and colors. You also have control over the placement and style of axis tic marks.

The Graph Viewer exists in two forms: it exists as the Graph Viewer subsystem accessible from the main AVS menu, and it exists as the **graph viewer** module in the Network Editor's module Palette.

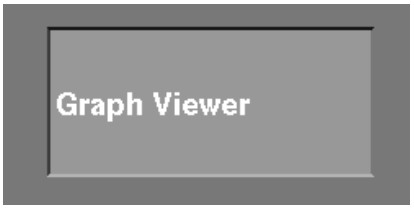


Figure 7-1 AVS Graph Viewer Button on Main AVS Menu

Figure 7-2 Graph Viewer Module in the Network Editor

You can view graphs of multiple datasets within the same window, the same dataset in different windows (such as a line plot showing trends and a scatter plot showing groupings), or different datasets in separate windows.

The Graph Viewer accepts one or two-dimensional data as input. The input data can be either an ASCII file in a particular format, or an AVS field. In addition, the Graph Viewer will read in an AVS *.x* image file and use it as a back-

drop for its plots; and it can re-generate plots saved from an earlier Graph Viewer session in an AVS Plot file.

Data can enter the Graph Viewer in two ways: ASCII data files, AVS fields (and the non-data AVS Plot and *.x* AVS image files) can be read using the Graph Viewer's **Read Data** function. AVS fields (and non-data *.x* AVS image files) can flow into the **graph viewer** module from an AVS network.

The Graph Viewer will interpret the input data it receives as either a series of Y values to be plotted against a set of constantly-spaced X axis intervals that the Graph Viewer generates automatically (**Plot as Y Data**); as a series of XY values to be plotted against Y and X axes that are automatically generated to match the range of the input data (**Plot as XY Data**); or it will interpret the input data as a 2D array through which it will draw contour lines connecting the same level values (**Plot as Contour Data**).

You can save plots as ASCII data, AVS Plot-format, AVS Geometry, and PostScript files. The **graph viewer** module will output an AVS geometry or an AVS image. The AVS image can be converted into a PostScript file with the **image to postscript** module.

The Graph Viewer is used for viewing data; it cannot be used to alter data.

There are sample ASCII plot data files in */usr/avs/data/graph*. The AVS **Demo** suite, accessible from the main AVS **Applications** menu, illustrates Graph Viewer usage.

Entering the Graph Viewer

The Graph Viewer subsystem can be started using one of the following methods:

From the shell directly

The following command line starts the Graph Viewer automatically when you begin the AVS program:

```
avs -graph
```

To exit the Graph Viewer, press the **Exit** button at the top of the Graph Viewer control panel. See the "Starting AVS" chapter for additional command-line options that affect how the Graph Viewer is started.

From the main menu

You can start the Graph Viewer at the AVS main menu by moving the mouse pointer to the Graph Viewer menu item and clicking any mouse button.

Data Viewers

At the top of each subsystem control panel is a **Data Viewers** button. Pressing this button creates a pop-up menu listing the AVS subsystems.

While still holding the mouse button down, roll the mouse cursor down to **Graph Viewer** and release. This raises the Graph Viewer control panel. You may wish to use your window manager to move the Graph Viewer control panel to another part of the screen so that other subsystem control panels are also visible.

In a network

You can include the **graph viewer** module in an AVS network.

Graph Viewer—Basic Interface

The Option Selection menu (Figure 7-3) at the top of the Graph Viewer control panel provides access to the basic operations for creating a plot.

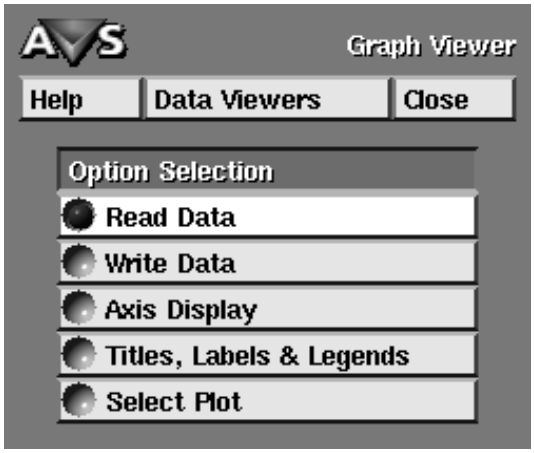


Figure 7-3 Graph Viewer Main Menu

The following top-level menu choices are always visible in the Option Selection menu area:

- **Read Data**—select the input file type (**File Type** menu); select how to interpret the input data (**Data Format** menu); select whether the new plot will replace the existing plot, be added to the plot, or be plotted in an entirely new window (**Plot Control** menu); select the type of linear plot to make, or the number and level of contours; and **Delete Plot Window**s.
- **Write Data**—write the current plot window as one of the four output file types.
- **Axis Display**—control the border accents; the number, type, and placement of axis tick marks; the axis range, and the axis type (e.g., logarithmic or linear).
- **Titles, Labels &Legends**—create titles for each plot window, labels for the individual X and Y axes, and legends for the plot lines.
- **Select Plot**—control the type of plot (line, scatter, etc.), as well as the appearance of individual lines within the plot (dotted, dashed, thickness, color).

One of these choices is always selected. The area below the Option Selection menu changes depending on which choice is currently selected.

Using the Graph Viewer

Using the Graph Viewer consists of four steps:

1. Input the data you want to view. You need to select the input file type (ASCII, AVS field, etc.) and the data format (**Plot as Y Data**, **Plot as Contour Data**, etc.) you want to use.
2. If you are creating a linear plot, choose the type of plot (line, area, scatter, bar). If you are creating a contour plot, set the number, value, and range of the contour levels.
3. If you wish, edit the plot axes, titles, labels, and style of the plot lines.
4. If you want to save the output, you then select the file type you want to save or print.

Multiple Plot Windows—The Current Window

You can create and delete plot windows at any time. This provides the ability to display the same dataset in different windows and work with each one independently, or to display completely separate datasets in different windows for comparison.

For example, Figure 7-4 shows three plot windows. The first window has a contour plot of one 2D slice through the center of a 3D field dataset. The second window is a bar plot of a 1D section through the middle of the same 2D slice. The third is a line plot of another 1D section through a different part of the same 2D slice.

Each window contains one or more plots and can be labeled with its own title, X and Y axis labels, and tick marks. These identifiers can be altered in size, style, color, and position at any time.

Every plot window has complete X Window System capabilities.

One of these plot windows is the **current** plot window that will be affected by all menu operations. **The current plot window is indicated by a red border**. You can change the current plot window by moving the mouse cursor into any plot window, then pressing *any* mouse button.

Read Data

The Graph Viewer accepts an ASCII file or an AVS field as input and uses the numerical data to generate a plot. It also accepts two other input file types

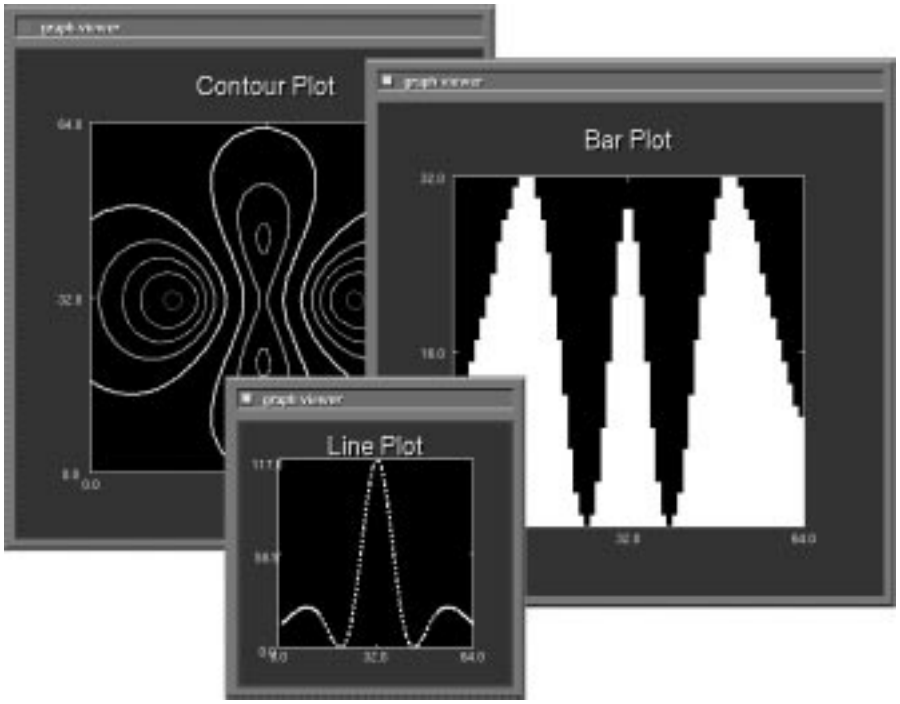


Figure 7-4 Multiple Plot Windows

that are not plotted as data—AVS Plot files and AVS image files. The menu of choices appears under the **Read Data** main menu button (Figure 7-5).

File Type Submenu

The Graph Viewer accepts the following *data* file types as input:

Read ASCII File

These are ASCII files with numeric data arranged in one or more columns. The data can be specified as integers, real numbers, or in floating point scientific notation style. You can read in any file by typing its name into the **Read Data**'s **New File** type in panel. However, for a file to show up in **Read Data's** file browser window, it must have the file suffix *.dat*.

The file can contain only numerical data as input. Non-numerical data in the file must be preceded by the pound sign (#) character in the first column.

ASCII files can only be read into the Graph Viewer through the **Read Data** function; they cannot enter the **graph viewer** module through an input port.

The Graph Viewer's assumption about ASCII data files is that they are basically *tabular*, that is, tables of columns of numbers. One column might be pressure, or density, or temperature; another column might be time, a Richter scale value, or daily rainfall, etc. You can select any one column to plot (such as pressure), or you can select any column and plot it against

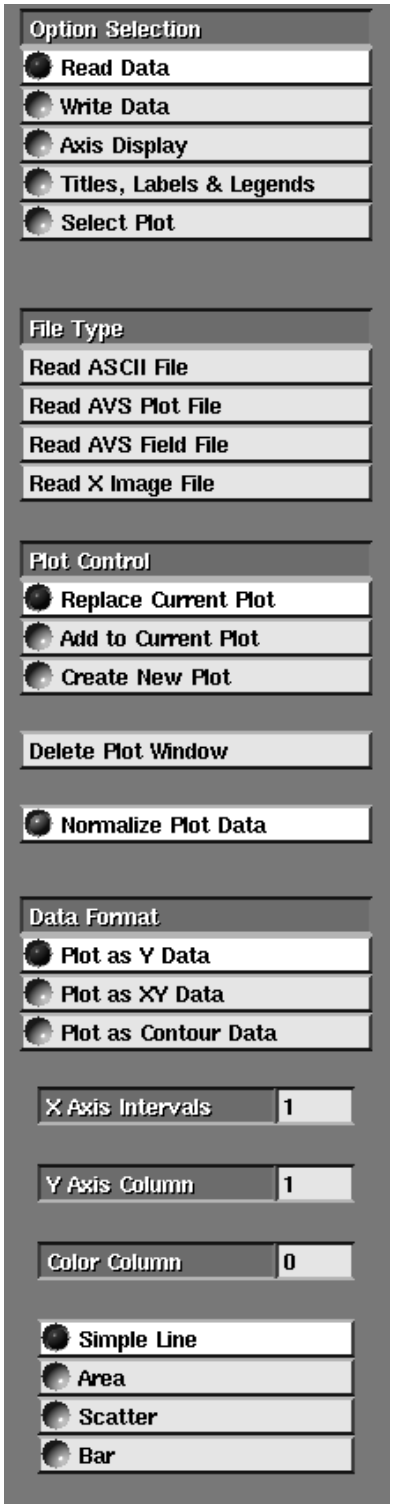


Figure 7-5 Read Data Submenu

any other (pressure against time). For example, here is the sample ASCII data file */usr/avs/data/graph/growth.dat*:

```
0 0 1
10 20  1
30 30  2
80 50  3
90 20  4
95 80  5
```

Choosing **Read ASCII File** causes the Data Format menu described below to appear.

Read AVS Field File

The Graph Viewer accepts AVS fields of any dimension, of any vector length, and of any coordinate type (uniform, rectilinear, irregular).

However:

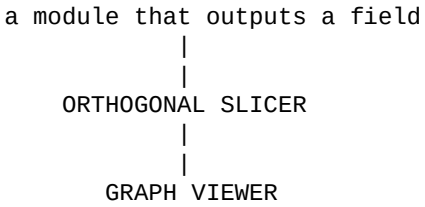
- If the field is 3D or greater, the Graph Viewer will, by default, use just the *first* XY 2D "slice" of the field. (The Graph Viewer does not create three-dimensional graphs.)
- If the field has a vector of data at each point, the Graph Viewer will graph just the first element of the vector.
- Uniform and rectilinear fields will be plotted on a corresponding grid, preserving the original spatial relationship of the data. Irregular (curvilinear) data will be plotted on a corresponding grid if the input field is 2D (ndim=2) and 2 space (nspace=2). Otherwise, irregular data is plotted on a uniform grid.
- For a field file to show in **Read Data**'s file browser, it must have the file suffix *.fld*. You can read in any field file without this suffix using the file browser's **New File** typein panel.

Choosing **Read AVS Field File** causes the Data Format menu described below to appear.

It is more typical to graph field data entering the Graph Viewer *through a network* rather than reading the field data iVn through this **Read AVS Field File** function.

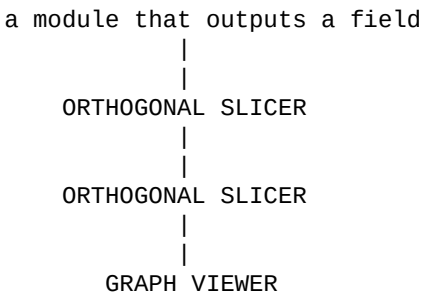
AVS fields enter the **graph viewer** module through either its rightmost or center input ports. *By default* the rightmost port assumes you wish to make a linear plot, and the center port assumes you want to make a contour plot; but either can be changed from the Graph Viewer interface.

Because the Graph Viewer will, by default, plot just the first XY plane of a 3D field, you should use one or more **orthologgnal slicer** modules in the network to pare down the field to a dimensionality that is meaningful to the Graph Viewer, and that contains just the data you want to graph. (The **crop** module can also be used.) To turn a 2D AVS field into a 1D field or to turn a 3D field into a 2D field use a network like the following:



Use the **orthogonal slicer**'s controls to pick the column or plane of the field you want to graph. This is how, for example you would create a 2D slice of a 3D scatter data field that the Graph Viewer could then use to make a contour plot.

If you have a 3D field that you want to plot as a 1D **Plot as Y Data** operation, use two **orthogonal slicer** modules in succession:



If the field contains vector data, you should use the **extract scalar** module to isolate the vector element that you want to graph. (The Graph Viewer graphs just the first vector element in a field.) This could come before or after any **orthogonal slicer** modules.

The Graph Viewer also accepts the following two non-data input files:

Read AVS Plot File

AVS Plot files are files that you have already created with the Graph Viewer, and saved with the **Write Data** and **AVS Plot Data** buttons. The plot file does not contain numeric data. Rather, it contains the instructions for re-drawing a Graph Viewer window, with plot lines, axes, titles and axis labels exactly as they were originally saved.

For a plot file to show in **Read Data**'s file browser, it must have the file suffix *.plt*. You can read in any plot file without this suffix using the file browser's **New File** typein panel.

Note: These files create a new window when selected—they cannot be read into an existing window.

Read X Image File

The Graph Viewer uses the term **X Image File** to mean an AVS image datatype file with the *.x* suffix. The Graph Viewer uses the image file as a backdrop to its plotting field. Image files can be read in with **Read Data**. They can also enter the **graph viewer** module through its leftmost input port.

When an image enters the Graph Viewer, the plot window is resized to fit the image.

For an image file to show in **Read Data**'s file browser, it must have the file suffix *.x*. You can read in any image file without this suffix using the file browser's **New File** typein panel.

Plot Control Submenu

The Plot Control Submenu selects among three methods for reading plot data into the plot window. You should establish your choice *before* you read in the data file through the file browser, and before the upstream module sends its AVS field or image to the **graph viewer** module.

Replace Current Plot

This selection replaces all of the plots in the current plot window with a new plot dataset.

Add to Current Plot

This selection is used when the you want to read in a new plot dataset and add it to the current plot window. This allows you to display multiple plots in the same plot window. You can quickly compare several datasets within a single window using the same axis and tick values.

You can also mix plot styles within the same window — for example, dataset A as an area plot and dataset B as a scatter plot. Because a dataset plot style can be changed at any time, you can start with the same plot styles and change one window plot, or start with dissimilar plot styles.

Each dataset read into a window as input is automatically assigned a different color. You can change its plot characteristics (line type, scatter character, color) at any time.

For example,Figure 7-6 shows three plots in one window: a scatter plot where points are represented by the letter "A," a bar plot, and an area plot. Because the plots are within the same window and use the same axis values, their comparison is easy.

The axis and tick values of a window plot are always updated to reflect the largest dataset read into the window, unless **Normalize Plot Data** (below) is deselected.

Create New Plot

This selection creates a new plot window.

Normalize Plot Data

This switch controls whether a plot's dimensions change when new data enters the Graph Viewer.

When the **Normalize Plot Data** button is on (the default), the Graph Viewer automatically normalizes the plot dimensions. For example, if the plot's x-axis range was (0.0, 100.0) and you read in a dataset with an x range of (0.0, 200.0), the plot's x-axis is automatically resized to (0.0,

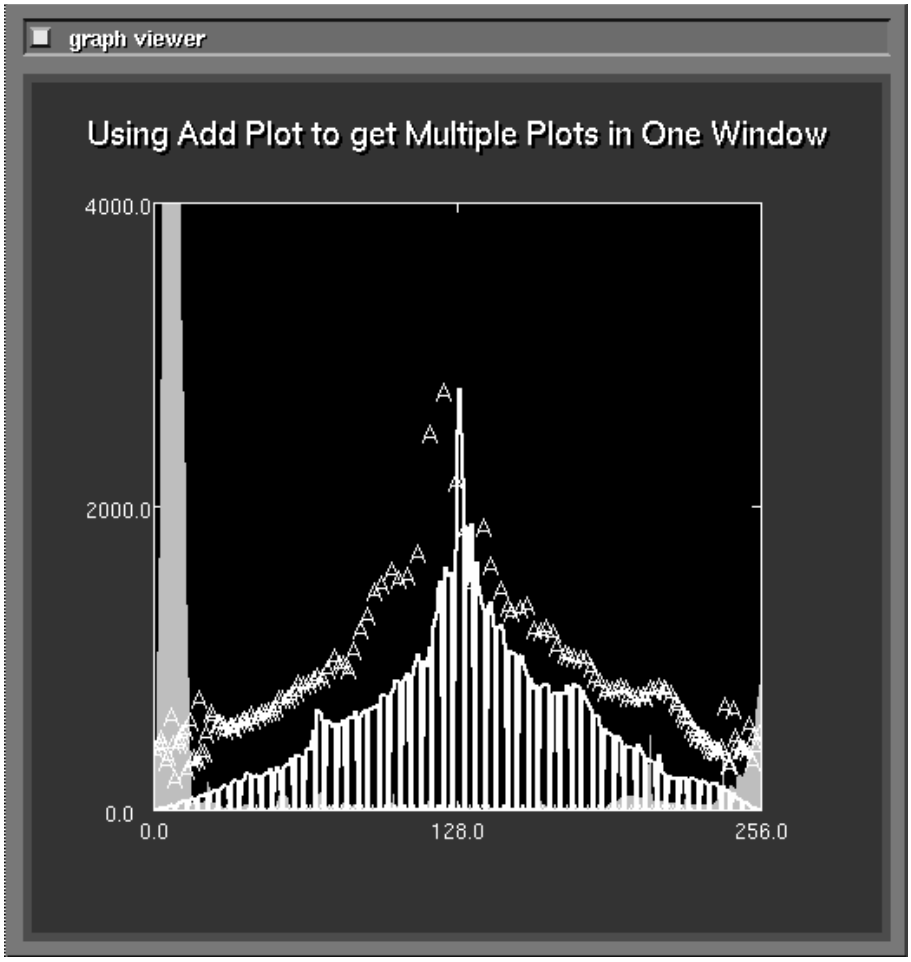


Figure 7-6 Adding New Plots

200.0). When **Normalize Plot Data** is off, the plot dimensions do not change when a new dataset is read into the Graph Viewer.

In practical use, you will want to normalize your data whenever you are not sure of the range of the data, or if you want to make sure that the entire dataset is visible within the plot area.

You would not want your data normalized in situations such as a strip chart. For example, if you had a network that is monitoring an instrument such as an EKG, you would not want the plot's dimensions to be normalized each time the plot is updated with a new data value.

Data Formats Submenu

Once you have selected an input file type (ASCII, etc.), then the Data Format submenu and the file browser widget appear.

There are three different ways for the Graph Viewer to map its input data into a plot. You select the technique from the following set of radio buttons:

Plot as Y Data

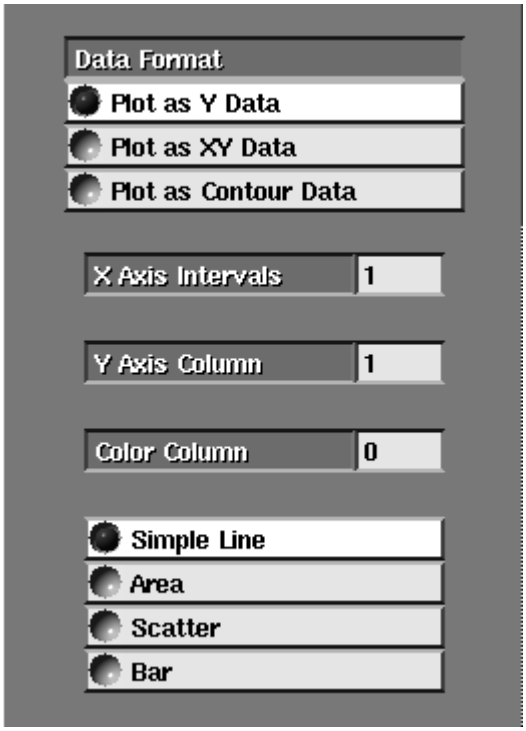


Figure 7-7 Plot as Y Menu

Directs the Graph Viewer to use *one column* of the input data as a series of Y values. If the input data is a 2D field, it is viewed as a two-dimensional array, where each X value (X=1, X=2, etc.) is a column of Y values. By default, column one of either an ASCII file or AVS field will be used. The X Axis for the graph is generated automatically. The Graph Viewer counts the number of data items in the Y Axis column and places the same number of items on the X Axis in intervals of 10. You can change which column of the input data is used.

