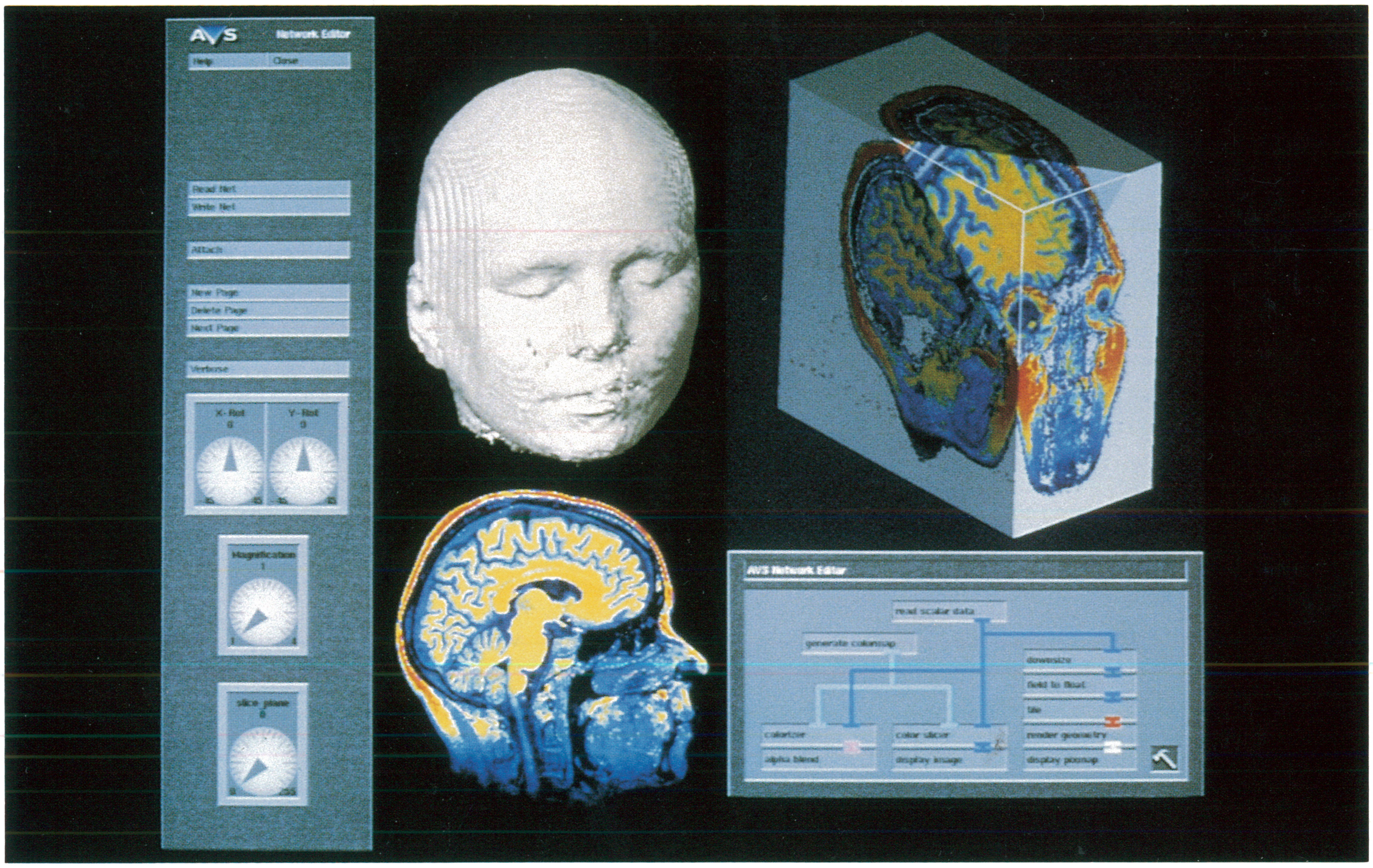
Stardent


AVS
Network Editor
Help
Close
Read Net
Write Net
Attach
New Page
Delete Page
Next Page
Verbose
X-Rot
0
Y-Rot
0
-45
45
-45
45
Magnification
1
1
4
slice plane
0
0
255
AVS Network Editor
read scalar data
generate colormap
downsize
field to float
tile
render geometry
display pixmap
colorizer
alpha blend
color slicer
display image