X Axis Intervals

Sets the interval range between item counts on the X Axis. You can change the interval range at any time.

Y Axis Column

Selects which column of the input data to use as Y values. You must select the data column before inputting the file.

To change the input column after reading in the file you must delete the dataset (see **Delete Plot Dataset** in **Select Plot**) or delete the window (see **Delete Plot Window**), then reselect the file for input with the input column you want.

If you type in an invalid column number, the Graph Viewer displays a message showing how many columns are available. Repeat the steps to select a valid column number for input.

Color Column

You can also use column data to control line segment color (see "Using Column Data For Color Control" below).

Plot Styles

You can select **Simple Line**, **Area**, **Scatter** and **Bar** plot styles for displaying the data. The default setting is **Simple Line**. You must select the plot style prior to reading the file. To change the plot style after you have read the file see "Select Plot."

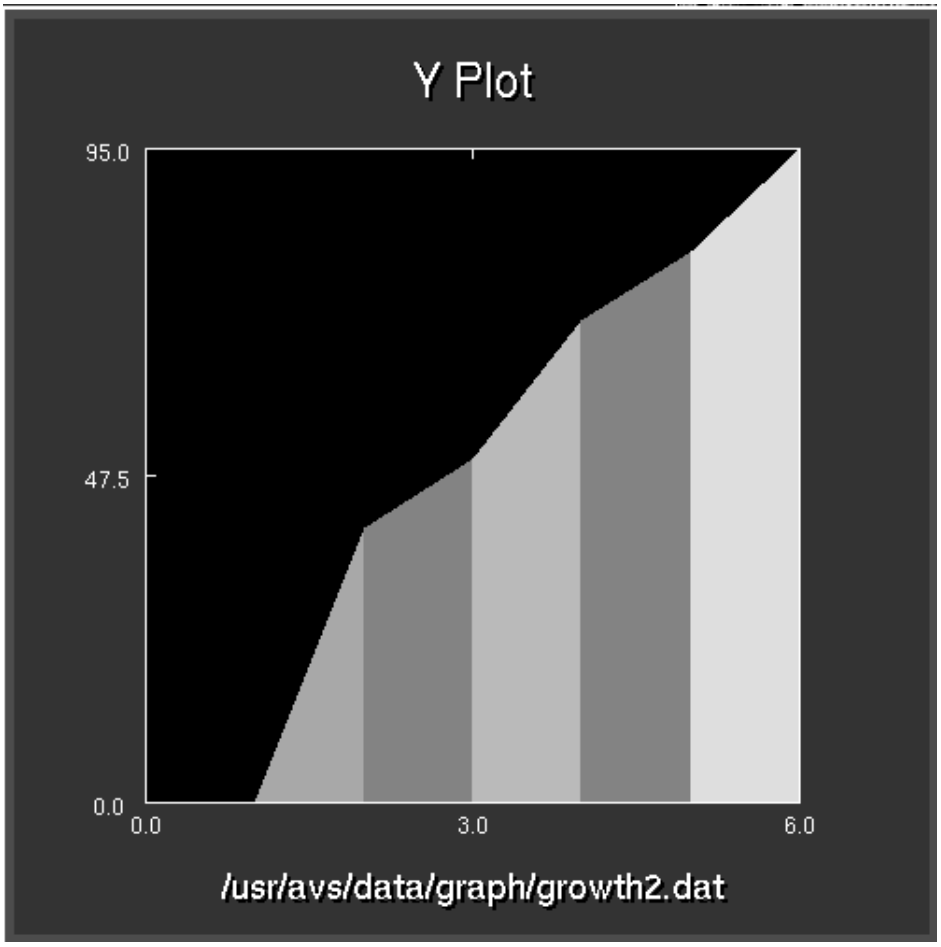


Figure 7-8 Example of Plotting as Y Data

Plot as XY Data

Directs the Graph Viewer to use two columns of the input as a pair of X,Y data values to be plotted against correspondingly scaled X and Y axes. By de-

fault, the Graph Viewer uses column one as the X Axis values and column two as the Y Axis values. You can change the input columns as follows:

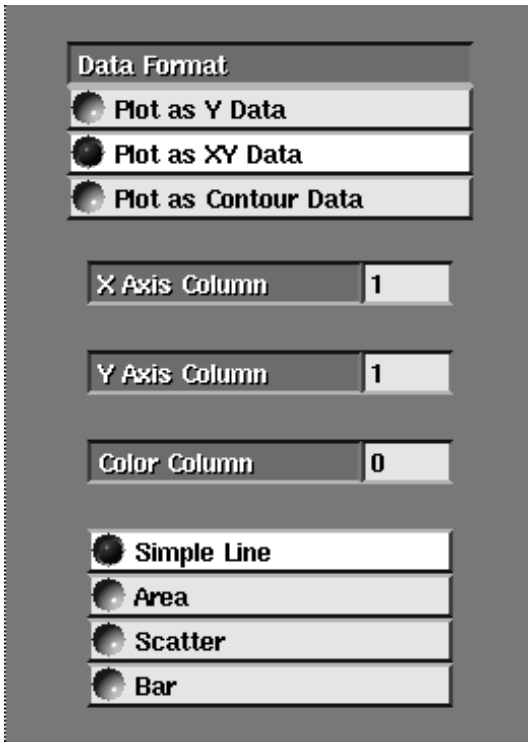


Figure 7-9 Plot as XY Data Menu

X Axis Column

Selects which column of the input file or field to use as the X values. You must select the data column before inputting the file.

Y Axis Column

Selects which column of the input file or field to use as the Y values. You must select the data column before inputting the file.

If you need to change the X Axis or Y Axis input column after reading in the file you must delete the dataset (see "Delete Plot Dataset" in "Select Plot") or delete the window (see **Delete Plot Window**), then reselect the file for input with the input columns you want.

If you type in an incorrect column number the Graph Viewer displays a message showing how many columns are available. Repeat the steps to select a valid column number for input.

Color Column

You can also use column data to control line segment color (see "Using Column Data For Color Control").

Plot Styles

You can select **Simple Line**, **Area**, **Scatter** and **Bar** plot styles for displaying the selected data. The default setting is **Simple Line**. You must select the plot style prior to reading the file. To change the plot style after you have read the file see **Select Plot**.

Note: It does not usually make much sense to plot typical AVS field data as **Plot as XY Data**. You would essentially be plotting one column of X, Y, or Z values against another column of the same data—X1Y1 against X2Y1, X1Y2 against X2Y2, etc.). One could write a module that constructs an output field where this operation was meaningful, but no supplied modules do this.

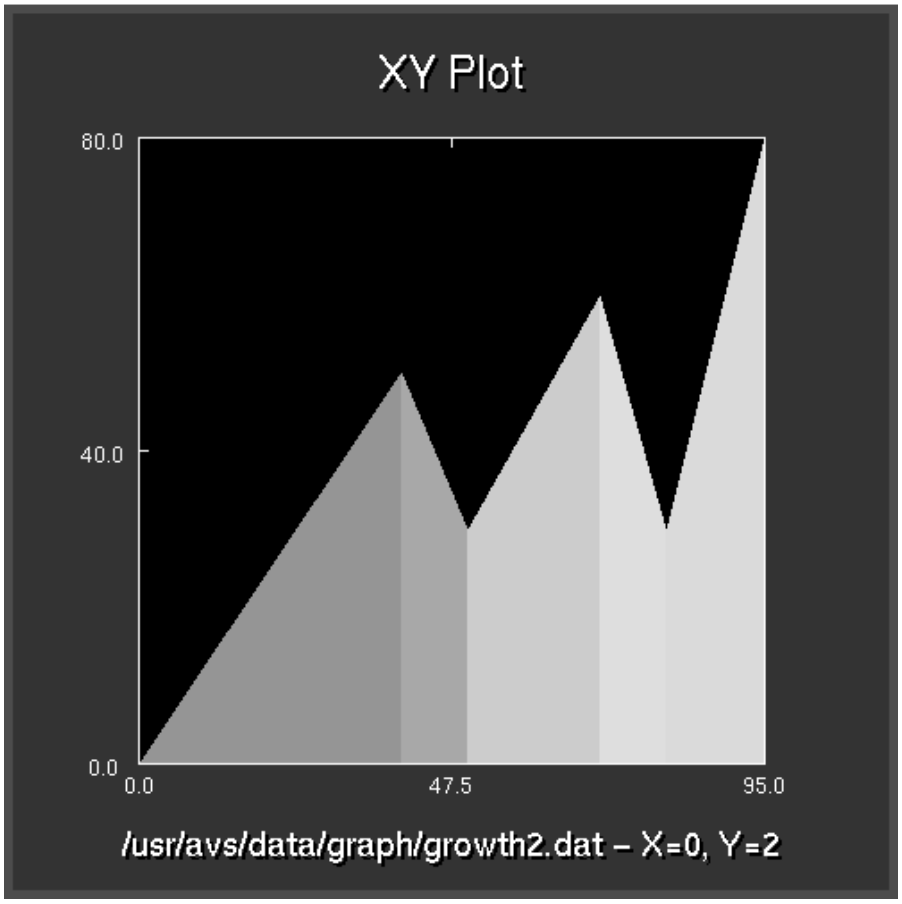


Figure 7-10 Example of Plotting as XY Data

Plot as Contour Data

Directs the Graph Viewer to use all columns of the input data to construct a 2D contour plot. By default, each contour level is displayed as a different color. You can also select monochrome contour mapping to assign all levels the same color. The number of contour levels and coloring can be changed at any time.

If you want to make a contour plot of a 3D field, remember that the Graph Viewer plots only the *first* XY plane of field. Use the **orthogonal slicer** module in a network to select the 2D planar slice of the data from which to create the contour plot.

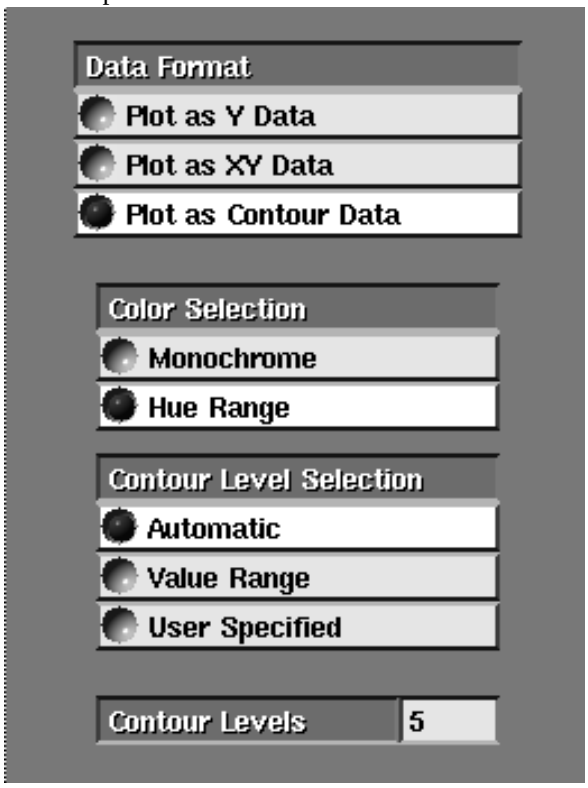


Figure 7-11 Plot as Contour Data Menu

Note: because of the complexity of contour plot data, the computing and display time can be significant. Be aware of the size of your input data, and the number of levels you are trying to plot. If necessary, you can use the **downsize** module to "thin out" AVS field data before it enters the **graph viewer** module, and the **Value Range** or **User Specified** controls listed below to keep the number of levels reasonable.

Color Selection

There are two options for color selection of contour plots.

Monochrome
Displays all contour levels of the plot in the same color.

Hue Range
Displays each contour level of the plot in a different color. This is the default. The range is not user-selectable.

Contour Level Selection

You specify the number of contour levels. The range of data (from low to high) is divided into evenly spaced contour levels. Data occurring on each contour level is assigned a color and plotted.

For example, data containing values from 0 to 90 can be assigned 10 contour levels. The contour levels are evenly divided between the low value and the high value (inclusive with the high value). Therefore the contour levels are placed at the following values: 0, 10, 20, 30, 40, 50, 60, 70, 80, and 90. It should be noted that contour lines will not be generated at a dataset's minimum (0) and maximum values (90).

If you are not sure what the range of your AVS field data actually is, you can find out with modules such as **statistics** and **generate histogram**. **statistics** produces an ASCII output widget showing the data minimum and maximum values; and **generate histogram** produces a 1D field profiling the distribution of data values in a field that is *meant* to be displayed with the **graph viewer** module.

There are three **Contour Level Selection** options:

Automatic
: Divides the entire range of values into evenly spaced contour levels. In this case, you only specify the number of contour levels you want. It should be noted that the minimum contour value will not be the minimum value of the dataset. It will be a value greater than the minimum value. This ensures that a contour line is drawn. Also, the maximum value will not be the maximum value of the dataset. It will be a value less than the maximum value.

Value Range
: Selects a range between a low number and a high number and divide the range into contour levels. For example, you can read a file ranging from 1 to 50 but specify a starting value of 10 and ending value of 40. Only values occurring between these points (inclusive) are permitted. Then this range is evenly divided into the number of contour levels you specify. Only data points occurring on the contour levels are plotted.

User Specified
: Explicitly states the contour values to be plotted. See the figure below. For example, you can read a file ranging from 0 to 500, then specify the contour levels of 26, 48, 89, 150, 223, 415, 487 and 499. Only data points occurring at these levels are plotted.

Using Column Data for Color Control

Color control enables you to select the colors used for displaying line segments. You need to choose an input column of data values to control the line segment colors.

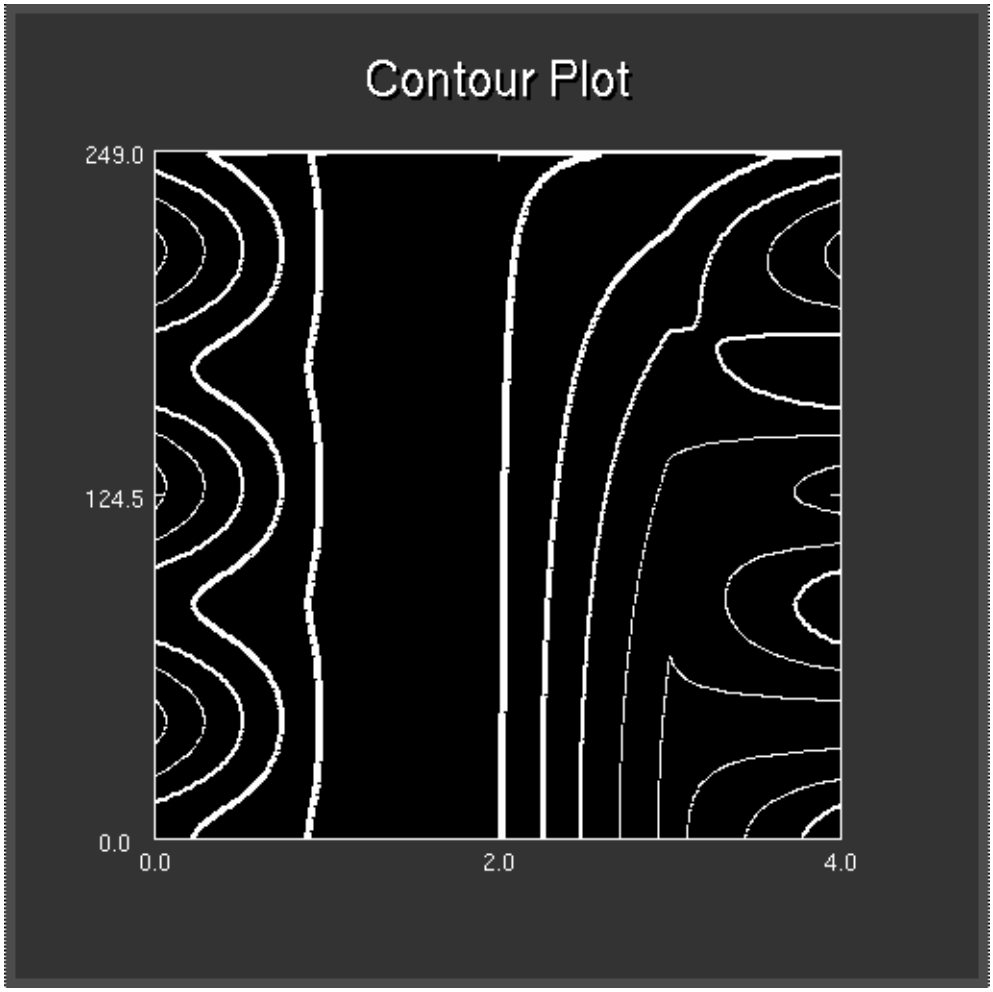


Figure 7-12 Example of Plotting as Contour Data

A data value of one tells the Graph Viewer to use color one, a data value of two means use color two, and so on.

Note: A value of 0 means no color displayed.

For example, if Column Y has three data items (Y1, Y2, Y3) and Column X has three items (X1, X2 ,X3), then Column C items (C1, C2, C3) can control the colors used to connect line segments Y1-Y2 and Y2-Y3.

There are three data values in Column C, but only two (C2 and C3) are actually used for line colors. The first data value of the color control column is always ignored and reset to zero.

Color control values range from 0 to 499 to control Hue color. If your color control data value is greater than the number of available colors, the data value is wrapped around to the beginning color.

For example, a value of 500 wraps around to 0, 501 wraps to 1, 502 wraps to 2, and so on...

Decimal values are rounded to the nearest integer. Values between one and zero are rounded to either one (color one) or zero (no color).

Plot Styles Submenu

The Graph Viewer provides four styles of plots for viewing linear data (see Figure 7-13). You can change from one style of plot to another at any time. You can display different plot styles within the same window. The plot styles are:

Line plots

Line plots connect data points with lines so that you can easily see changes or trends within your data. You can change the type of line (e.g. solid, dashes), its thickness and color at any time.

Area plots

Area plots fill in the data points with solid color so that similar and dissimilar data points are easily viewed. You can change the color of the filled area at any time.

Scatter plots

Scatter plots show the data points as a character (e.g., an asterisk) so that groupings of data can be easily seen. You can change the character type, style, size and color at any time.

Bar plots

Bar plots display data in vertical bars so that you can easily compare data values. You can change the color of the bars at any time.

Delete Plot Window

Deletes the active display window. There is no request for confirmation—once a window is deleted its display is gone.

Write Data

Saves the selected plot data for later retrieval or printing. Selecting this menu item causes the **File Type** menu to appear.

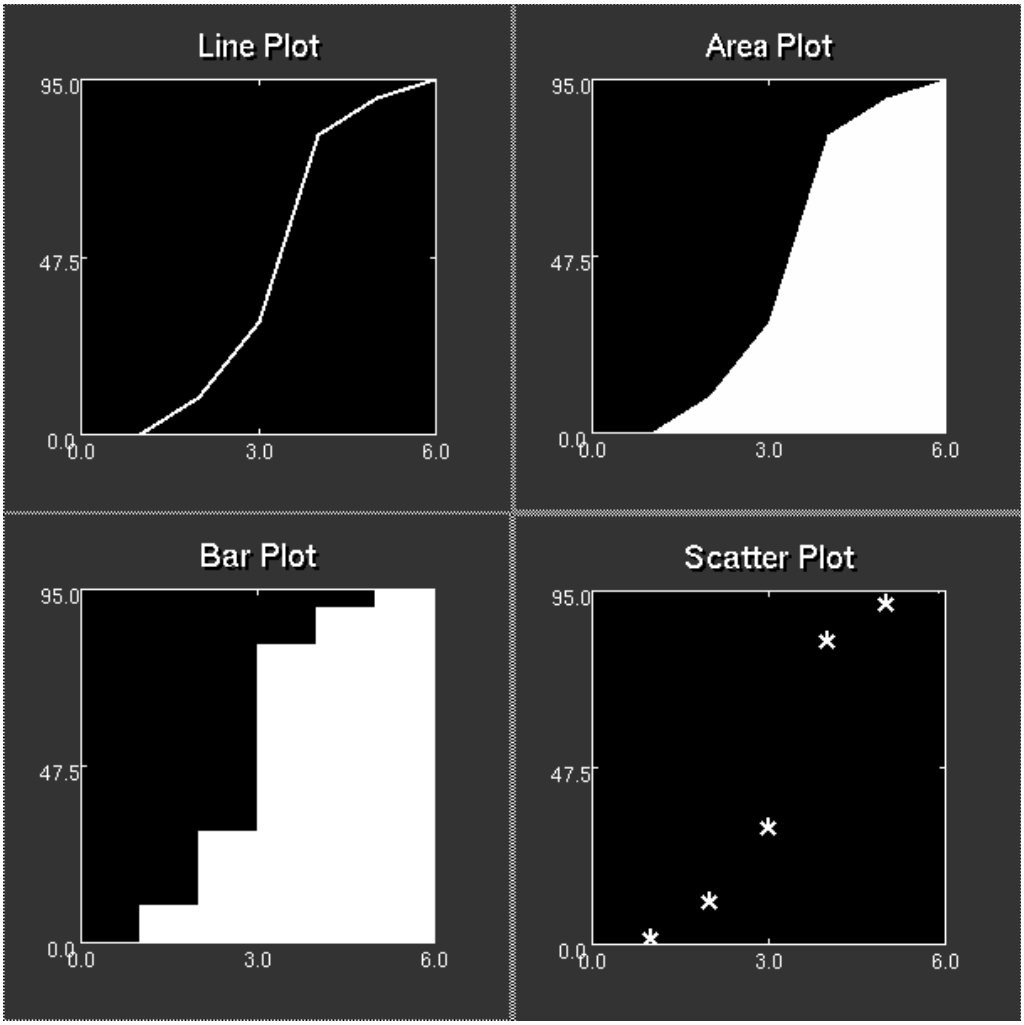


Figure 7-13 Plot Styles

File Type Submenu

You can select the following file types for output:

Write ASCII XY Data

Saves the selected plot data in a two column ASCII file. Title, axis and tick mark information (if any) is not saved. This format is typically used to save data without additional plot parameters in the file, possibly for use by another program. Note that a one column input file is saved with the generated X Axis as a two column output file. AVS automatically appends the *.dat* suffix to the output filename.

Write AVS Plot Data

Saves the selected plot in AVS Plot format. Title, axis, tick mark and plot size information are saved. This format is typically used to save a completed plot for retrieval and display. AVS automatically appends the *.plt* suffix to the output filename.

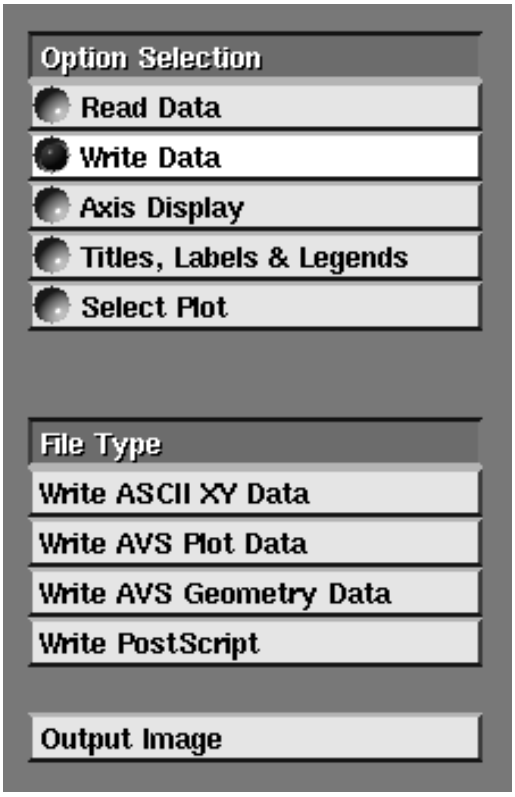


Figure 7-14 Write Data Option

Write AVS Geometry Data

Saves the selected plot in AVS Geometry format. Title, axis and tick mark information is included. This format is typically used to save a completed plot for further use in the AVS Geometry Viewer subsystem. AVS automatically appends the *.geom* suffix to the output filename.

Write PostScript

Saves the selected plot in PostScript format. Title, axis and tick mark information is included. This format is typically used to save a completed plot for printing on a PostScript printer. Color information and background images are not saved in the PostScript file, nor sent to the printer. AVS automatically appends the *.ps* file suffix to the filename. **Output Image** below can also create PostScript files, with some additional flexibility provided by the **image to postscript** module (encapsulated PostScript, scaling, landscape / portrait).

In order to actually write the file, you must click on the file browser's **New File** button, then enter the filename in the typein panel that appears.

Output Image

This button is only active when you are using the Graph Viewer in its **graph viewer** module form in an AVS network. The **graph viewer** module has an image output port. This output port allows you to convert the active plot window into an image. In turn, this image can be sent to modules such as **display image** and **image to postscript**.

Because the conversion of a plot into an image is a computationally intensive operation, the Graph Viewer does not update the image output port every time the current plot is changed. In order to get an image sent out through the **graph viewer** module's image output port, you must press this **Output Image** button.

Axis Display

This menu option controls axis display information for the current plot window. You can change these settings at any time. See Figure 7-15.

Border Display

Sets the border partially or completely around the selected plot.

Left & Bottom
Displays the border on the left side and bottom of the plot.

Full Border
Displays the border around all four sides of the plot.

Axis Selection

Selects the X or Y axis of the plot for control of all of the following menu items. Only one axis can be selected at a time.

X Axis
Selects the X Axis of the plot.

Y Axis
Selects the Y Axis of the plot.

Axis Scale

Sets the selected axis of the plot as linear or log based. Each axis can be only linear or log based.

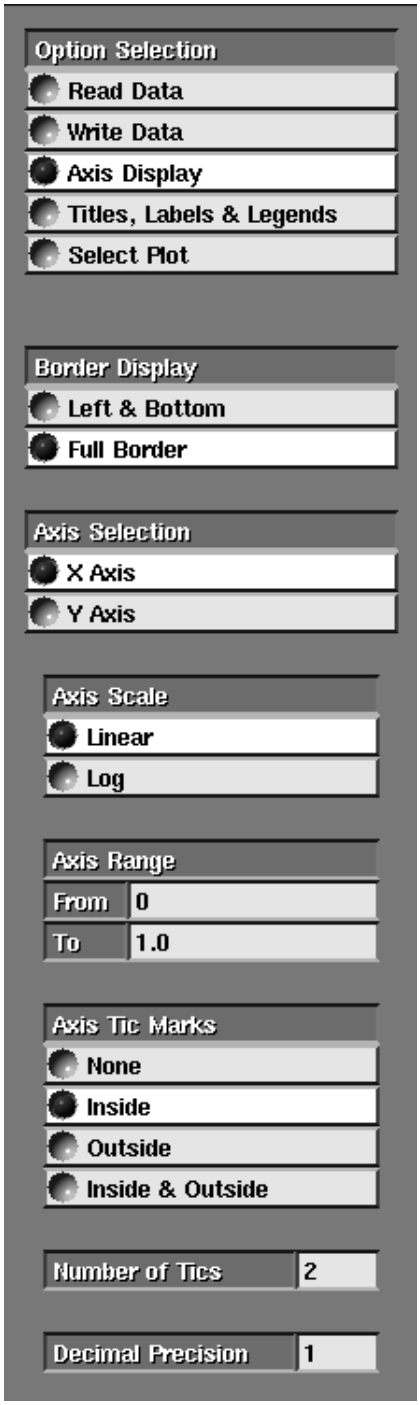


Figure 7-15 Axis Display Options

Linear
Sets the selected axis as a linear base display.

Log
Sets the selected axis as a log base display.

Axis Range

Sets the low and high data input thresholds of the selected axis. The defaults are the lowest and highest data values of the axis dataset.

This enables you to select a superset of the data as input for the selected axis. For example, you can narrow the range of 0-100 to 25-75 for a better examination of data between these points.

From
Sets the lowest value of the dataset allowed on the selected axis. The default is the lowest dataset value.

To
Sets the highest value of the dataset allowed on the selected axis. The default is the highest dataset value.

Axis Tic Marks

Selects the type of tick marks (if any) that are displayed for the selected axis. The default setting is for tick marks to be displayed **Inside** the plot axis. Only one of the following can be active at a time:

None
Sets the tick marks to none for the selected axis—meaning no tick marks are shown.

Inside
Sets the tick marks inside the selected axis.

Outside
Sets the tick marks outside the selected axis.

Inside & Outside
Sets the tick marks both inside and outside the selected axis.

Number of Tics

Selects the number of tick marks shown on the selected axis. The default value is 2 tick marks. The axis is redisplayed with the new value of tick marks.

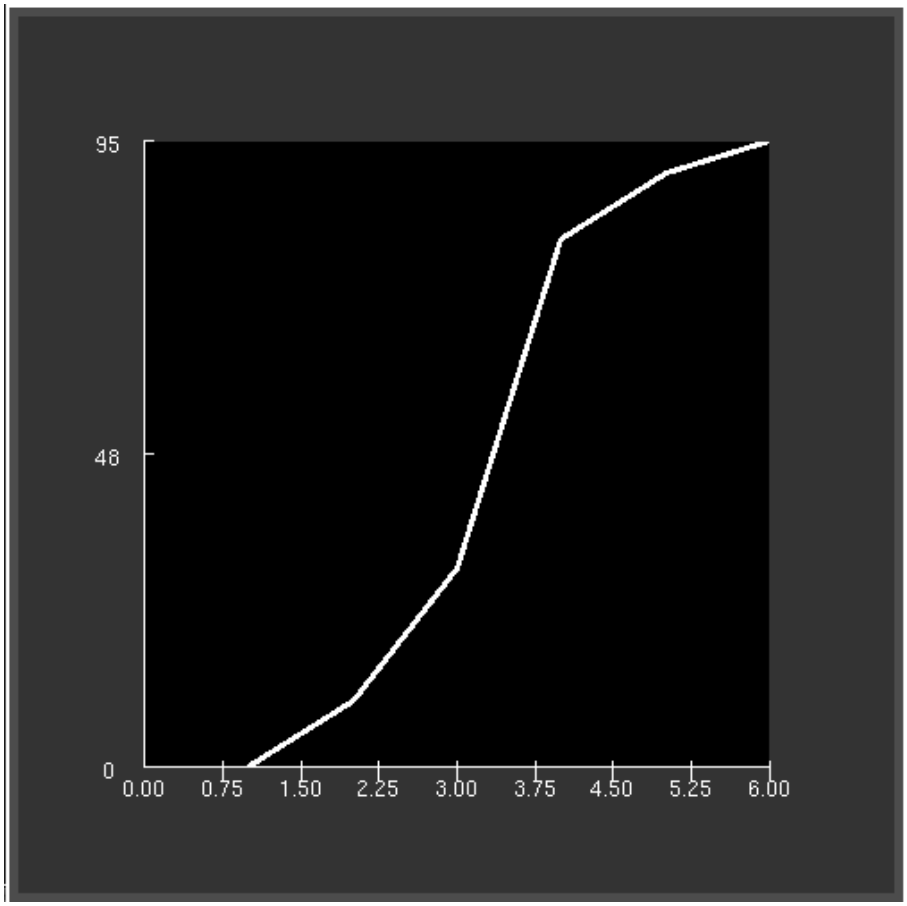


Figure 7-16 Effect of Varying Axis Display Options

Decimal Precision

Sets the decimal precision values for the selected axis. The default value is one position right of the decimal. The affected menu item is **Axis Range** (**From** and **To** values).

The affected menu items are not shown with the new precision until the **Axis Display** menu is redisplayed. If you need to see the new precision after changing the value, select another menu item (e.g., **Read Data)** and then reselect **Axis Display**.

The figure shows a plot where the Y axis is unchanged, while the X axis has had its intervals reset, its decimal precision increased, and its tick marks drawn both above and below the X axis line.

Titles, Labels & Legends

This menu option controls the titles, axis labels, and legends for the current plot window. You can change these settings at any time. See Figure 7-17.

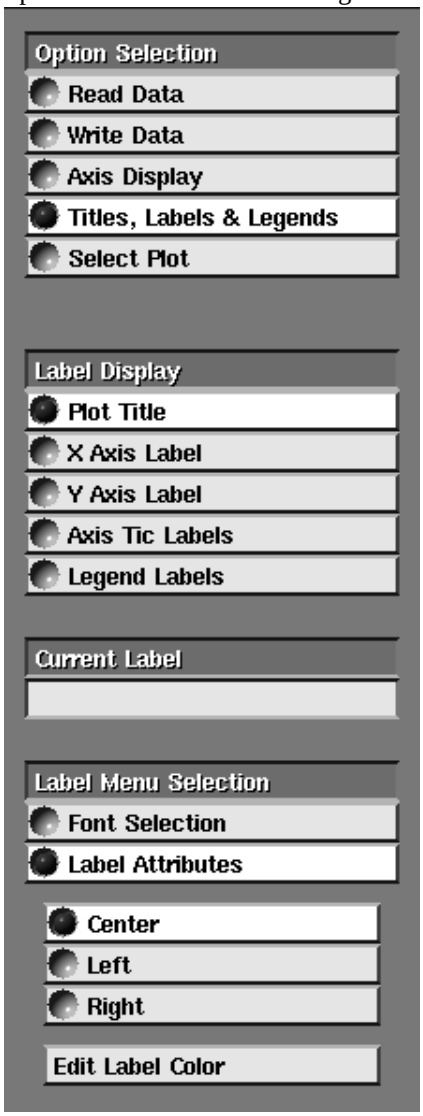


Figure 7-17 Titles, Labels & Legends Option

Label Display

Selects between the title, axis labels, tick labels, and legend labels for changing text and attributes.

Plot Title

Selects the current plot window's title text and attributes for addition or change. If there is no current title the Current Label data area remains blank. If a title exists it is shown in the Current Label data area. Attributes (size, style, location, color) can be changed at any time (see "Label Menu Selection" later).

Perform the following steps to add or change Title text:

1. Select **Titles & Labels** in the **Option Selection** menu.
2. Select **Plot Title** in the **Label Display** menu.
3. Place the mouse pointer in the **Current Label** data area (it will highlight).
4. Press **Ctrl-U** to delete the existing entry (if one exists).
5. Type in the new entry and press **Return**.

X Axis Label

You can select the **X Axis Label** text and attributes for addition or change. If there is no current X Axis Label the **Current Label** data area remains blank. If an X Axis Label exists it is shown in the **Current Label** data area. Attributes (size, style, location, color) can be changed at any time (see "Label Menu Selection" later).

Perform the following steps to add or change X Axis Label text:

1. Select **Titles & Labels** in the **Option Selection** menu.
2. Select **X Axis Label** in the **Label Display** menu.
3. Place the mouse pointer in the **Current Label** data area (it will highlight).
4. Press **Ctrl-U** to delete the existing entry (if one exists).
5. Type in the new entry and press **Return**.

Y Axis Label

You can select the **Y Axis Label** text and attributes for addition or change. If there is no current Y Axis Label the **Current Label** data area remains blank. If a Y Axis Label exists, it is shown in the **Current Label** data area. Attributes (size, style, location, color) can be changed at any time (see "Label Menu Selection" later).

Perform the following steps to add or change the Y Axis Label text:

1. Select **Titles & Labels** in the **Option Selection** menu.
2. Select **Y Axis Label** in the **Label Display** menu.
3. Place the mouse pointer in the **Current Label** data area (it will highlight).
4. Press **Ctrl-U** to delete the existing value (if one exists).
5. Type in the new value and press **Return**.

Axis Tic Labels

You can select the **Axis Tic Labels** for changing attributes (see "Label Menu Selection" later). Tick mark labels are generated automatically us-

ing settings in the **Axis Display** menu. No value or entry is shown in the **Current Label** data area when **Axis Tic Labels** is selected.

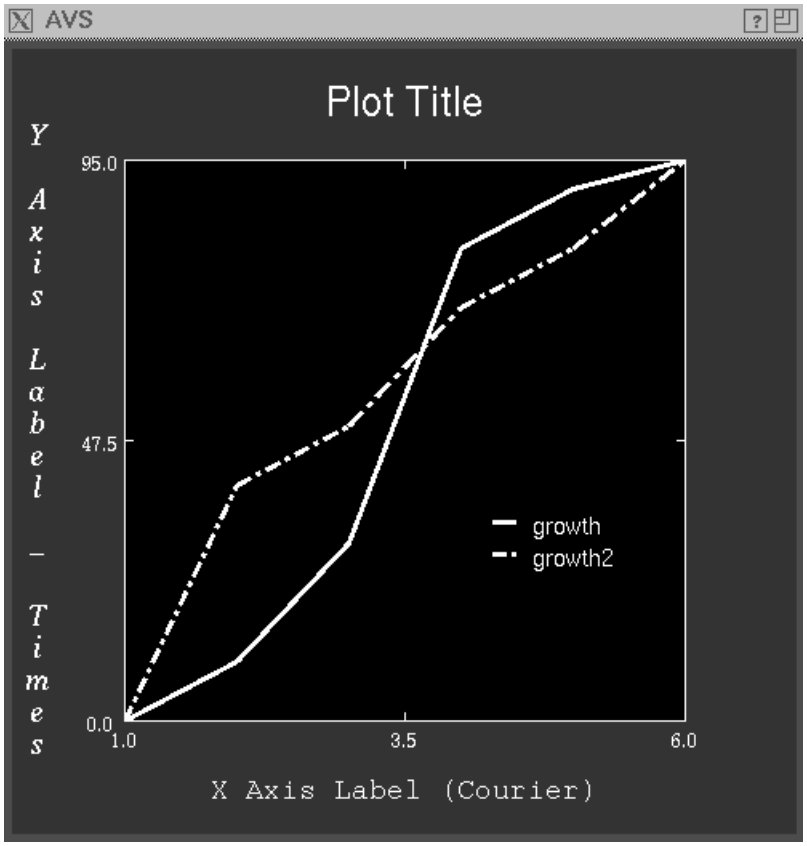


Figure 7-18 Effects of Varying Titles, Labels & Legends Options

Legend Labels

To create legends:

1. Position the mouse pointer over the plot line to be selected.
2. Click any mouse button to select the plot line.
3. The selected line will be highlighted. Type into the **Current Label** area to set the plot line's legend label.

Legend Position

By default, the legend will appear in the upper right hand corner of the plot. This position can be altered by changing the X and Y typein values in the **Legend Position** submenu. The values for X and Y can range between 0.0 and 1.0. For X, 0.0 is the left side of the plot and 1.0 is the right side of the plot. For Y, 0.0 is the bottom of the plot and 1.0 is the top of the plot.

Label Menu Selection

Selects between **Font Selection** and **Label Attributes** for the label chosen in the **Label Display** menu. Only one can be selected at a time.

Font Selection

Selects the type of font used for the label chosen in the **Label Display** menu. Only one font for the chosen label can be used at one time. You can use different fonts for different labels. Fonts can be changed at any time. The following list of fonts is representative; different font lists will appear on different hardware types.

Courier
Is a typewriter style font.

Helvetica
Is a sans serif style font.

New-Century
Is designed for easy reading.

Times
Is a bolder appearing font.

Charter
Is a modern serif font.

Symbol
Is a font with a mix of Greek characters.

Bold
Provides a darker character outline for the selected font. It can be used alone or with **Italic** and **Drop Shadow** attributes.

Italic
Provides an angled script-like appearance for the selected font. It can be used alone or with **Bold** and **Drop Shadow** attributes.

Drop Shadow
Provides a 3D shadow appearance for the selected font. It can be used alone or with **Bold** and **Italic** attributes. (Note: this control may not be present on all systems.)

Label Height
Controls the display size of the label chosen in the **Label Display** menu. Size can range from 0 to 40 points. The current size is shown in the **Label Height** menu area.

Place the mouse pointer in the **Label Height** menu area (it will highlight) and drag the mouse pointer rightward to increase or leftward to decrease

size. The changing point size is shown as you move the mouse. Release the mouse button to set the new label size.

Label Attributes

Changes the position and color of the label chosen in the **Label Display** menu. Position can be set to one of the following:

Center
Places the label in a centered position.

Left
Places the label in a left of center position.

Right
Places the label in a right of center position.

Edit Label Color

Creats a color editor panel that will control the color of the label. Label color is adjusted by dragging the mouse pointer on the color bars.

Select Plot

This submenu (Figure 7-19) lets you change the plot attributes (plot type, line type, color) of already-plotted data.

There may be multiple plots in a single plot window. Perform the following steps to select an individual plot within the current plot window:

1. Choose **Select Plot** on the Option Selection menu.
2. Position the mouse pointer over the plot to be selected.
3. Click any mouse button to select the plot.

The plot changes color to white to show it has been selected. The selected point on the the plot is highlighted with a circle (O). Note that clicking the mouse button finds the *closest real data point*, not the "closest line."

The plot remains selected until another plot (if any) within the same window is selected. Perform the above steps to select another dataset plot within the same window.

Display Crosshair

Displays a crosshair marker as the mouse pointer moves over the window display area (Figure 7-20). This makes the mouse pointer location within the

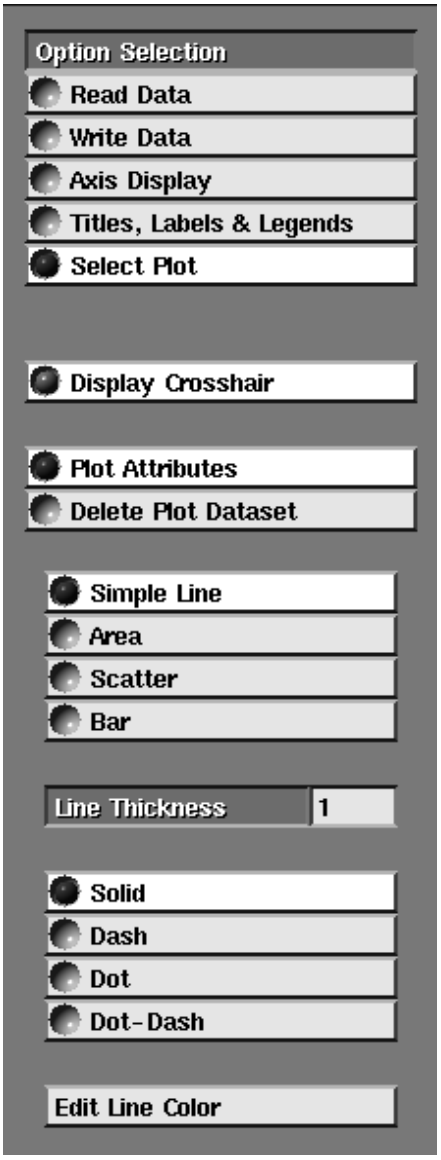


Figure 7-19 Select Plot Options

window area easily seen (see also **Cursor Position** and **Selected Point** below).

The default setting is for the crosshair to be displayed. The **Display Crosshair** menu item lighted means the crosshair is displayed. Unlighted means the crosshair is not displayed.

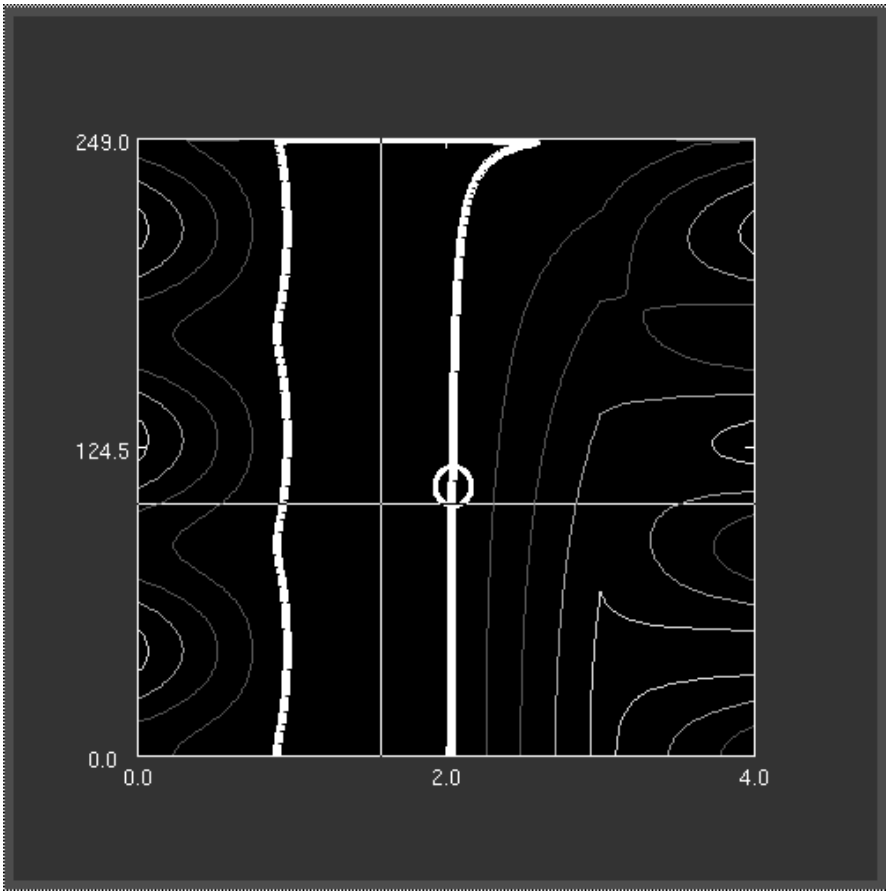


Figure 7-20 Crosshair with a Selected Contour Plot

Plot Attributes

These controls change the plot type and plot attributes. Plot type can be changed at any time. Line type is available only when the plot type is **Simple Line**.

Simple Line

Displays the dataset as a line plot. Line type can be a solid line, dashed line, dotted line, or dot-dash line. Line thickness can also be changed. The default setting is **Simple Line** as a **Solid** with a **Line Thickness** of zero. **Simple Line** types are as follows:

Solid
Displays a solid line between data points.

Dash
Displays a dashed line pattern between data points.

Dot
Displays a dotted line pattern between data points.

Dot-Dash
Displays an alternating dot-dash pattern between data points.

Line Thickness
A typein that controls the line thickness of a dataset plotted as a **Simple Line**. The default line thickness is zero. **Line Thickness** applies to all four **Simple Line** types.

Area

Displays the dataset as an area plot. Coloring can be changed at any time.

Scatter

Displays the dataset as a scatter plot. The default scatter plot symbol is the asterisk (*) character. Font style, height and coloring can be changed at any time.

Scatter Plot Symbol
It can be changed to any displayable character by deleting the default asterisk and typing in the character you want.

Symbol Height
A slider bar that controls the height of the scatter plot symbol. Height can range from 0 to 40 points. The current size is shown in the **Height** widget.

To change the height, place the mouse pointer in the **Symbol Height** menu area (it will highlight) and drag the mouse rightward to increase or leftward to decrease size. The changing point size is shown as you move the mouse. Release the mouse button to set the new size.

The actual size displayed on your screen depends upon the X Window font sizes installed. The exact size shown is printed on Postscript printers.

Font Styles

The font styles listed below are representative; font lists vary among hardware types.

Courier
Is a typewriter style font.

Helvetica
Is a sans serif style font.

New-Century
Is designed for easy reading.

Times
Is a bolder appearing font.

Charter
Is a modern serif font.

Symbol
Is a font with a mix of Greek characters.

Bar

Displays the dataset as a bar plot. Coloring can be changed at any time.

Edit Line Color

Creates a color editor panel that controls the color of the plot lines, bars, area, or scatters. This can be changed at any time by selecting the dataset and dragging the mouse pointer on the colorbar(s) you want. You cannot change the color of the black, background plotting field. However, you can replace it with an AVS image file.

Delete Plot Dataset

Deletes the currently-selected plot from the current plot window. Once deleted, the plot is lost forever. Confirmation of the delete is required:

Confirm Delete
Verifies the deleting of a plot. Once deleted the plot is lost forever.

Cursor Position

Shows the relative position of the mouse pointer within the plot axis coordinates. Coordinates are based entirely upon the axis values within the plot.

For example, a plot with an X Axis range from 0 to 60 and a Y Axis from 0 to 100 results in a minimum coordinate of 0.00; 0.00 and maximum coordinate 100.00; 60.00 displayed.

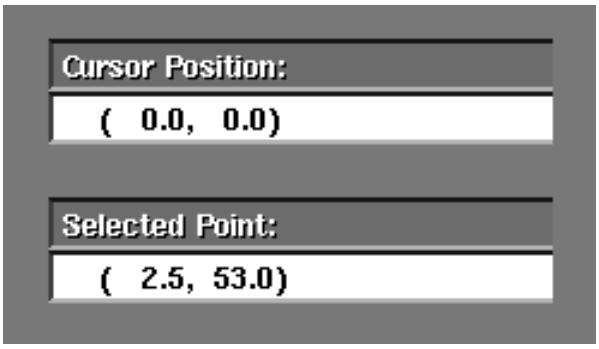


Figure 7-21 Cursor Position Display

Moving the mouse pointer within the plot area causes the coordinate display to vary. The left coordinate is the X Axis and the right coordinate is the Y Axis.

Selected Point

Shows the coordinate value of the selected point within the plot. Select a point by placing the mouse pointer within the plot area and clicking any mouse button. The X-Y Axis coordinate for that point is then displayed in the **Selected Point** data area. This control does not display the *data value*; just the coordinates.

Graph Viewer Command Language Interpreter

It is possible to drive the Graph Viewer with commands rather than the X display interface. The commands can be either typed in interactively from a terminal emulator window while AVS is running, or they can be read from a script file.

This opens many possibilities:

- One could create scripts that animate the Graph Viewer.
- One could create demonstration, illustration, and test scripts.
- One could create scripts that batch-process graphs.

To run AVS with the Command Language Interpreter (CLI) active, type this:

```
avs  -cli  other-options
```

This starts AVS as usual, but also starts the CLI command line interpreter in the invoking window. (You might have to press carriage return to get the **avs>** prompt.)

To get a list of the Graph Viewer CLI commands, type the following:

```
avs>  help Graph
```

This produces a list of the many Graph Viewer CLI commands. To get help on an individual commands, type "help" plus the command name:

```
avs> help graph_output_image
graph_output_image  Send an image to the Graph Viewer module's output port
    Usage:  graph_output_image
avs>
```

There are sample Graph Viewer CLI scripts located in the directory */usr/avs/demosuite/General/Graph*.

The Command Language Interpreter and the Graph Viewer set of CLI commands are documented in detail in the "Command Language Interpreter" chapter of the *AVS Developer's Guide*.

CHAPTER 8

ADVANCED NETWORK EDITOR

Introduction

The "Network Editor Subsystem" chapter described the basic Network Editor operations necessary to construct, use, and save visualization networks. This "Advanced Network Editor" chapter describes features that the typical AVS user will need as he/she moves from experimenting with AVS to using it on a production basis. This includes:

- A module's parameters can send data to and receive data from other modules just like other module input/output ports. Thus, module parameters can be used to control the behavior of other modules in the network. A special case of this is **upstream data**, where downstream modules are able to communicate information back "upstream" to modules that execute earlier in the network.
- Sets of modules can be interactively grouped together to form one **macro** module. Macro modules behave in the same way as individual modules. The macro module can be added to the module Palette, saved, read in, and made part of a module library.
- Networks can contain a mixture of modules that execute locally, and modules that execute on a remote host that runs AVS . The remote host can be "heterogeneous"— of a different hardware type than your workstation.
- The Network Editor's Layout Editor can redesign the user interface to a network. Control widgets, including those of macro modules, can be reorganized onto different-sized control panels, making it easier to perform visualization tasks without having to be knowledgeable about network construction.
- You can create libraries of modules to support an individualized repertoire of visualization functions.
- Production networks can be optimized in a variety of ways:
  - Where multiple processors are available, modules can execute in parallel.
  - The user can see and control which process a module executes in.
  - The performance of networks containing curvilinear field data may be improved by instructing AVS to use adaptive block tables.

Parameter Ports

Modules can receive and send *parameters* through one or more **parameter ports**. Parameters are the values generated by module widgets or by a another module that sends the parameter value to other modules through its output port(s). **Parameter ports are normally invisible.**

Parameter data (integers, floating point values, boolean on/off switches, strings of characters) control some aspect of the module's execution. An integer parameter might determine which slice plane to take through a volume, or which element of a vector to extract and map. Floating point values might control what values should be used to construct an isosurface (3D contour). Switches can control whether or not to interpolate data. Strings might specify what file a module should read.

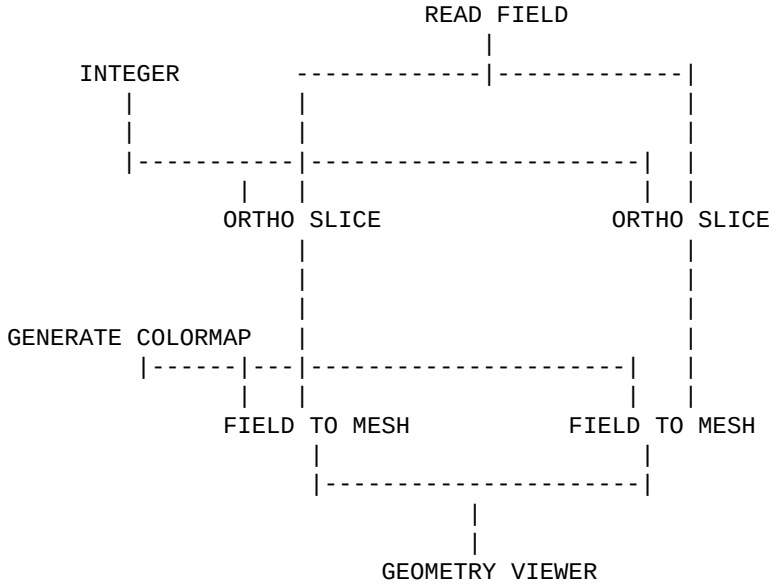
The parameters generated by widgets normally just affect the module to which they belong. However, it is possible to generate parameter values and send them to other modules' widgets through an AVS network connection.

Here are some examples of what you can do with parameter ports:

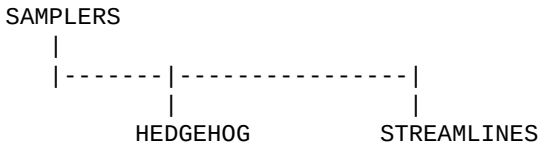
- A single module producing parameter-type output can send the same parameter value simultaneously to multiple modules.

  For example:

  Here an integer parameter module connected to a dial widget produces two orthogonal slices through the field in different planes (I and J) but at the same offset value. The results are converted to geometries and composited together.

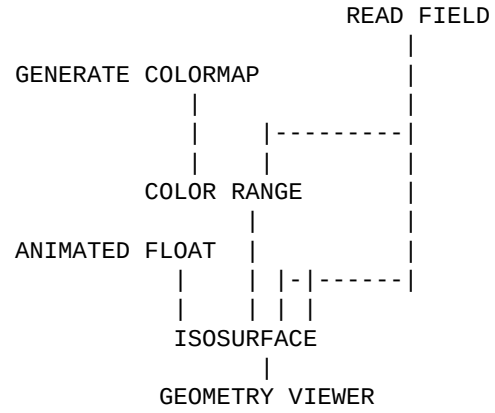


  Here a single **samplers** parameter module causes **hedgehog** vector arrows and streamlines to appear simultaneously for the same set of sample points:



- Coroutine modules that execute independently from the rest of a network, such as user-written simulations, can send an automated series of parameter values to other modules. For example, one could implement a "master clock" module for controlling multiple coroutine simulations.

  The data coroutine module **animated_float** is an example. On this module's control panel, you set minimum, maximum, and step values. When switched on, **animated_float** sends a stream of evenly-spaced floating-point values to another module's floating-point parameter port. Used with the **isosurface** module's "level" parameter, this animates a sequence of isosurface contours. One can animate **any** floating-point port on an AVS module in this way.



Connecting Parameter Ports

AVS comes with a set of parameter modules that generate each of the standard parameter data types. They appear in the Supported Module Palette in the Data Input column. The parameters modules are as follows:

integer (light purple)
float (dark purple)
file browser (grey blue)
boolean (light purple)
oneshot (white)
tristate (light purple)
character string (grey blue)

In addition there are two coroutine modules that produce a stream of parameter values that can be used to make automated animations:

animated integer
animated float

One module, **field legend**, takes field and colormap input, but outputs a single floating point parameter. Another parameter module, **euler transformation** outputs a small 2D field representing a transformation matrix for input to **tracer**.

For more information, including networks which show these modules in use, see the *AVS Module Reference* manual, or call up the online module documentation by clicking on the **Show Module Documentation** button in each module's Module Editor panel. You can also click on the **Help** button at the top of the Network Editor menu. This brings up the Help Panel. On the Help Panel is a **Help Demos** button. This raises a browser of automatic scripts that you can run that illustrate the use of most key modules in sample networks.

Creating the Connection

Because some modules have many parameters, showing all the parameter ports on all the modules in the palette and network would be confusing. Therefore, input parameter ports are invisible by default.

Before you can connect output parameter ports to input parameter ports, you must make the input parameter ports visible.

1. Call up each module's Module Editor window by clicking on the module icon's "dimple" with the middle or right mouse button.
2. The module's parameters are listed in the Module Editor window. The associated color bar shows the parameter type. Click on the parameter's button to bring up its Parameter Editor panel.
3. The Port Visible button on the Parameter's Editor panel is gray, showing that it is off. Click on this button. A colored parameter port will appear on the module's upper (input) edge. (Note: if you **close** the main Module Editor panel, it will also close the Parameter Editor panel.)

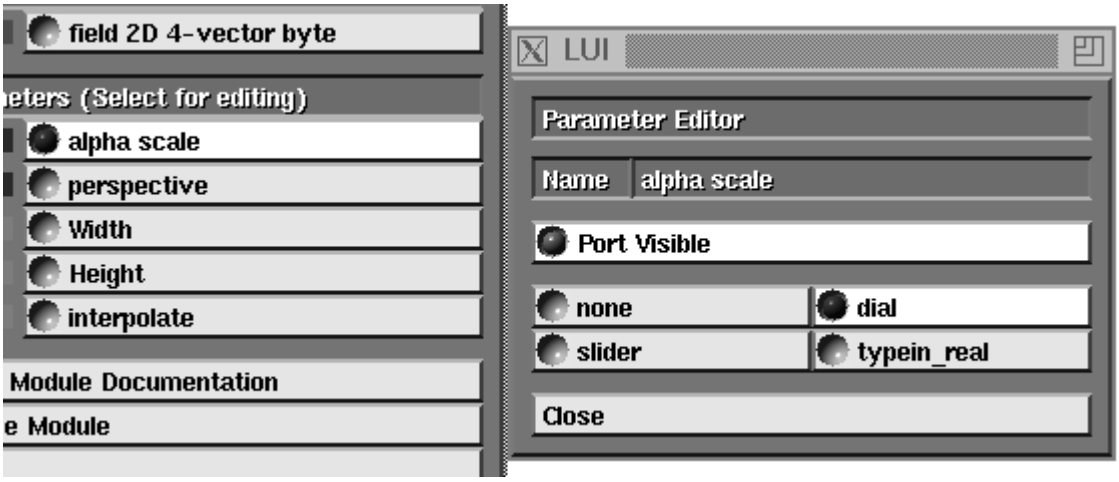


Figure 8-1 Making Tracer's Alpha Scale Port Visible

Connect and disconnect the parameter ports in the usual way: middle mouse button to make a connection; right mouse button to disconnect ports.

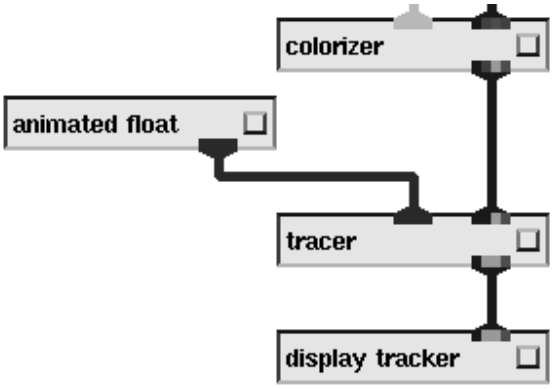


Figure 8-2 Animated Float Parameter Module Connected to Tracer

Upstream Data Ports

Data in an AVS network normally flows from top to bottom. There is one exception—if two modules can agree on a data structure, then if module B receives input from module A, then module B can also send the agreed-upon data structure *back up* the network to an input port on module A. This is called **upstream data**. Modules can control the visibility of upstream data ports and the connections between them. **Generally, upstream data ports and the connections between them are invisible.** The upstream connection between the two modules usually occurs automatically, once any other connection between them has been made. This is also under the control of the modules involved.

Two important uses for this upstream data feedback mechanism are: you can have direct mouse manipulation control of "data highlighting" objects such as slice planes and data probes; and you can pick structures like chemical bonds by clicking on them.

AVS supplies two kinds of feedback data, the **upstream transform** and the **upstream geometry**, to flow from data output modules such as **geometry viewer** back up the network to the data mapper modules such as **brick**, **probe**, and **arbitrary slicer**.

For example, the **probe** module takes a data field as input. It outputs an object to the **geometry viewer** module (Geometry Viewer) that looks like an electronics probe. The idea is that you use the Geometry Viewer's virtual trackball mouse cursor to move this probe around the volume of data and the **probe** module tells you what numeric values are present at any given point in 3D space.

The Geometry Viewer knows where you have moved the mouse cursor. But the Geometry Viewer does not know what the numerical values in the field are at that point.

To produce this information, The Geometry Viewer outputs an **upstream transform** which says where in 3-space the probe has moved *back up the network* to an input port on the **probe** module. **probe** takes this "the probe is at X, Y, Z" information, maps it to the correct data cell in its data field, producing a numeric value for the probe's position, which it sends back to the Geometry Viewer for display.

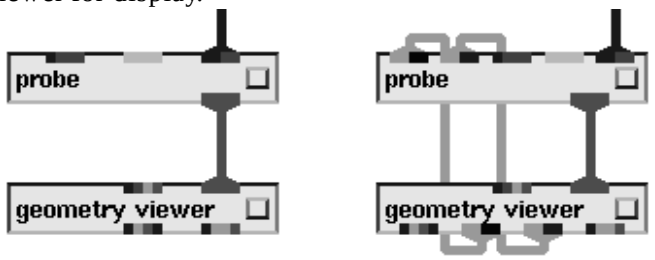


Figure 8-3 Upstream Data Ports Made Visible

Picking bonds in a molecule is much the same problem. The Geometry Viewer knows where a user has pointed the mouse and clicked, and it knows what geometry vertex is intersected by a ray pointed at the object from the mouse cursor, but it knows nothing about the molecular structure. It sends an **upstream geometry** reporting the selected vertex back up the network to a molecular mapper module. This module can translate the vertex to a particular molecular bond, then highlight it, delete it, or whatever, producing a new output geometry to give to the Geometry Viewer. (The **probe** module also supports picking in its **Pick Geometry** mode.)

Connecting Upstream Data Ports

You don't have to do anything to use upstream data—connections happen automatically and transparently when you construct an AVS network. When a module with upstream data output ports, such as **geometry viewer**, has another module connected to one of its input ports, it automatically checks to see if that module has a compatible upstream input port. If it does, it connects its upstream output port to the previous modules upstream input port. (This automatic connection only happens between adjacent connected modules.)

The following module pairs use upstream data and can be used together to produce direct manipulation:

geometry viewer	and	arbitrary slicer
geometry viewer	and	brick
geometry viewer	and	hedgehog
geometry viewer	and	probe
geometry viewer	and	streamlines
geometry viewer	and	thresholded slicer

geometry viewer and animated lines
geometry viewer and samplers
geometry viewer and particle advector
geometry viewer and ucd crop
geometry viewer and ucd probe

display tracker and tracer
display tracker and gradient shade (no auto-connect)

hedgehog and samplers
streamlines and samplers
animate lines and samplers

The **display tracker** module is interesting. It does not use either an upstream geometry or an upstream transform. Rather it passes a 4x4 field upstream to control the **tracer** module. Any two modules that can agree on an upstream data type can use the upstream feedback mechanism.

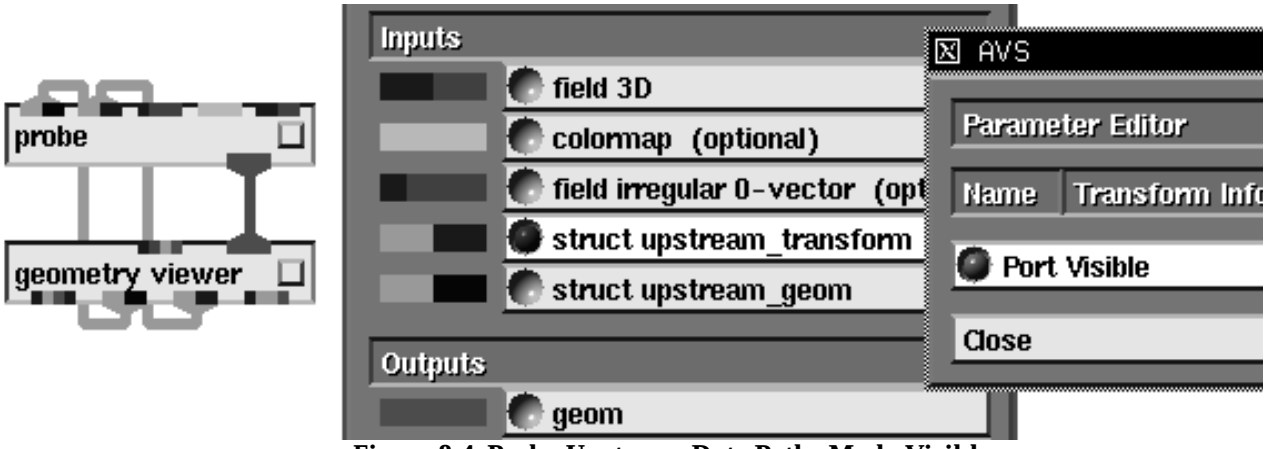


Figure 8-4 Probe Upstream Data Paths Made Visible

You can make upstream connections visible. Bring up the Module Editor panel for **both** modules by clicking the middle or right mouse button on their icon "dimples". Find the color-coded **struct upstream-transform** or **struct upstream-geom** input or output button. Click on that button to bring up its Parameter Editor panel, then click on **Port Visible**. When both output and input sides of the connection have been made visible, the upstream flow pipe shows in green. (See Figure 8-4.)

You can disconnect upstream connections. However, if you disconnect an upstream transform to a module, the object for that module in the Geometry Viewer will "go dead". That is, you will no longer be able to move it with the mouse buttons or **Transformation Options** panel. You will still be able to move it from the mapper module's widget controls.

A better way to get Geometry Viewer control over the upstream probe or slice plane is to use the **Override** button on the Geometry Viewer's Transformation Options panel.

Editing Tools: Macro Modules

The Editing Tools menu controls two fundamental Network Editor operations: network editing operations such as copying, cutting, and pasting groups of modules, and **macro** module editing operations.

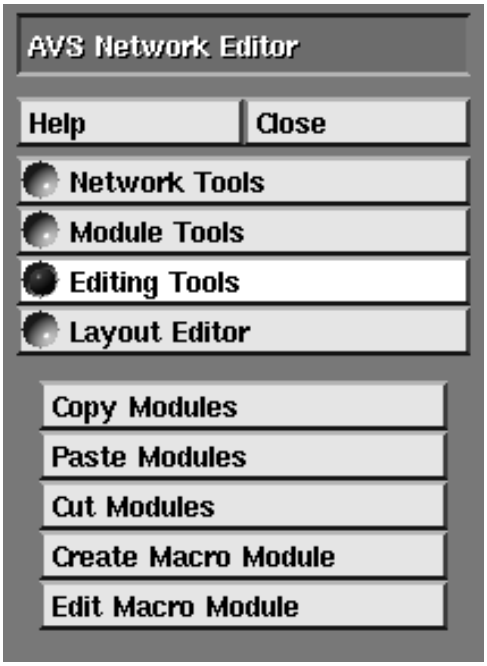


Figure 8-5 Editing Tools Menu

Selecting Module Subsets

Network editing operations are performed on the **currently selected module subset**. To select a group of modules:

- Click and hold down the left mouse button when the cursor is in the workspace but not on any particular module. This defines one corner of the rectangular region that contains the current module subset.
- Drag the mouse button to expand the rectangle so that you fully enclose all of the modules that you desire in the module subset.
- Release the mouse button to complete the selection operation.

Any modules that are contained within this rectangle are highlighted and added to the current module subset.

Note that you may need to rearrange the location of your modules on the Workspace in order to group all of the modules that you desire and prevent undesired modules from being included in the group.

To cancel the selected group, press and release the mouse button anywhere in the work space outside of the currently selected rectangular region.

Copy/Cut/Paste Modules

These operations allow you to cut/copy and paste a subset of the network. This speeds the creation of networks that have duplicate groups of modules in them.

Cut and copied modules are stored in a **current module buffer**. You can copy a group of modules, then clear the network, then paste the group of modules. The current module buffer is maintained throughout the AVS network editing session until the **Copy** or **Cut** operations are used again. If you **Exit** out of the Network Editor, the buffer contents are removed.

When you cut, copy and paste modules, all changes that have been made to their layout are carried with them.

Copy Modules

The **Copy Modules** button copies the currently selected group of modules, their connections; and their layout into the current module buffer. If no modules are selected when this button is pressed, it clears the current buffer. The currently selected group of modules is cleared when the Copy operation is performed. The module buffer can be used in a subsequent **Paste Modules** operation.

Cut Modules

The **Cut Modules** button copies the currently selected group of modules, their connections, and their layout into the current module buffer. It then deletes each of the modules in the current selected group. The current module buffer can be used in a subsequent **Paste Modules** operation.

Paste Modules

The **Paste Modules** button merges the modules, connectivity, and layout information stored in the current module buffer into the active network. The location of the modules is determined by their original location in the workspace when they were copied/cut, plus an offset down and to the right for each time that the buffer has been pasted since the copy or cut operation.

Macro Modules

Macro modules allow the user to represent a collection of modules, connectivity and widget layout with a single module icon. When this module icon is brought down into the workspace, the Network Editor creates the underlying modules, connections, and widgets. These underlying modules are not visible to the user unless the user explicitly edits the macro module.

Here are two examples where this feature might be used:

- The network builder might want to instance the same module in multiple networks with a customized widget layout. Rather than having to bring the module down, then using the Layout Editor each time, the user could create a macro module that contained just the single module with a customized layout. This module is then instanced in each network.
- The network builder might want to reduce the visual complexity of a complicated network by combining pieces of the network. A **generate colormap** module could be combined with an **arbitrary slice** module to form a macro module called "slicer", for example.

There are some important facts about macro modules that you need to know:

- The description information necessary to reconstruct a macro module is stored in an ASCII file. This file can be used with a **Read Module** command to load the module into a module library. This file is called the macro module's **description file**.
- When a macro module is included in a saved network, the original macro description file is not used to restore the network. Rather, all of the information describing the macro module is included in the network file. This makes it easier to transport networks that make use of macro modules to other environments.
- A macro module does not affect the way in which the network executes. There are no efficiency gains or losses in using macro modules.
- Macro modules icons have a purple "dimple." They are thus easily identified in a module library or on the network workspace.
- Macro modules can contain other macro modules

Steps to Create a Macro Module

To create a macro module:

1. If you are creating a macro module out of modules that are already in the Workspace, then first lasso those modules together into a module group.
   - Use the left mouse button to draw a box around the modules. Press and hold down the left mouse button. Drag it until just the modules you want grouped together are highlighted. Release the left mouse button.

   Otherwise, just skip to the next step.
2. Click **Editing Tools** on the Network Editor's main control panel. The module editing submenu appears.
3. Click on **Create Macro Module**.
4. A pop-up appears. Type in a name for the macro module. You can change this later with the Module Editor panel.
5. The workspace is redrawn. There are several things to note:
   - The **IN->** module represents what will be the input ports on the new macro module. **OUT->** represents the output ports.

- The module control panel contains only the buttons for the selected modules.
- The complete state of the existing network has been saved.
- The complete Network Editor is still active: you can add, connect, and delete any modules from the Palette and Workspace that you require.
- At any time, you can cancel the macro module creation operation by:
  - Hammering either the **IN->** or **OUT->** modules.
  - Pressing **Clear Network** under **Network Tools**.

6. Connect the input ports of the "real" modules to the output port of **IN->.** Similarly, connect their output ports to **OUT->**. Each different data type produces an additional, matching input/output data port.
7. Decide which module library Palette category—Data Input, Filter, Mapper, or Data Output—the macro module should appear in. The default is Data Input.
   1. With the middle mouse button, click on **IN->**'s dimple to bring up the Module Editor panel.
   2. On the Module Editor, press and hold down **Change Module Category**. Roll down the pop-up to select a module category.

   The category can be changed later using **Edit Macro Module**.
8. To change the name of the macro module, type into the Module Editor panel's **Name** field. This may also be changed later with **Edit Macro Module**.
9. At this point, you may decide to edit the macro module's control panel. This is discussed below. If you do not edit the control panel, it appears just as it would if there was no macro module defined: each component module appears as a separate button. You can edit the control panel at a later time with **Edit Macro Module**.
10. To complete the macro module definition, press **Done Editing Level: 1** on the **Editing Tools** menu. You will be prompted for a filename. A filename is interpreted relative to the current directory in the shell from which AVS was started. Otherwise, enter a full pathname and filename. Like network files, macro module files have no file suffix convention.
11. The original, saved network reappears. The macro module is present in the Workspace, indicated by its purple dimple. Connect the macro module and proceed.

    The macro module is also in the module Palette, under whichever category you defined (default: Data Input).

With no other action, the macro module is available permanently as a file accessible through the **Module Tools** menu's **Read Module** or **Read Remote Module(s)**. For portability, a **Write Network** on a network containing a macro module will produce a network file that contains the entire definition of the macro module; not a file reference. Macro modules can be included in module library files.

You can change the definition of a macro module at any time with **Edit Macro Module** under **Editing Tools**.

Creating a Macro Module

Create Macro Module

The **Create Macro Module** function enters a mode where the user is creating a new macro module. The user is prompted to enter a name for the new macro module with a dialog box. The Create Macro Module operation can be canceled at this point by choosing **Cancel** on the dialog box.

After the user enters a name for the module, the current network state is saved, then a clear network operation is performed. If the user had selected a group of modules before pressing the **Create Macro Module** button, they are cut from the original network and then pasted into the the new macro module description.

In addition to any modules pasted from the original network, there are two new empty "modules", one at the top and one at the bottom of the network workspace. The top module is called:

```
IN-> <mod name>
```

The bottom module is called:

```
OUT-> <mod name>
```

The top module has a grey output port, the bottom module has a grey input port. The grey ports on the "input" and "output" stubs of the macro module can be connected to any data type. You can use these grey ports to create input and output ports for your macro module that will be visible when this module is used in a network.

Connecting a module to the grey output port of the module called: **IN-> <mod name>** will cause a new input port for your macro module with the type and name of the input port that you connected it to.

Similarly, connecting a module to the grey input port of the module called: **OUT-> <mode name>** will cause a new output port for your macro module that will have the type and name of the output port that you connected it to.

So that the user can create additional inputs and outputs, the grey port is moved one slot over to the left. When a port has no modules connected to it, it is deleted and all of the ports slide over one slot. The resulting macro module icon will resize itself to accommodate all input / output ports.

All normal network editing operations (including layout editor operations) are in affect during this time. The user can copy and paste network fragments, read / merge in network fragments, etc.

Editing Macro Module Category and Name

By default, the macro module is of type **Data Input** and has the name that you specified originally. When you complete editing the macro module, it will

appear in Data Input column of the current module library and have the name you originally specified. You can change the category and or name of the macro module by pressing on the purple dimple of either the **IN-> <mod name>** or **OUT-> <mod name>** modules. This will bring up the Module Editor panel. This panel contains a button called **Change Module Category**. Press and hold down the mouse with the cursor on this button and you will see a pop-menu that contains the choices available for the module category. Release the mouse button when the cursor is over the new choice.

Also in this menu is a typein that shows the current module name. You can change the module name by editing the name in this typein. The name field can only be edited for macro modules.

Saving Your Macro, Resuming Your Session

When you press the **Create Macro Module** button, you enter a mode in which three more menu entries appear on the Editing Tools menu: **Create Macro Stack**, **Create Macro Page**, and **Done Editing Level: 1**. The buttons **Create Macro Stack** and **Create Macro Page** are described later in this section. The **Done Editing Level: 1** button is pressed when you have completed editing the macro module and want to return to your original network context.

When this button is pressed, the user is prompted to provide a file name to save the changes to the "module macro". The description of the macro module is saved in the file that you specify. The module is added to the current module library under its assigned column (see the above section to change the module category), and the previous network state is restored.

If you had a group of modules selected when you pressed the **Create Macro Module** button, these modules will not be part of the network state that is restored. Instead, it creates an instance of the newly defined macro module.

The **Create Macro Module** button can be used even when already creating a macro module. In this way, you can create arbitrary levels of nesting of macro modules. The menu entry: **Done Editing Level: <n>** displays <n> as the level that you are currently editing.

Canceling a Macro Module

You can cancel the editing of a macro module by moving either the: **IN->...** or the **OUT->...** module to the hammer icon in the lower right corner of the workspace. This exits you from editing the macro module and restores the previous network state.

Another way to exit the macro state is to press the **Clear Network** button. If you are creating a macro module, you have three options: **Clear Network** which cancels all macro modules that you are currently editing/creating, **Clear Macro** which clears only the active modules in this editing request and **Cancel** which cancels the Clear Network operation.

Editing an Existing Macro Module

Edit Macro Module

The **Edit Macro Module** button can be used to make changes to an existing macro module. In order to use the **Edit Macro Module** button, the macro module must be instanced on the network workspace, and the only selected module in a module group. This is done by lassoing the module with the left button as described in the introduction to this section. If this is not the case, an error dialog message will appear.

Once you have pressed the **Edit Macro Module** button, you enter edit macro mode. This mode is exactly the same as the create macro module mode above with only a few minor exceptions. The initial module state of edit macro mode consists of the modules and connections contained in the macro module description. Another difference is that when you exit edit macro mode, you have the option of using its previous macro module file and the new edited version of the macro module is restored when the previous network state is restored.

Note that any connections between the macro module and the previous modules in the network are not maintained when you exit edit macro mode. This is done to prevent incompatibilities that might have been introduced while editing the macro.

See the **Create Macro Modules** section for more information on operations that can be used when editing a macro module.

Modifying a Macro Module's Widgets

A very common operation that macro modules are suited to is customizing the layout of the user interface of a group of modules. If you do not edit the layout of a macro module, the Network Editor Control Panel interface does not reflect the existance of the macro module: each component module has its own button and page of widgets. By using macro modules, you need only perform the Layout Editor operations once, and then use the resulting modules over and over again in different networks. The basic method to do this requires no extra features at all. When editing a macro module, you can use the **Layout Editor** functionality to delete widgets, change widgets, create new pages, move widgets from one page to the next, delete pages etc.

There are two other **Editing Tools** buttons that make these features a little bit easier to use:

Create Macro Stack

Pressing **Create Macro Stack** creates a hierarchical representation of a group of sub-modules, reflecting the hierarchical nature of the macro module.

In this interface style, the macro module appears as a single button on the Network Editor Control Panel, labelled with the macro module's name. When the user presses this button, a new series of buttons appears in the Control Panel—one for each component module. This is a "stack." It is la-

belled with the name of the macro module. In turn, pressing one of these buttons raises the individual "page" of widgets for the individual module.

This operation can be performed equally well through the Layout Editor but there are some important differences. By using the **Create Macro Stack** button, you do not need to add each module's panel individually; it is done automatically. Any new modules that are created while editing the macro module have their widget panels automatically added to the stack. Another important difference is that the stack that is created is associated with the macro module. When you click on the macro module's dimple, the stack will be raised. When you delete the macro module, the stack is deleted automatically.

Create Macro Page

In this interface style, all of the widgets associated with a macro module's component modules are collected together on a single page. Pressing **Create Macro Page** creates a new widget page that is associated with the macro module. You then use the **Layout Editor** to take the widgets off of their own individual pages and put them on the macro's page. Normally you would then delete the widgets own pages.

This button has significant differences from the **Create Page** button of the **Layout Editor**. The page created by the **Create Macro Page** button is associated with the macro module. When the macro module's dimple is pressed with the left mouse button, the page will raise automatically. More than that, the Network Editor knows to automatically delete pages that are associated with a module when the module is destroyed. This is not true of generic pages created with the Layout Editor.

Note: Only one page or stack may be associated with the macro module at any given time. If you select either **Create Macro Page** or **Create Macro Stack** you will need to delete this page / stack by hand with the **Layout Editor** before being able to choose the other option. Neither option can be selected more than once for a particular macro module.

Layout Editor

The AVS **Layout Editor** has been called one of the best, but least-known AVS features. Selecting **Layout Editor** places the Network Editor in a mode that allows you to "redesign" the user interface of an AVS network. You can, for example, place widgets outside the Network Editor control panel to use screen space more effectively.

By default, each module in the network has its own control panel, and all the control panels are assembled into the Network Control Panel window. You can switch among the various modules' panels, but you can see (and work with) only one at a time.

The facility for editing the layout of the Network Control Panel includes these features:

Figure 8-6 Layout Editor Menu

- Changing the "widgets" that provide interactive control over parameter values as a network executes. For example, you might change a *dial* into a *slider*, or into a *type-in*. Some parameters can be assigned to the spaceball or dialbox input devices.
- Moving widgets around within their control panels.
- Resizing browser control widgets.
- Moving widgets to other control panels.
- Creating *new* control panels. You might create a new panel, then move widgets from various existing control panels to the new one.

For example, if you are developing a network useful for visualizing your type of data as a convenient, packaged application, you should use the Layout Editor to design your interface to the application. You can put the most commonly-used widgets on a single control panel for immediate access. Perhaps you create a bar along the bottom of the screen that shows all the controls for all of the modules in the network at once.

For another example, the **print field** and **compare field** modules display the contents of an AVS field file as ASCII data that you can read and interpret. They are often used when you are trying to import data with **read field**. However, their scrolling *output* browser is very narrow—often too narrow to actually see most of the contents of the field. You can use the Layout Editor to make the output browser as wide as you need.

Any changes you make to the Network Control Panel layout are automatically saved and restored by the **Write Network** and **Read Network** functions under **Network Tools**.

Elements of a Layout

The user interface to a network consists of control widgets that are organized hierarchically:

- Individual **control widgets** (sometimes simply called **controls**) correspond to the input parameters of the modules in the network.
- A **page** is a window that contains one or more control widgets. The page construct allows you to see all of its control widgets at the same time.

  By default, all of a module's control widgets are assembled onto a single **page**.
- A **stack** is a window that contains one or more elements (typically, pages). The stack construct allows you to see just one element at a time. You can switch among them by clicking in the choice menu at the top of the stack window. (This menu is automatically created as you add elements to the stack.)

  The Network Control Panel window is, itself, a stack. AVS automatically assembles all the pages of control widgets for a network—one page for each module—into this stack.

Working with the Layout Editor

When you select **Layout Editor**, the following submenu appears:

Create Page
Creates a new, empty control panel page.

Create Stack
Creates a new, empty stack.

Undo
Undoes the effect of the most recent layout operation. Clicking **Undo** repeatedly will step back at most five actions.

This feature does not undo the creation of new pages and stacks. It does not undo creation of a Spaceball or Dialbox manager. It does not undo the effects of X window manager actions.

AVS creates files with names of the form */tmp/avs_undN.PID* to implement the undo feature. (*N* is a small integer and *PID* is the process number.) Don't delete these files during a Network Editor session if you want to use the undo feature. They are automatically deleted whenever you start AVS or perform a **Clear Network**.

Snap to Grid
Causes widgets to automatically align themselves according to a grid, making the resulting interface neater. The resolution of the grid squares defaults to 10x10 pixels. You can change this with the GridSize *.avsrc* startup file keyword.

Spaceball Manager
Creates a Spaceball Manager widget for the network. When the network executes, you use this widget to choose which module parameter is controlled by the spaceball.

Dialbox Manager
Creates a Dialbox Manager widget for the network. When the network executes, you use this widget to choose which module parameters are controlled by the eight dials on the dialbox device.

DialsmatrixMgr
Like the spaceball, it generates a single 4x4 matrix from all the dials and sends it to a single module paramter.

Click these choices to create additional places in which to organize the network's control widgets. Then, use the mouse buttons to rearrange the control widgets. To add a widget or page (or even another stack) to a stack, move it onto the stack's set of buttons; a new button appears for the newly-added item.

Note that in the Layout Editor, the mouse buttons modify the *layout* of control widgets, pages, and stacks rather than changing the *values* of parameters. Red borders around the control widgets, pages, and stacks remind you that you are in this Layout Editor mode.

You can resize pages to allow them to accommodate more control widgets. You can reorganize pages into one or more new stacks. (You can also place individual control widgets, or even other stacks, within a stack.)

As you move the mouse, elements whose layout can be changed are outlined in white. A simple white border means the window can be moved or deleted; a border with a series of "handles" (corner and side boxes) can be resized, as well.

While in the Layout Editor, most of the titles on pages and stacks become **type-in** widgets. You can edit the titles just as you would use any type-in.

You can move, resize, and delete control widgets as follows:

- **Left Button:** Move element. You can move any type of element—control widget, page, or stack. The destination can be elsewhere within the same page, to a different page or to the root window. In the latter case, the control widget becomes a top-level window, and can be manipulated using the X window manager.
- **Middle Button:** Resize a control panel or stack. (Not supported for individual control widgets—a question mark cursor appears.) Click and hold

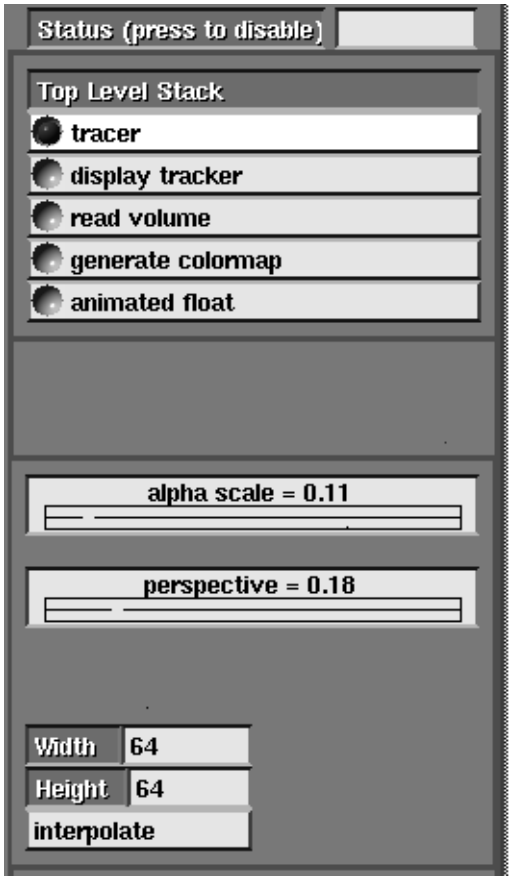


Figure 8-7 Window Borders in the Layout Editor

down the button, then drag the cursor through the edge or corner you want to move.

- **Right Button:** Pops up a menu appropriate to the element. The menu may include:
  - **Delete:** Delete the element. If you delete a control widget, you'll have no way to affect the value of the associated input parameter when the network executes. Deleting a page or stack effectively deletes all the control widgets it contains.

    If you did not mean to delete the element, select **Undo** immediately. You may also need to perform a **Reconfigure** (see below) to adjust the page size.

    To recover a control widget after it is too late for an **Undo**, you must invoke the Parameter Editor. In the Workspace, find the module whose parameter is associated with the deleted control widget. Click the small square button on the icon with the middle or right mouse button to open the Module Editor window. In the Parameter Editor section of this window, click on the desired parameter. This pops up a menu of choices (e.g. dial, slider, type-in) for the form in which the control widget is to be reinstated.
  - **Add Title:** Add a title box to a page (if one doesn't already exist). To edit the title, move the cursor into the title box, and type in a new ti-

tle. You'll probably want to start by using **Backspace** (delete last character) or **Ctrl-U** (delete entire title). Don't forget to press **Return** to finish the title.

The title box is itself an element that can be moved, deleted, or edited. It cannot, however, be resized or moved out of its original page.

- **Reconfigure:** Resize a page or stack to fit its contents.
- **Control widget type (radio buttons):** Change the type of control widget (e.g. change slider to dial). You can also change the type of a parameter's control widget using the Parameter Editor, as described above.

For example, to resize **print field**'s scrolling output browser, you would do the following:

1. Drag a **print field** module into the workspace and make it the currently-selected module by clicking on its menu button on the Network Editor Control panel.
2. Click on **Layout Editor** in the Network Editor main menu.
3. Place the mouse cursor inside the **print field** scrolling output window. The border of the window should turn grey.
4. Press the **left** mouse button and drag the output window all or partially outside of AVS control panel. The window will be "reparented" by your window manager.
5. With the cursor inside the output window, press the **middle** mouse button and drag it to the left or right to widen the window. (Don't resize the window with your window manager, use just the AVS middle mouse button.)
6. Position the output browser window anywhere that is convenient on the screen.
7. Leave the Layout Editor by clicking on either **Network Tools** or **Module Tools**.

Including Display Windows in a Reorganized Layout

You can include the *display windows* created by the **display image**, **display pixmap**, **image viewer**, **geometry viewer**, and **graph viewer** modules in a reorganized layout. Make sure that the page or stack into which you want to move the window is large enough. Then, move the window using the left mouse button, just as you would move any control widget. The display window is automatically subsumed under the page or stack, so that you can no longer manipulate it using the X window manager. If you subsequently move the display window out of its page or stack, it becomes a top-level X window again.

The **Zoom** function temporarily makes the window a top-level X window; **Unzoom** returns it to the page or stack whence it came.

In this way you can completely hide unneeded Network Editor functions, reduce the need for frequent window manager operations, and present a more traditional application interface to your end user.

Using the Spaceball and Dialbox Managers

Note: AVS now supports direct connection of the spaceball and dialbox to the Geometry Viewer's transformations through AVS command line options, *.avsrc* keywords, or environment variables without the necessity of using the Layout Editor. See the "Starting AVS" chapter.

Each network can have one Spaceball Manager widget and/or one Dialbox Manager widget. The first time you click its button in the **Layout Editor** submenu, the widget appears on its own page in the Network Control Panel. (Subsequent clicks on the same button are no-ops). Examples of these widgets are shown in the next two figures.

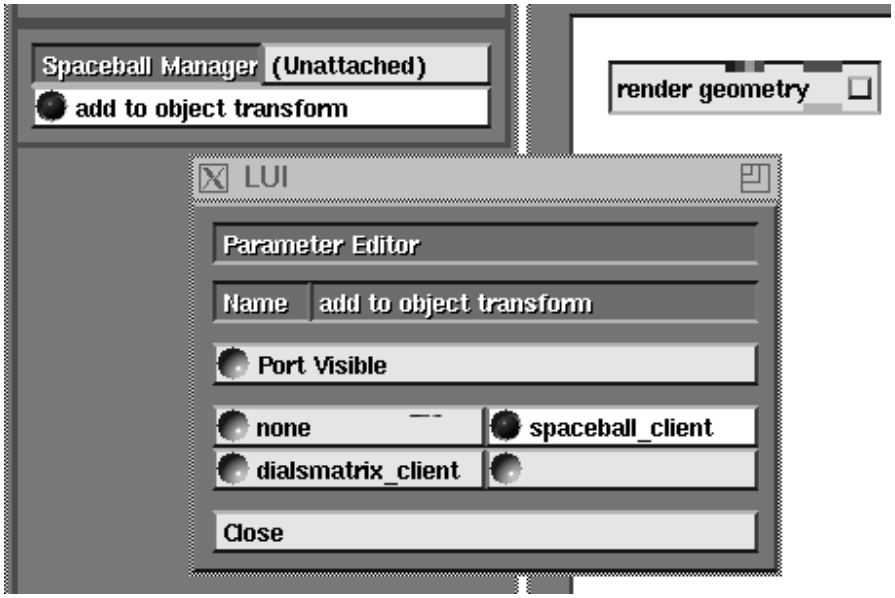


Figure 8-8 Spaceball Manager

AVS needs to know which serial ports the Spaceball and Dialbox are attached to. You can use the environment variables SPACEBALL and DIALS to specify the connections:

```
setenv SPACEBALL /dev/tty01
setenv DIALS     /dev/tty02
```

You can also use the **SpaceballDevice** and **DialDevice** settings in the AVS startup file (see the "Starting AVS" chapter).

When you click **Spaceball manager**, **Dials matrix manager**, or **Dialbox manager**, AVS attempts to locate the physical device, initializes it, and displays the name of the serial communications port at the top of the Manager widget. If

AVS cannot initialize the device (because it doesn't know where to look, because the device is not connected or is malfunctioning, etc.) it displays the string "(unattached)" instead. You can click on this word to bring up a dialog box that allows you to specify the device name at runtime.

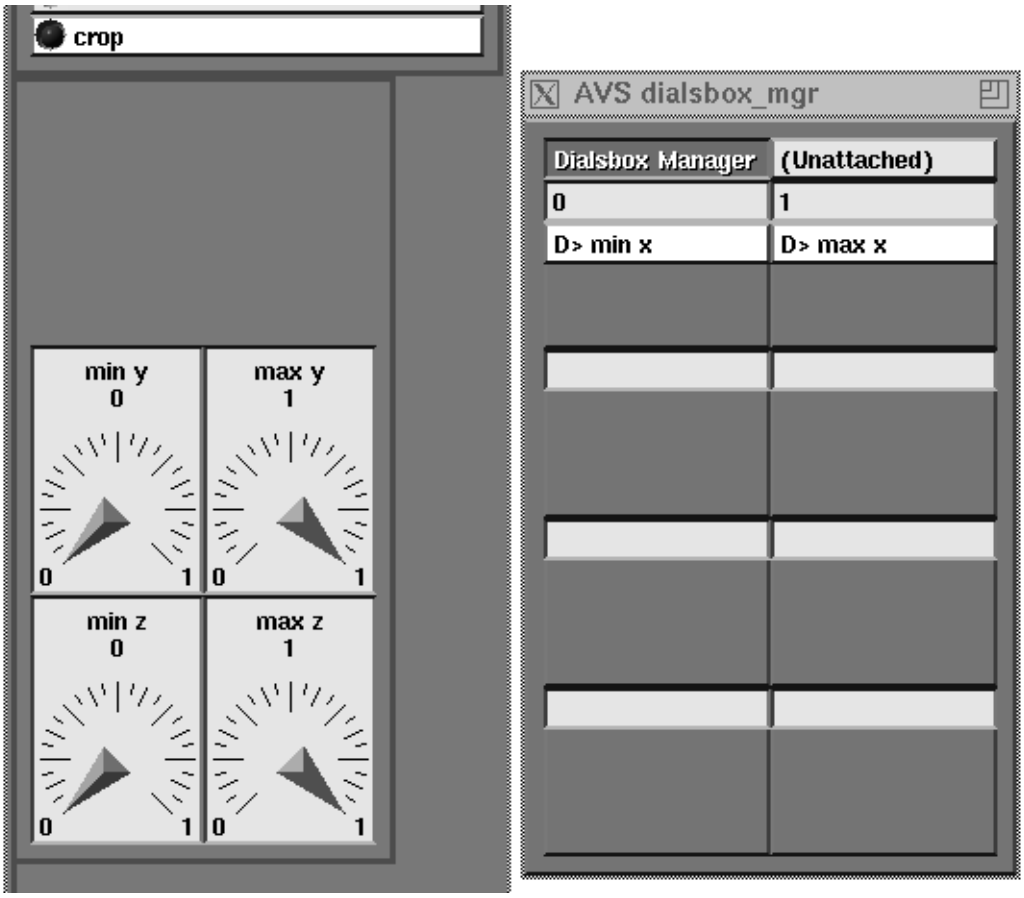


Figure 8-9 Dialbox Manager

Whenever the name of a serial port *is* displayed, you can click on it to detach the device. If you wish, you can then attach the device to another port.

Note: It is not necessary for a spaceball or dialbox to be attached to the system when you create a network. Just leave the the device "unattached".

Only one module parameter can be assigned to the spaceball at any one time. Similarly, each dialbox dial can control only one parameter at a time. For flexibility, the devices can be multiplexed:

- The creator of a network can associate several parameters with the spaceball or with a particular dialbox dial.
- When the network is executed, radio buttons on the spaceball or dialbox Manager allow you to select which of the several parameters is to be controlled by the device. (The radio buttons are active only when **Layout Editor** is *not* selected.)

The spaceball and dialbox differ in the way the creator of a network associates module parameters with them:

Associating module parameters with the Spaceball/Dials matrix

In the current network, find the icon for the parameter to be associated with the spaceball. Use the middle or right mouse button to click on the icon's small square, invoking the Module Editor. Then click on **Parameter Editor** to bring up a menu that lists the available control widgets for the parameter (dial, slider, type-in, etc.). If the menu for a parameter has a **Spaceball-client** choice, then the parameter can be assigned to the Spaceball; otherwise, it can't. Click on the **Spaceball** choice, then click on **Close** to close the menu.

Associating module parameters with the Dialbox

Use the left mouse button to move the parameter's control widget onto one of the dials in the Dialbox Manager widget. The control widget disappears and its name appears above the dialbox dial. You can move up to three control widgets onto each dialbox dial.

To recover the original control widget (that is, to disassociate it from the dialbox), grab the name with the left mouse button and move it outside the Dialbox Manager widget back into a page or stack. This always reassigns the parameter to a dial control widget. If necessary, use the right mouse button or invoke the Parameter Editor (see preceding item) to change the widget type.

When the device managers are deleted, the devices revert back to direct interaction with the Geometry Viewer.

Remote Module Execution

AVS supports the synchronous execution of AVS modules on remote AVS hosts. The remote host must be running AVS Release 3 or later. The network communication and data transfer mechanism between the AVS kernel/flow executive and the remote module are based on standard Unix TCP/IP network protocols. Data representation is based on Sun's External Data Representation (XDR). Thus, the remote host can be **heterogeneous**—of a hardware type other than type of machine the AVS kernel is executing on. The user interface to remote modules is smoothly integrated in the AVS Network Editor.

There are many situations where you might want to include one or more remote modules in an AVS network; these are a few typical cases:

- You have a compute-intensive module that runs best on a particular kind of hardware
- You have a module, perhaps a real time data collection application, that *only* runs on a particular kind of hardware.
- The data files your AVS network needs reside on a remote host's filesystem that is not easily accessible from your workstation; it would be easier

to read the files on the remote host, than to manually transfer the data to your workstation's filesystem domain.

- You wish to run modules in parallel—some executing on the local host; others on remote hosts.

In any given situation you must ascertain whether the benefits of remote heterogeneous module execution outweigh any data transfer overhead that might be introduced.

There are three aspects to remote module execution:

- What must be present on the *remote system*.
- The *hosts* file that AVS uses to access remote modules.
- The AVS Network Editor user interface to remote modules.

The exact interface to the remote system, particularly the command (*rsh* or *c_rsh* or some other variant) that is used to first establish contact with the remote host, may differ from system to system. This manual describes the general case that will usually work between Unix hosts on an "open" network that does not place barriers to a user going from one system to another. Users should consult the AVS documentation for the remote system.

Remote System

The remote module(s) must have been compiled and linked on the remote machine against version 3 or later of the standard AVS libraries. This implies that the remote host must be running AVS 3 or higher.

Note: Modules compiled and linked under AVS 2 cannot be executed remotely.

No additional libraries or calls are needed to make a module "into" a remote module.

Certain modules cannot execute remotely:

- The following "builtin" AVS modules are part of the AVS kernel, and therefore cannot be remote modules:

  geometry viewer
  generate colormap
  display image
  display pixmap
  graph viewer
  image viewer
  render geometry
  transform pixmap (not on all platforms)
  colormap manager
  image manager
  render manager

- Modules that write results to a */tmp* file if other parts of the system must read the */tmp* file. This is because the remote module and the AVS running on the workstation may not share the same filespace domain. Modules that do this include:

  print field
  compare field
  vbuffer

Some AVS systems may have versions of these modules without these restrictions.

The remote host must accept TCP/IP network communications through the "Berkeley" socket mechanism.

The remote host must have some way of accepting remote requests to execute programs on its system, such as the Berkeley Unix **rsh** (remote shell) command. (Your workstation must be able to produce these requests.)

Data is transferred between modules in device-independent XDR format. However, if the remote hardware has a 64 bit wordsize and the local workstation has a 32 bit wordsize, integers and real numbers past a certain size will lose significance when transferred.

The remote host does not need to share a filesystem domain with the local host.

Setting Up A Remote Module Directory

Together with AVS, the self-contained directory of modules described below is all that is required on the remote host side. There can be multiple sets of modules accessible on the remote host, each set in its own directory.

1. Make a directory on the remote host.
2. Place in it a collection of modules compiled on the remote host (or an identical system) and linked against AVS libraries. The format is the same as in any directory containing AVS modules—there can be one binary per module, or multiple modules in a single binary. (The Remote Module execution mechanism does **not** support reading remote ASCII module **library** files. However, one can have a "local" library, some or all of whose elements are remote modules. This is described later.)
3. The directory must contain an executable binary of the AVS module **list_dir**. **list_dir** is the "point of first contact" for the AVS kernel attempting to access remote modules. Specifically, it is a special module that lists the contents of the remote directory for the remote user and implements the Remote File Browser. **list_dir** is not in any module palette. A binary of this module can be usually be copied from a file named */usr/avs/avs_library/list_dir* on the remote host.

Finding Remote Modules: the hosts file

The AVS kernel uses a *hosts* file to find remote module directories. When you press **Read Remote Module(s)**, AVS looks for a *hosts* file in two places, in this order of precedence:

1. Your *.avsrc* startup file may contain a Hosts specification like the following:

   Hosts *full-file-specification*

   For example:

   ```
   Hosts /home/users/username/avsstuff/hosts
   ```

   or

   ```
   Hosts /hostname/ourproject/avs/utilities/remotehosts
   ```

   If such a specification exists, AVS will use the file named as a *hosts* file. The file must be findable from your local workstation.

   This mechanism lets you create and maintain your own list of remote module directories.

2. AVS uses the system default *hosts* file in */usr/avs/runtime/hosts*. It is expected that an AVS system administrator would maintain this file. The AVS product comes with a sample */usr/avs/runtime/hosts* file to use as a template.

hosts File Format

The *hosts* file is an ASCII file. Each line in the file has four columns of information. If the string occupying a column contains blanks, it must be enclosed in double quote marks ("/usr/ucb/rsh host -n"). Here is a sample:

```
mercury_std  "/usr/ucb/rsh mercury -n" /usr/avs/avs_library       /usr/avs/data
mercury_my   "/usr/ucb/rsh mercury -n" /usr/myproject/avs/modules /usr/myproject/avs/data
cruncher1    "/usr/avs/bin/c_rsh nexus"  /usr/avs/avs_library      /usr/avs/data
```

- The first column contains a logical name. This is the name that will appear in the Remote Host Browser described below. It merely identifies the line of instructions to execute. It makes it possible to have multiple sets of AVS modules and/or data on remote hosts that you can select among as the situation demands. You can also run a network on a different host by modifying the mapping of the logical name to the actual remote host.
- The second column is the full file specification of a program that will run a command on the remote host. This is where the specific remote host is actually identified.

  The obvious choice for this "run remote program" function in networks of Unix-type machines is *rsh* (remote shell). This program is usually found in /usr/ucb/rsh in your workstation's file system domain. **Be sure to add the -n** flag to *rsh*. It will prevent the remote process from grabbing terminal input, placing your AVS in the background where it will seem to "go dead."

  Many workstations will be on secure networks where the remote host (usually some central number cruncher) will not permit a simple *rsh* that just "shows up" from the network to execute. The remote host may re-

quire that you **authenticate** yourself first. Consult the remote host's documentation.

Because there are blanks in this column, enclose it with double quote marks.

- The third column is the directory on the remote host where the modules are kept. This string is part of the *rsh* command that AVS sends the remote machine. Thus, this file specification is in the file system domain that the **remote** machine understands. This may or may not be the same as your workstation file system domain.
- The fourth column is a default data directory on the remote machine. Many AVS modules such as **read field** use a file browser widget. This tells them what to use as the default directory for the file browser widget. Again, this file specification is in the file system domain that the **remote** machine understands. This may or may not be the same as your workstation file system domain.

The example shows three cases:

- The first is a standard Unix file specification, presumably on some powerful cycle server machine called mercury. The location of the modules and data specified are the usual standard AVS modules in the standard place.
- The second is also a Unix file specification for the cycle server mercury; but here the modules and data are kept in some private directory for a research project.
- The third example uses the *c_rsh* program to run modules on a remote system called nexus.

The first module that the AVS kernel directs the remote machine to execute is **list_dir**, which produces the directory listing of the Remote Module Browser. Once started, the **list_dir** module continues to execute until the AVS session ends.

Network Editor User Interface

The user interface to run remote modules is logically and smoothly integrated into the AVS Network Editor. The interaction is basically identical to reading and using modules from a local file system with **Read Module(s)**.

1. Start AVS and enter the Network Editor as usual.
2. Click on **Module Tools** on the main Network Editor menu.
3. The **Module Tools** submenu has a button: **Read Remote Module(s)**. Clicking on this button raises the **Remote Host Browser**.

The host names in the Browser window list are taken either from the *hosts* file specified by the **Hosts** line in your *.avsrc* startup file, or from the system */usr/avs/runtime/hosts* file.

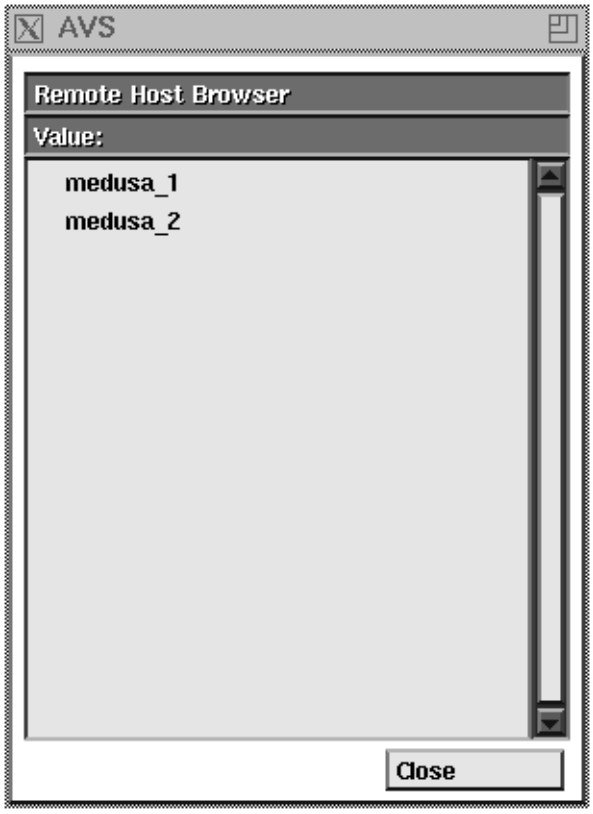


Figure 8-10 Remote Host Browser

The Remote Host Browser functions like any other AVS browser; click on a hostname to select it. Scrollbars scroll the list of hostnames. **Close** takes down the browser.

4. Once you click on a hostname, the Remote Host Browser is replaced with the **Remote Module Browser**. The names in the Remote Module Browser are the filenames of modules on the remote host.

   AVS reads the "run remote program" command for that host from the second Column of the *hosts* file, usually */usr/ucb/rsh*. It executes the remote program giving the hostname, remote module and file directory specifications (third and fourth column of *hosts*), and **list_dir** as parameters. It also passes the X Window System DISPLAY environment variable to the remote host so that it can make windows on your workstation. If the connection to the remote host was successful, and the remote directories were found, and **list_dir** was initiated, then the Remote Module Browser widget appears with the remote modules listed.

5. To load a remote module into the module palette, click on its name in the Remote File Browser. If the module has the same name as an existing module in the palette, it replaces it. If it has a new name, it is added to the module palette. The visual clue that this is a remote

module is the module icon's "dimple"—it will be pink instead of the usual gray.

Thereafter, one deals with the remote module like any other module: drag it into the workspace with the left mouse button, display its Module Editor by clicking the right mouse button on its dimple; turn its parameter ports on and off, connect it up to other modules in the network, click on its button, manipulate its widgets, run data through it, disconnect and destroy ("hammer") it just like any other module.

Only two differences may be noticeable:

- Remote modules with File Browser widgets, such as a remote **read field**, have Remote File Browser widgets. (These communicate with the remote **list_dir** module.) Remote File Browser widgets navigate in the filespace domain **of the remote host**.

  You can use the Remote File Browser's **New Dir** or **New File** button to select a remote directory.
- Remote modules that are not part of the standard AVS module may not have online man pages associated with them that are accessible through the Module Editor's **Show Module Documentation** button.

Otherwise the interaction is the same.

If you save a network that contains remote modules, remote modules in the directory specified in the hosts file are saved with the string **$RemMods** prepended to the module's name. When it comes time to read the network in again, **$RemMods** will be replaced with the module directory for the destination host. This makes it possible to define networks containing both local and remote modules that can be initiated independent of any specific remote host. The network should be read in correctly provided that the **Read Network** function is issued in an environment where the logical host name can be interpreted by a *hosts* file.

If you save an ASCII module **library** containing some (or all) remote modules, it too should work correctly when it is read in again, providing that the **Read Module Library** function is issued in an environment where the logical name can be interpreted by a *hosts* file. You can have such libraries loaded automatically when you start AVS by adding the library's filename to your *.avsrc* startup file.

As noted earlier, this release of AVS does not support reading or writing *remote* module *libraries*—that is, libraries of modules whose ASCII description file exists on a remote machine.

Constructing a Module Library

You may find that the collection of modules in the AVS supported and unsupported module libraries is not exactly what you'd like the Palette to contain:

- There may be modules that you never use, and would like to remove.
- You may want to add some modules that you've written.
- You may want to organize the modules into subsets (perhaps overlapping) that contain modules for a specific type of visualization, a particular data type, etc.
- You may want to change the names of the categories: **Data Input, Filters, Mappers, Data Output**.

There are two approaches to constructing your own module libraries:

- Module libraries are defined by ASCII text files. You can construct a module library "source" made up of library **file** commands followed by module binary filenames using any text editor; then "compile" the library using the *avs* command line option **-compile_library**. This will automatically generate the module description text.
- You can use the **Module Tools** menu's **Edit Module Library** function to interactively add and delete modules from a module library. This interactive editing does not support direct manipulation "moving" of modules from one Palette to another.

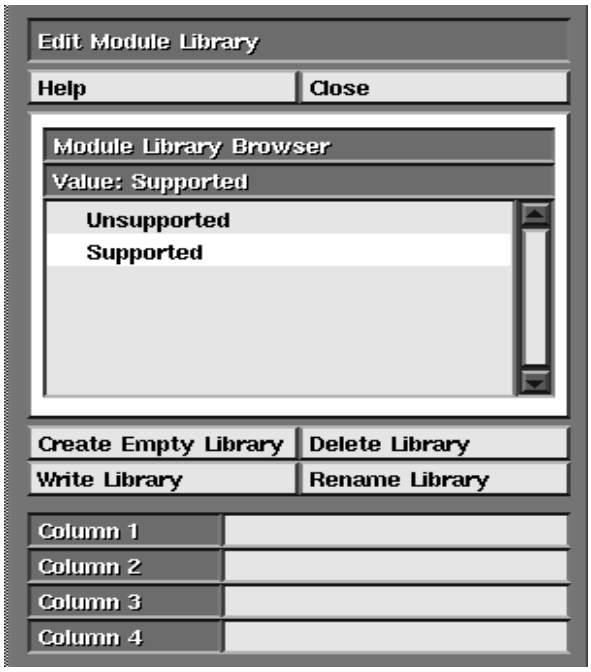


Figure 8-11 Edit Module Library Panel

Sample Interactive Procedure

Suppose you are an instructor teaching a class on the behavior of particles in vector fields. You are going to use AVS for your students' labwork. The default Supported module library in the AVS palette contains far more modules

than are pertinent to the subject. You've also written a few data input and visualization modules that you want to add to the students' module palette.

1. Start AVS and enter the Network Editor with the default Supported module library in the Palette.
2. Press **Module Tools** on the Network Editor menu.
3. Press **Edit Module Library** on the new submenu. This brings up the Edit Module Library panel:
4. Now start deleting the modules you don't want:
   - Position the mouse cursor over the module's icon in the module palette.
   - Press and hold down the left mouse button.
   - Drag the module icon into the work space all the way down until it is over the hammer icon and release the mouse button.
   - The module is deleted from the palette.

   If you release the mouse button before the module icon is over the hammer, then the module is not deleted from the palette. You need to hammer away the instanced version of the module and repeat the operation again from the start.

   There is no module editing "mode". It is possible to inadvertently delete modules from the Palette if you accidently drag a module icon all the way to the hammer without releasing the mouse button. If this happens, you can read in a new copy of the module with the **Read Module(s)** button.

   In truth, it may be faster to do this initial large-scale deletion using a text editor on a copy of the actual library file.
5. Now begin adding in your own modules. Press **Read Modules** and individually type in the file specification for each module's binary file. They will be added to the currently selected module library and will appear on the Palette.

   If your module binary files have multiple modules in each file, you can delete the individual elements that you don't want interactively as described above.
6. Next, press **Rename Library** on the Edit Module Library panel and typein a new library name. This changes the string that is displayed at the top of the Palette from "Supported" to your own name.
7. You can change the names of the columns by typing in strings into the spaces next to the **Column 1**, **Column 2**, **Column 3**, **Column 4** labels.
8. You can move modules from one category to another by bringing up the **Module Editor** window (click with the middle mouse button on the Palette modules' dimple), then use the **Change Module Category** menu (see the "The Module Editor and Parameter Editor Windows" section for more information).
9. The final step is to use the Edit Module Library panel's **Write Library** function to create the ASCII library file version of what is in the Palette to disk. Module library files do not have a particular file suffix.

You will have to exit AVS and re-enter it before your new library will show in the **Read Module Library** browser.

One case is not explicitly covered here: suppose there is one module in the Unsupported library that you want in your own module palette? There is no direct manipulation way to move modules from one palette to another as there is to delete modules from a palette. Instead, use **Read Module** to read the *binary* form of the unsupported module found in the directory */usr/avs/unsupp_mods*.

The Edit Module Library panel's **Create Module Library** functions creates a new, empty library in the Palette area. The new library is not written to disk until you enter **Write Library**. The **Delete Library** function deletes the library from the Palette, not from disk.

Lastly, you should give your students a *.avsrc* or *.avsrc.X* file with a **ModuleLibraries** line in it that will load your module library automatically when they start AVS. To be the default library that shows in the Network Editor module Palette, your library file specification must be the last listed on the **ModuleLibraries** line.

Compiled Module Libraries

In AVS, module libraries usually provide static snapshots of the descriptions of a set of modules for fast startup of the network editor. Although convenient, these libraries can easily get out of date when individual modules undergo changes. Module libraries can now be "compiled" from lists of module executables; AVS will read in the specified executable files or directories, and builtins and produce a static version of the module libraries. The source version of the library is itself a valid module library but one which would require waiting for all the specified modules to identify themselves to AVS for use. This feature allows an easily updated source version of the module library to be "made" into a compiled library as part of a normal Makefile whenever one of the source modules is updated to ensure an accurate module library.

For example, a source library such as

```
# AVS Module Library Version: 1
file field_legend
```

would produce a compiled library with the entry

```
# AVS Module Library Version: 1
external "field legend" 0 "field_legend" 3 2 "Input" 65 "field"
        "Input Colormap" 1 "colormap" 1 "value out" 0 "real" 4 "node data"
        "choice" "value" "real" "lo value" "real" "hi value" "real"
```

The command to compile a library is:

```
avs -compile_library source_filename compiled_filename
```

1. The source library consists of **builtin** lines, **file** commands and any other normal module library entries. Currently the source files just use builtins and files. During compilation the files are read synchronously (we wait on each one) and the resulting library is written out. The DISPLAY variable does not need to be set. (See the next section on module library file format.)
2. An executable can be deleted from the compiled library using the module library **delete** command. If there were a combined executable read into a compiled library that brought in an undesired module named foo, use the following line to remove it from the compiled library:

   ```
   delete foo
   ```
3. The compiled library is located in the same directory as the module executables if relative pathnames are desired (no leading directory name).

Module Library File Format

You can look at the AVS-supplied module library in */usr/avs/avs_library/Supported* to see what a library file looks like. AVS itself uses a somewhat more complex file format for the module library file it creates when you select the **Write Library** function on the Edit Module Library panel. The more complex format allows the Network Editor to load the module library more quickly. The format encapsulates the information created by the module's *AVSmodule_from_desc* call.

ID Line

The module library file *must* begin with this line:

```
# AVS Module Library Version: <n>
```

The version number <n> indicates the version of the module library file that you are creating. If you are copying commands from an automatically generated module library file, make sure to copy the version number from that file as well. Other lines that begin with **#** comment character may precede this line.

Command Lines

Each subsequent line must begin in one of these forms:

builtin "*module_name*"
external "*module_name*" *flags filename* <additional info>
remote host "*module_name*" *flags filename* <additional info>
macro "*module_name*" *flags filename* <additional info>
file *filename*
directory *dirname*
name *module library name*
title *column column name*
delete *module_name*

The **builtin** keyword specifies a module that is built into AVS itself, rather than being implemented as a separate executable file. These are the current supported builtin modules:

generate colormap
geometry viewer
display image
display pixmap
graph viewer
image viewer
render geometry
transform pixmap

The **external** keyword means that the modules is not builtin. **external** expects the detailed module description information created by **Write Library** to be present in the library file. It is not advised that you type this information in by hand. Rather copy the external line(s) from a module library file created with the **Write Library** function.

remote is the same as **external** except a logical host name is specified. It is used to locate modules that will execute on remote hosts in the network.

macro is the same as **external** except the module is created as a macro module.

The **file** keyword specifies an executable file that includes one or more module definitions. **file** is the same as **external**, except that it directs the Network Editor to read the binary file *filename* and, from that, construct the detailed description information. This is much slower for the Network Editor. It is intended instead to be used in source module library files that are converted to compiled module library files with the *avs* command line option **-compile_library**.

The **directory** keyword specifies a directory that includes executable module definition files. Like **file**, it makes the Network Editor open each file to discover the description information, rather than finding it in the library file. This can be very slow for large directories.

The **name** keyword specifies that the next argument should be used as a name for the module library. By default, the filename is used as the name.

The **title** keyword modifies the categories of the section headings (i.e. Data Input, Filters etc.) The **column** argument is the column number (between 1 and 4), and the **column name** argument specifies the new name for that category.

In each case, if the *filename* or *directory* is a simple file name or directory name without a path, then AVS assumes that it resides in the same directory as the library file. Otherwise, it expects a complete path.

The description information is used to accurately draw the module icon's name and input and output ports in the module Palette, and provides the information you see when you bring up the Module Editor panel for the module by clicking on its dimple to fill in the panel's information. It is only

relevant before a module is dragged into the workspace. It is not practical to type in this information yourself. You should let the **Write Library** function do it for you.

Optimization: Parallel Module Execution

AVS has the capability to run modules in parallel when the opportunity to do so is encountered in a network. The user specifies the maximum number of modules that should be run at any given time using the *-parallel n* command line option. The number *n* might correspond to the number of processors or hosts available to the application. When there are parallel branches in a network (a module sends its output to more than one downstream module), the downstream modules may be run in parallel.

There are a few conditions that can inhibit running modules in parallel:

- The maximum number of parallel modules has already been met, as set by the *-parallel n* option.
- The two modules are in the same process. Note that most supplied AVS modules are compiled together in one binary and execute in one process. Thus, one must usually expect to have to divide them into separate processes before they will execute in parallel.
- The module to be run in parallel has an input that is connected to an output of the running module, either directly or indirectly through a series of other modules.

To control which modules are put in the same process when editing a network, you can use the Module Editor panel's process id printout and **Group** typein. This is particularly useful for ensuring that two potentially parallel modules are not placed in the same process. See the "Optimization: Controlling Module Groups and Processes" section below for information.

Alternately, you can force all modules to run in separate processes using the **-separate** command line option.

The "Multiple Module Processes in AVS" section of the *Developer's Guide* "AVS Overview" chapter describes which AVS modules are grouped together into single processes by default.

Optimization: Adaptive Block Tables

The **AVS_ADAPT_TABLE** *switch* environment variable causes AVS to use adaptive block tables. Setting **AVS_ADAPT_TABLE 1** can speed the execution of networks processing irregular fields. **AVS_ADAPT_TABLE** is off (**0**) by default.

A block table is a data structure that maps field points' I, J, K indicies in an irregular field within a "block" of X, Y, Z world space. Modules such as **arbi-**

trary slicer and **probe** use the block table to interpolate values at points "on" their sampling surface, determining which need to be mapped as colored polygons.

AVS normally builds a regular, evenly-dimensioned block table. Where data points are fairly uniformly spaced within the field, such a block table provides efficient access to the I, J, K values in each block of the grid—each block has approximately the same number of points. However, where data values are concentrated in some areas of the field, but sparse elsewhere (e.g., the wing surface of the *bluntfin.fld* dataset) search times in the dense blocks become much longer.

An adaptive block table creates the block table as an octree. Where data values are dense, the block grid is divided and subdivided again until each block contains only a short list of I, J, K values to search through, improving performance.

Adaptive block tables are slower to construct, but execute more rapidly in the areas with dense grids. People with irregular datasets where the distribution of data points is uneven should try setting **AVS_ADAPT_TABLE 1** to see if it improves the performance of the **arbitrary slicer**, **threshold slicer**, **streamline**, **particle advector**, **hedgehog**, **probe**, and **color geom** modules.

Optimization: Controlling Module Groups and Processes

Modules have an attribute called the "module group" that can be used to control whether two modules are placed in the same process or not. This feature is useful for advanced network editing operations to:

- Increase the parallelism of a network by ensuring that parallel fragments of a network are not placed in the same process. Two modules that are in the same process cannot be run in parallel.
- Increase the efficiency of executing a particular network by ensuring that modules that are connected are placed in the same process so that they can share data if inter-process shared memory is not available for some reason. (The "AVS Overview" chapter of the *AVS Developer's Guide* has a diagram that shows AVS's various strategies for minimizing data memory and copying requirements when modules are in the same or different processes, and when shared memory is or is not available. The "Multiple Module Processes in AVS" section of the save *Developer's Guide* chapter describes which AVS modules are in which binaries.)

The module group attribute is an ASCII string that the user can define for a particular module by typing into the Module Editor panel of a particular module. When this string is the same for two modules, they share the same module group and can be placed in the same process when the module is created.

The module group can only be used to inhibit two modules from being placed in the same process but by inhibiting a module from sharing another mod-

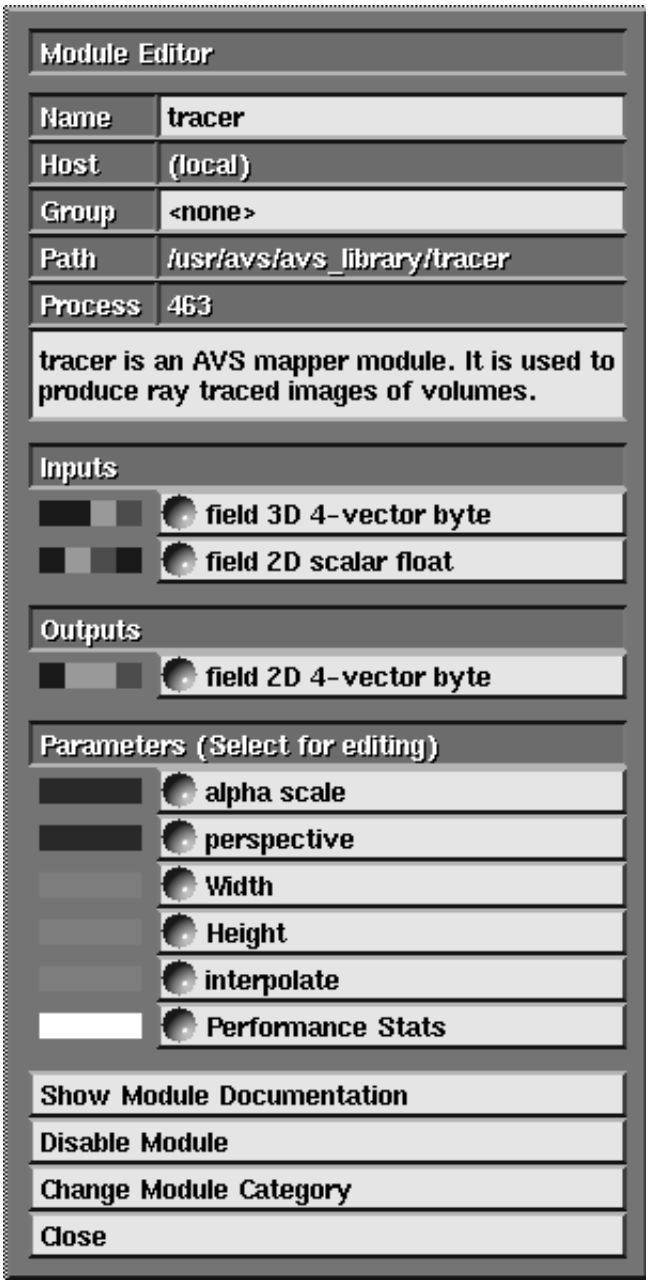


Figure 8-12 Module Editor Panel

ule's process, it opens up an opportunity to share the process with a third module.

Here is an example. We have three modules "module A", "module B" and "module C". All three modules are contained in the same executable, all were originally declared to be **REENTRANT** and **COOPERATIVE** in their module source. (See the "Advanced Topics" chapter of the *AVS Developer's Guide* for a definition of these terms.) We want module A and module B to be run in par-

allel so they cannot be in the same process. But, we want module A and module C to be in the same process to reduce the startup time of the network. We can do this most concisely by defining a module group for module B when the module is still in the module Palette. Now, when we instance module A and module C, they will be placed in the same process because neither module has a defined module group string. When we instance module B, however, its module group won't match the non-existant group of module A, and so it will start a new process. Module A and module B can now be run in parallel if the network is constructed in a way that allows parallel execution.

There are other situations where defining the module group may make sense. Some systems do not support shared memory communication of data between two processes. On these systems, when two modules are connected it is necessary for AVS to make an extra copy of the data set in order for the downstream module to have access to it—unless the two modules are in the same process. A module that is **COOPERATIVE** but not **REENTRANT** can only have one module instance in a particular process. We can therefore come up with a situation in which the system will make an arbitrary decision when grouping modules into processes. The user can use the module group to impose an order on this grouping.

The module group can be modified for either an inactive module in the Palette or for an active module that has been instanced in the Network Editor Workspace. The module group attribute is only referenced, though, when you are creating a new module (dragging it into the Workspace). Changing the module group of an active module does not rearrange modules in processes but will potentially affect how a new module is started.

The module group attribute is properly saved and retrieved when you save away a network.

Note: There is a CLI **net_group** command that supports a different, simpler function: the lassoing of modules to be moved as a group. The **net_group** CLI command is irrelevant to module grouping as just described. Module process grouping is controlled by a **-group** option to the **module** CLI command.

APPENDIX A

AVS ON COLOR X SERVERS

Introduction: Renderers

It may be possible for you to run AVS as a remote X Window System client on any system in your network that runs AVS. Your own display hardware need only be a color workstation or X terminal that meets some minimum requirements.

AVS's graphics functionality is always implemented on top of an underlying graphics subsystem. Whether you will be able to run AVS as a remote X client depends upon whether there is a common ground between:

- The graphics protocol your display hardware understands.
- The graphics protocol the remote AVS host produces.

There are two "graphics protocols" to choose from:

1. The X window system and AVS's software renderer.
2. The platform's native graphics subsystem and AVS's accompanying hardware renderer.

At this date, all major workstations and X terminals use the X Window System to control their display. The AVS interface, the Image and Graph Viewers, the Network Editor, and all applications use the X Window System to produce their screen output. There is thus no problem with generating those aspects of AVS on a remote host for display on a local host.

The AVS **software renderer**, used by the Geometry Viewer, also produces an X image as its final output. This X image can be shipped to the local color display for viewing on the screen. Thus, provided your local display hardware meets the minimum requirements and you specify that you wish to use the software renderer only (**avs -nohw**), you should always be able to run AVS as a remote X client with the software renderer.

Using remote hardware 3D rendering in the Geometry Viewer is more problematic. In this case, the hardware renderer is producing a stream of graphics protocol commands. Most graphics protocols assume a local display. However, some, such as PEX (PHIGS Extension to X) and distributed GL, are capable of sending their graphics protocol stream to a remote display. That remote display must, of course, understand the protocol stream it is receiving and be able to translate it into the correct picture on the screen.

You must consult your platform's AVS release notes to discover how hardware rendering in your AVS is implemented, and what kinds of display devices can communicate with it remotely.

The remainder of this Appendix discusses the minimum requirements necessary to run AVS as a remote X client from a workstation or X terminal display. It then gives pointers for using AVS efficiently in this configuration that are relevant on a wide variety of local display / remote host combinations.

Minimum Requirements

AVS can be run as a remote X Window System client from any **color** display hardware that meets the following *minimum* requirements:

1. It runs an X server at version X11R2 or later.
2. It has an 8-bit per pixel (8-plane) *or greater* **color** frame buffer.
3. Its X server has an X Window System "visual" defined that supports *one* the following:

   8 bit PseudoColor (minimum)
   12 bit PseudoColor
   16 bit TrueColor
   24 bit TrueColor

   DirectColor is also supported. Monochrome is not supported.
4. This visual must be the *default* visual. AVS normally only communicates with an X server's default visual. If it is not, you must use the **VisualType** startup file keyword to redesignate the visual that AVS will use. (It is also possible to change the default visual when the X server is started. See your system's X server documentation.)
5. It has at least a 256 element colormap.
6. Its screen resolution is a minimum of about 1024x768.

Standard X systems support the command **xdpyinfo** which causes the server to print out a list of its capabilities. You can use this list to determine if your display hardware can use AVS as a remote X client. (If your system does not support the **xdpyinfo** command, then remote login to a machine that does have **xdpyinfo**, set the DISPLAY environment variable on the remote host to point back to your local display, then issue the **xdpyinfo** command. You should get the X server report.)

The following is shortened version of an **xdpyinfo** report issued on a display that meets the minimum requirements. The numbers at the side show which of the numbered requirements listed above is being reported on.

```
name of display:    mercury:0.0
version number:    11.0                          <------- 1
vendor string:    Hardware Computer Inc.
   .
   .
   .
screen #0:
```

```
dimensions:    1024x780 pixels                <------- 6
resolution:    80x80 dots per inch
 .
 .
 .
default number of colormap cells:    256      <------- 5
 .
 .
 .
number of visuals:    3
default visual id:  0x80067                   <------- 4
visual:
  visual id:    0x80067                       <------- 4
  class:    PseudoColor                       <------- 3
  depth:    8 planes                          <------- 3
  size of colormap:    256 entries            <------- 5
  red, green, blue masks:    0x0, 0x0, 0x0
  significant bits in color specification:    8 bits
visual:
  visual id:    0x80066
  class:
  depth:
 .
 .
 .
```

Of course, you can always just try it and see if it works.

Overall Performance

AVS was originally implemented to run on very high-performance, true color (24 plane) workstations with considerable hardware support for 3D graphics rendering, including hardware gouraud shading, lighting, texture mapping, and hardware-assisted polygon and sphere rendering and transformations.

When you run AVS on a 3D graphics system, but display on a generic X server running on another piece of hardware, you will be missing some or *all* of these hardware performance accelerators and some color rendering abilities. The standard X protocol that AVS might use to communicate with the remote X server does not transmit the information that would be needed for the X server to take advantage of any graphics hardware support, even if it is present on your local workstation. All that is sent with the software renderer is an X image.

If the protocol between the client and server is based on PEX, or some other standard, more of the local display's graphics power may be used.

On pseudocolor systems, color rendition will be more limited in images and pixmaps. Often, instead of smooth gradations in shade, you might see several bands of colors.

AVS will not be rendering data "colorized" with **generate colormap** in true color. Instead, it must map the values in **generate colormap**'s colormap into the 212 color slots (on 8 plane devices) of the 256 element hardware colormap on the workstation. The **generate colormap** Colormap Editor widget will show you an accurate picture of the colors that will be used for data values.

In short, this configuration preserves most of AVS's functionality, but has a more limited color palette, and it is likely to be comparatively slow for 3D rendering.

Set Up: Startup File Keywords, and Environment Variables

You should define the following Unix C shell environment variable, either interactively or in one of the files executed automatically at login or shell startup:

```
setenv  DISPLAYCLASS  X
```

Alternatively, invoke AVS with an additional option:

```
avs  -class X
```

Both commands are *case sensitive*.

This has two effects described below.

.avsrc.X Startup File

With *-class X* specified on the command line, or *setenv DISPLAYCLASS X* set in your environment, AVS will first look for a file in */usr/avs/runtime* called *avsrc.X*. The released version of AVS does not have such a file. Not finding it, it will execute */usr/avs/runtime/avsrc*. Next, AVS will look for a file in your HOME directory called *.avsrc.X*. This AVS startup file can be a copy of your usual *.avsrc* file, but with the few extra lines described below that customize the session for the remote client/server configuration. If there is no *.avsrc.X* file, it uses the regular *.avsrc* file in your HOME directory.

If DISPLAYCLASS or **-class** are set to something besides "X", say "myhardware", AVS will look for a *.avsrc.myhardware* file instead.

Some systems preset DISPLAYCLASS to another value. Check for this and make sure your environment sets it to **X**, overriding the system default if necessary.

Display Brightness: Xdefaults.X File and Gamma Keyword

Some machines' displays are "gamma-corrected". The AVS user interface color scheme, which looks fine on these gamma-corrected displays, will usually appear much too dark and "muddy" on the X display. The control panels will look almost black instead of medium grey, and button labels may be unreadable. If you are using a gamma-corrected system to run the AVS client, defining DISPLAYCLASS X will cause AVS to use a file called */usr/avs/runtime/Xdefaults.X* to define an initial interface color scheme that is balanced properly for most X display hardware.

It is not possible to have a "personal" Xdefaults file in your HOME directory. The best alternative is to get your system administrator to install an Xdefaults.*yourclass* file in */usr/avs/runtime* and set your DISPLAYCLASS environment variable to match *yourclass*.

Alternatively, you can use the **-gamma *n*** command line option or the **GAMMA *n*** keyword in your *.avsrc.X* file to uniformly brighten the display. Higher real values for ***n*** produce a lighter display.

VisualType

AVS normally uses an X server's default visual. If the default visual (as shown by the **xdpyinfo** command example above, reference note 4) on your X server is not the highest-power visual suitable for AVS that your hardware is actually capable of, you have two choices.

- Change the default visual that the X server uses when the X server itself starts using a server option. Consult your system's X Window System documentation for the correct way to do this.
- Use the AVS startup file option **VisualType** to select a different visual. You would normally define this in your *.avsrc.X* file rather than your usual *.avsrc* file.

VisualType has four choices for parameters. You may use one of the strings: **PseudoColor** , **TrueColor**, or **DirectColor**. In this case, AVS will search down the server's list of visuals looking for the *first* visual with a "class" that matches.

On some systems, the first PsuedoColor, TrueColor, or DirectColor visual may not be the most powerful available. In this case, specify an exact visual id as follows:

```
VisualType VisualID n
```

Where *n* is the decimal equivalent of the X server's hexadecimal visual id for the visual you want to use (line 4 in the **xdpyinfo** sample printout above).

Note: Some systems may not support the **VisualType** option. See your vendor's AVS release notes for details.

BoundingBox and Freeze Camera

Bounding Box is a necessity of life when using AVS on an X terminal.

"Real time" rendering of moving 3D objects (and possibly Image Viewer subimages) is very compute-intensive. It is usually not possible to rotate or move a 3D object, light, or camera in the Geometry Viewer with the middle mouse button in "real time" on an X server the same way that you are accustomed to doing on a high performance graphics system. Depending upon the complexity of the object or scene, the X terminal may not be able to keep up with the volume of "real time" rendering information coming from the remote AVS host.

Bounding Box is a toggle switch found in the AVS Image Viewer and Geometry Viewer below the Transformation Options menu. It disables this compute intensive real time rendering. With **Bounding Box** turned on, when you place the mouse cursor over the current transformable and press the middle or right button, a wireframe box enclosing the volume of the object appears. As you move the mouse, the bounding wireframe box moves—the object does not. You move the bounding box to the destination position / rotation / scale, then let go of the mouse button. Only then is the object rendered at its new location and orientation.

You can either remember to turn on **Bounding Box** each time you run AVS, or you can add this line to your *.avsrc.X* file:

```
BoundingBox    1
```

Again, match the case of the letters exactly.

You can further reduce re-rendering in the Geometry Viewer by switching on **Freeze Camera**. See the **Freeze Camera** discussion in the "Geometry Viewer Subsystem" chapter of the *AVS User's Guide.*

Changing the Entire Interface Size

If your display is smaller than the normal 1280 x 1024, you can cause AVS to run in a virtual screen size smaller or larger than the actual screen size with the **-size** command line option or the **ScreenSize** startup file keyword. The format of the option is:

```
ScreenSize   XDIMxYDIM
```

Where *XDIM* and *YDIM* are the size, in pixels, of the virtual screen.

You can adjust the size of the Network Editor's Network Construction Window, which includes the Network Editor menu, the Module Palette, and the Workspace. Use the **NetworkWindow** *.avsrc* keyword as follows:

```
NetworkWindow 850x850+250+250
```

The specification is an X Window System Xgeometry.

Colors: Colormap Cell Allocation

AVS will use the system's colormap to allocate its colors. How many color cells it will allocate to itself depends upon how many planes of pseudo color or true color the X visual calls for. The lowest-usable number of planes is 8-plane pseudo color. In this case, AVS will try to allocate 242 cells: 6 red x 6 blue x 6 green, plus 26 grey tone cells (6 6 6 26). If this is unacceptable for some reason, perhaps because it takes too many cells away from other applications, you can modify the default color allocation with the **Colors** startup file keyword. **Colors** takes four numeric parameters separated by spaces that specify how many red, green, blue, and gray cells to try to allocate:

```
Colors red green blue gray
```

Colors can also be used to establish a wider grayscale for monochrome images. Increase the gray value while decreasing the red, green, and blue values. For example:

```
Colors 5 5 5 100
```

would produce 125 color cells and 100 gray tones.

Starting AVS from a Remote X Server

AVS will be running as a remote X client. The following lines show the usual procedure for running remote X clients on a remote Unix system. Check with your local system administrator to see if there are differences in this procedure at your site.

```
login to your localhost
   .
   .
   .
localhost% xhost +avshost            <--gives permission to remote
   .                                      host to make windows on
   .                                      your display
   .
localhost% rlogin avshost            <--login to the remote host
   .
   .
   .
avshost% setenv DISPLAY localhost:n.m <--on the remote system,
                                         set DISPLAY to point back
```

```
                                    to your display, e.g.,
                                    mercury:0.0
```

If you are using an actual "X terminal" that is connected to the AVS system, you will only need to make sure that your DISPLAY environment variable is set correctly.

Software Renderer

If you will be using the software renderer for 3D graphics rendering in the Geometry Viewer, start AVS as follows:

```
avshost%  avs -nohw
```

Alternatively, add this line to your *.avsrc.X* file:

```
NoHW 1
```

Without this specification, whichever hardware renderer is present on the remote platform will attempt to establish communication with the local display. Unless this is one of the few cases (PEX, distributed GL, etc.) where this actually works, AVS will likely fail to come up properly.

Interaction

For the most part, AVS behaves and looks the same on X servers as it does running on a local AVS host. The differences are noted below by subsystem.

Image Viewer

All Image Viewer functions should work. If you are going to be doing image processing techniques on a subimage area, turn on the **Bounding Box** option.

The main difference between the Image Viewer running on a true color system versus running on a pseudo color X server is the appearance of the images.

On 24 plane true color systems, each pixel can have one of (2**8)**3 (16,777,216) color values. There are 8 bits to represent red tones, 8 bits for green tones, and 8 bits for blue tones. The red, green, and blue tones combine to create the actual pixel color. The sample image files in */usr/avs/data/image* are all true color images.

On 8 plane pseudo color systems, in AVS each pixel can normally have one of 216 color values. There are 6 red tones, 6 green tones, and 6 blue tones. To display a true color image on an 8 plane pseudo color device, AVS takes the original red value for each pixel and finds the closest numeric value from among

the 6 reds available. It does the same for green and blue. The final pixel color is the combination of these three best-matches. This might sound very limited, but the end result is surprisingly satisfactory.

Pseudo color and true color displays with more planes use the same method to produce a closer approximation to the original 24 plane true color representation.

On pseudo color devices, the Image Viewer's **Image** menu will contain five additional buttons that will allow you to control how the images in the output window are **dithered** for display. Dithering means that AVS uses an algorithm to adjust the actual pixel values arrived at above by small amounts to create the impression of greater color range than is actually present. This reduces the "banding" you see when a colors change shade slowly over a wide screen area and produces a closer approximation to a true color image. There are several dithering alternatives (including "none") that produce different effects.

The **display image** module has these same five dithering controls on pseudo color devices.

Graph Viewer

There are no significant behavior differences in the Graph Viewer.

Geometry Viewer

Most significant differences between AVS running on a high-end graphics workstation and AVS running on a lower-end machine appear in the 3D graphics rendering-intensive operations of the Geometry Viewer.

In general, you should:

- Use the **Bounding Box** control, with **Freeze Camera**,
- *Wait* for operations to complete before pressing more buttons or making more transformations to avoid creating a huge backlog of operations,
- Avoid exposing the **geometry viewer**'s display window unnecessarily. This will trigger a refresh of the entire window. With complex objects having many polygon surfaces, this can take time.
- When using the software renderer, do *not* select **Polygonal Spheres** under the **Cameras** menu. Rather, allow the software renderer to use its true sphere rendering emulation. This saves both time and memory.

  Where **Polygonal Spheres** are desired, be sure to use the **Spheres Subdivision** control at the bottom of the Geometry Viewer **Objects** menu. To make an object look spherical requires *many* polygon surfaces, which are slow to render. The **Subdivision** slider bar controls how many polygons

to use to represent a sphere. A small number, such as 1, will produce an 8-polygon diamond instead of a sphere, which renders quickly.

These recommendations are actually no different than what you can do to improve performance whenever you are using the software renderer.

Network Editor

Except for the color-coding on module icon ports being less subtle, there should be no differences in the behavior of the Network Editor.

You should look at the **generate colormap** Colormap Editor panel for an accurate picture of what colors will be used to "colorize" your data. Whatever colormap values are produced by **generate colormap**, they must be mapped into one of 216 fixed color values in the hardware colormap.

If you turn on the **Disable Flow Execute** function under the **Network Tools** menu, AVS will not re-execute a network every time you change a parameter. This makes it possible for you to arrange everything that you want to have happen in one pass without the network executing until you deliberately tell it to.

In general, AVS has no "cancel" function. The one exception to this is available in the Network Editor. If you have started a network executing and you realize that one module in the network is going to take a *very* long time to execute, you can often drag that module over the "hammer" in the lower right corner and terminate that instance of the module. You can do this even while it is executing. You can then create a new instance of the module, hook it into the network, and adjust its parameters to be more reasonable. (Note: builtin modules such as **geometry viewer** cannot be hammered while they are executing.)

If your datasets are large, you can subsample the data with the **crop** and **downsize** modules in the Network Editor.

APPENDIX B

GEOMETRY VIEWER SCRIPT LANGUAGE

Prolog

The Geometry Viewer is the oldest part of AVS. It was introduced in AVS Release 1. The Geometry Viewer Script Language was developed at the same time to provide an ASCII language for specifying and saving the properties of geometry objects, object collections, and Geometry Viewer scenes. More recent releases of AVS have newer means, such as the Command Language Interpreter, of performing these same tasks that offer better functionality and are more consistent with the rest of the AVS interface.

Nonetheless, the Geometry Viewer Script Language is still a useful interface. It is the language of existing AVS 3 and earlier *.obj* and *.scene* files, and it remains a convenient means for performing functions such as creating object hierarchies that are not possible with the Geometry Viewer interface. For these reasons, it continues to be documented in this appendix to the *AVS User's Guide.*

Introduction

The AVS **Geometry Viewer Script Language** provides a simple method for creating objects with specific properties (color, reflectance characteristics, rendering method). You can define objects hierarchically, and specify multiple instances of an object in a hierarchy. You can also define entire scenes, which comprise objects, lighting, and one or more cameras (views).

Note: The Script Language *is not* the same thing as the Command Language Interpreter.

A script is an ASCII file, which you can create with any text editor. You store the script under a filename with extension *.obj* or *.scene*. Such scripts can then be read by the AVS viewing application, using the **Read Object** and **Read Scene** functions.

The AVS Geometry Viewer will create a file using the Script Language whenever you use the **Save Object** or **Save Scene** function rather than a CLI *.scr* file if you define the environment variable **AVS_GEOM_WRITE_V30.** It is often useful to create a file in this way, then revise it later using a text editor.

A programmer can access the script language's functionality through the OBJ object library. This library is documented in the *AVS Porting and Implementation Guide*.

Scene Files and Object Files

The AVS Script Language can be used to represent either object information alone, or object information along with viewing and light-source information. A single file format handles both these cases, but for convenience, filename extensions are used to distinguish a *scene* (which contains object, viewing and light source information), from an *object* (which contains only object information). A scene file should always have a *.scene* extension, an object file should always have a *.obj* extension. For both objects and scenes, the script file format specifies properties of the top-level object. Views and light sources are considered to be properties of this object. The only real difference between a file with a *.scene* and a *.obj* extension is that:

- Reading a *.scene* file creates a new top-level object, then modifies its properties.
- Reading a *.obj* file causes the existing top-level object's properties to be modified.

Example: This object (*.obj*) file sets the color of the current top-level object to red:

```
set_color 1.0 0.0 0.0
```

Script Language Commands

The script language commands are listed in the table below.

Table B-1 AVS Script Language Commands

Type	Command	Description
Object	read	Read object from disk file
	group	Create group of objects
	cycle	Create "animation group"
	set_color	Set color of object
	set_material	Set surface properties of object
	set_matrix	Set transformation matrix
	set_position	Set X-Y-Z position of object
	set_render_style	Set rendering style of object
	rotate	Rotate object
	translate	Translate object
	scale	Scale object

Table B-1 AVS Script Language Commands

Type	Command	Description
Viewing	view	Define a new view (window)
	set_matrix	Set the viewing matrix
	set_position	Set the world coordinates origin
	depth_cue	Turn on depth-cueing in the view
	inactive	Make the view inactive
	no_zbuffer	Turn off Z-buffering of lines in the view
	rotate	Rotate object
	translate	Translate object
	scale	Scale object
Lighting	light	Define a new light
	set_matrix	Set rotational position of light
	set_position	Set X-Y-Z position of light
	set_color	Set color of light

These commands are described in the sections that follow.

Object Commands

Each object command affects the properties of a particular object. Some object commands create a new object, which is added as a child of the current object. You can also specify the initial properties of the new object. This mechanism can be used to create an arbitrarily complex hierarchy of objects.

The read Command

read *name geom-file* **{** *object-properties* **}**

This command reads in a new object, making it the child of the current object. The object is given the name *name* and is read from file *file*. The file must be an AVS geometry file, with a *.geom* extension.

If the *.geom* file is in the same directory as the *.obj* or *.scene* file being created, specify it with a simple filename. Otherwise, specify it with an absolute (complete) pathname.

The new object can have its initial properties set in the optional *object-properties* field. (Currently the OBJ library does not support this functionality of including properties in a read command that creates a new object — an object can either have data or can reference other data, but it cannot do both).

The read_subset Command

read_subset *name geom_file groupname* **{** *object-properties* **}**

This command extracts from the named file, all geometry objects that have the specified group name. See the description of the **GEOMset_object_group** routine in the "Geometry Library" appendix of the *AVS Developer's Guide* for information on how to specify a group name for a geometry object.

The group Command

group *name* **{** *object-properties* **}**

This command creates an object whose sole purpose is to group together a list of subobjects. The object is given the name *name* and has a set of initial properties (including a list of subobjects) in *object-properties*. For example:

```
group TheFlintStones {
   group FlintStone {
      read Fred flintstone.geom {
         set_color 0.0 0.0 0.0
         set_position 0.0 1.0 0.0
      }
      read Wilma flintstone.geom {
         set_color 1.0 0.0 0.0
         set_position 0.0 0.0 1.0
      }
   }
   group Rubble {
      read Barney rubble.geom {
         set_color 1.0 1.0 1.0
         set_position 1.0 0.0 0.0
      }
      read Betty rubble.geom {
         set_color 0.0 1.0 1.0
         set_position 0.0 1.0 0.0
      }
   }
}
```

The cycle Command

cycle *name* **{** *object-properties* **}**

This command creates an *animation object*, for which all of its children are considered to be mutually exclusive representations of the same geometry. This can be used to create an animation sequence. It can also be used to create a list of different representations that can be selected for a particular object (e.g. spheres vs. lines for a molecule). The object is made a child of the current object, and initial properties can be specified for the object. For example:

```
cycle Molecule {
   read balls sphere.geom {}
   read stick ball_and_stick.geom {}
   read lines line.geom {}
}

cycle Face {
   read Smile smile.geom {}
   read Frown frown.geom {}
   read Grimace grimace.geom {}
}
```

The set_color Command

set_color *red green blue*

This command sets the color of the object to the specified RGB value. *red*, *green*, and *blue* must be between 0 and 1.

The set_matrix Command

set_matrix *4x4-matrix*

Sets the current transformation to be the *4x4-matrix* specified. Supplying a transformation in this matrix allows you to alter the center of rotation of the object.

Note that the specified matrix replaces the existing transform. Contrast this with **rotate**, **translate**, and **scale**, which concatenate transformations with the existing one.

The set_position Command

set_position *x y z*

This command sets the position of the object to be the *x*, *y*, and *z* values specified. Setting the position does not alter the center of rotation of the object.

The set_material Command

set_material *ambient diffuse specular spec-exponent transparency* \
spec-red spec-green spec-blue

Sets the material properties of the object. All values except for the specular exponent vary from 0 to 1. The specular exponent, which specifies the roughness of the surface, should lie between 1 (roughest) and 200 (smoothest).

The set_render_style Command

set_render_style *style*

Sets the rendering method used to draw the object. *style* should be one of the following:

lines
gouraud
phong
inherit
flat
smooth_lines
no_light

The rotate Command

rotate *angle x y z*

Rotates the object by *angle* degrees counterclockwise around the vector (*x*,*y*,*z*). This transformation is concatenated with the object's current transform.

The translate Command

translate *x y z*

Translates the object by the vector (*x*,*y*,*z*). This transformation is concatenated with the object's current transform.

The scale Command

rotate *angle sx sy sz*

Scales the object by *sx*, *sy*, and *sz* in the X, Y, and Z directions. This transformation is concatenated with the object's current transform.

Viewing Commands

Views (windows) can be created using the **view** command. Like objects, views also have properties. A view should only be specified in a file that has a *.scene* extension.

The view Command

view *name width***x***height*+*x*+*y bkg-red bkg-green bkg-blue*
{ *view-property-commands* **}**

The following example creates a 500x500 pixel view named "Bob", at offset 100,100 from the upper left corner of the screen, and with a red background.

```
view Bob 500x500+100+100 1.0 0.0 0.0 {  }
```

The *view-property-commands* are described in the following sections.

The set_matrix Command

set_matrix *4x4-array-of-floats*

This command sets the viewing matrix.

The set_position Command

set_position *x y z*

This command sets the position of origin of the world coordinate system.

The rotate Command

rotate *angle x y z*

Rotates the object by *angle* degrees counterclockwise around the vector (*x*,*y*,*z*). This transformation is concatenated with the object's current transform.

The translate Command

translate *x y z*

Translates the object by the vector (*x*,*y*,*z*). This transformation is concatenated with the object's current transform.

The scale Command

rotate *angle sx sy sz*

Scales the object by *sx*, *sy*, and *sz* in the X, Y, and Z directions. This transformation is concatenated with the object's current transform.

The depth_cue Command

depth_cue

This command turns on depth-cueing in the view.

The inactive Command

inactive

This command sets the initial state of the view to be inactive.

The no_zbuffer Command

no_zbuffer

This command disables Z-buffering of lines in the view.

For example, this command creates a view with a red background, and with depth-cueing of lines turned on and Z-buffering of lines turned off.

```
view Joe 100x100+100+200 1.0 0.0 0.0 {
   depth_cue
   no_zbuffer
}
```

Geometry Viewer Defaults File

For compatibility with Release 1 of AVS, you can specify a "defaults file" to be read by the Geometry Viewer when you start AVS with the **-geometry** command-line option. For example:

```
avs -geometry -defaults /usr/johnp/avs/geom_windows.dfl
```

The defaults file defines a series of windows, assigning each one a name, a size and position (in standard X Window System notation), and an RGB background color. The first window in the series is used for the first Geometry Viewer window that appears. Subsequent windows are used, in turn, by the **Create Scene** and **Create Camera** functions and by the **render geometry** or **geometry viewer** modules. If the end of the series is reached, additional windows are created with the same size as the last window, but slightly offset from each other.

Following is a sample Geometry Viewer defaults file:

```
view JOHNP01 400x400+300+100 1.0 1.0 0.0
view JOHNP02 400x400+750+100 0.0 1.0 1.0
view JOHNP03 300x300+400+500 0.0 1.0 1.0 {
        depth_cue
}
```

Note that you can set default viewing parameters for the windows you create. In this example, window JOHNP03 starts with depth-cueing turned on.

NOTE: Release 1 of AVS reads defaults from */usr/avs/runtime/avsrc* if you do not specify a defaults file on the command line.

Lighting Commands

Like viewing commands, lighting commands should only be specified in a file that has a *.scene* extension.

The light Command

light *type index* **{** *lighting-property-commands* **}**

This command turns on a light of *type,* where *type* is one of **ambient**, **directional**, or **point**. The light is given index *index* and can contain properties specified in *lighting-properties* (see below). Light indices should be in the range of 1 to 16. In the AVS viewing application, a single ambient light source is assigned to index 16.

For example, this command turns on light 1, making it a directional light with the default lighting properties:

```
light directional 1 {}
```

The *lighting-property-commands* are described in the following sections.

The set_matrix Command

set_matrix *4x4 matrix*

This command sets the transformation matrix for the light. In the case of directional lights, only the rotation portion of the matrix is used (although the

rest of the matrix can be used to affect the graphical display of light vectors). In the case of a point light, this matrix only affects the graphical representation of the point light source icon.

The set_position Command

set_position *x y z*

This command sets the position of point light sources. This attribute does not affect directional light sources.

The set_color Command

set_color *red green blue*

This command sets the light source color.

Example Scene File

```
view AVS 503x529+375+208 0.015686 0.078431 0.196078 {
   set_matrix    0.968946    -0.201782    0.142953    0.000000
                 0.246114    0.730629    -0.636879    0.000000
                 0.024065    0.652280    0.757599    0.000000
                 0.000000    0.000000    0.000000    1.000000
}
light directional 1 {
}
light ambient 16 {
}
read teapot.2 teapot.pobj {
   set_color 0.162 0.046 0.013
   set_material 0.287 0.444 0.931 10.000 0.000 0.990 0.241 0.027
   }
}
```