Stardent

Application Visualization System

User's Guide

MD – 2301

NOTICE

This document, and the software and other products described or referenced in it, are confidential and proprietary products of Stardent Computer Inc. (Stardent) or its licensors. They are provided under, and are subject to, the terms and conditions of a written license agreement between Stardent and its customer, and may not be transferred, disclosed or otherwise provided to third parties, unless otherwise permitted by that agreement.

NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT, INCLUDING WITHOUT LIMITATION STATEMENTS REGARDING CAPACITY, PERFORMANCE, OR SUITABILITY FOR USE OF PRODUCTS OR SOFTWARE DESCRIBED HEREIN, SHALL BE DEEMED TO BE A WARRANTY BY STARDENT FOR ANY PURPOSE OR GIVE RISE TO ANY LIABILITY OF STARDENT WHATSOEVER. STARDENT MAKES NO WARRANTY OF ANY KIND IN OR WITH REGARD TO THIS DOCUMENT, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

STARDENT SHALL NOT BE RESPONSIBLE FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT AND SHALL NOT BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR RELATED TO THIS DOCUMENT OR THE INFORMATION CONTAINED IN IT, EVEN IF STARDENT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The specifications and other information contained in this document for some purposes may not be complete, current or correct, and are subject to change without notice. The reader should consult Stardent for more detailed and current information.

RESTRICTED RIGHTS LEGEND (U.S. Department of Defense Users)

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights In Technical Data and Computer Software clause at DFARS 252.227–7013.

Stardent Computer Inc.
95 Wells Avenue
Newton, MA 02159

RESTRICTED RIGHTS NOTICE (U.S. Government Users excluding DoD)

Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in the Commercial Computer Software — Restricted Rights clause at FAR 52.227–19(c)(2).

Table of Contents

Chapter 6
The Network Editor Subsystem

Chapter 7
AVS Module Manual Pages

Appendix A
Tutorial: Image Viewer, Volume Viewer, and Network Editor

Appendix B
Geometry Viewer Tutorial

Appendix C
Geometry Conversion Programs

Appendix D
The Geometry Viewer Script Language

Appendix E
AVS File Formats

Preface

This manual describes the use of the interactive visualization program, **avs**, which is a major component of the Stardent Application Visualization System (AVS™). AVS also includes a facility for users to create their own computational *modules*, which can then be used with the **avs** visualization program. This facility is described in the *AVS Developer's Guide*.

After reading the introductory material in Chapter 1, you may want to skip to the tutorials in Appendix A and Appendix B. These tutorials provide a guided tour of the several subsystems of the **avs** program.

The remaining chapters discuss startup issues and each of the **avs** subsystems.

Stardent

1

Introduction to the Application Visualization System

Table of Contents

Chapter 1
Introduction to the Application Visualization System

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 particular, the *graphics supercomputer* combines minisupercomputer-class computational power with 3D graphics capabilities. 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, producing real-time interactive displays.

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 includes a substantial number of visualization techniques which the user can invoke simply by selecting them from a menu. These image-viewing and volume-viewing techniques will satisfy many users' first-level needs in turning data into pictures.

For situations in which these standard techniques do not suffice, 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 geometric primitives (triangles, lines, spheres, etc.)
- ❑ **Rendering** the geometric primitives into pictures on the display screen.

The flow networks are built from a menu of modules by using a direct-manipulation interface. 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 in this way, using standard modules and without resorting to any traditional procedural programming.

AVS includes a rich set of modules for construction of networks. Given the nature of scientific visualization and the need for extensibility, AVS also supports the creation and dynamic loading of new modules. 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 vectors of floating-point values, 2D and 3D grids with vectors of floating-point values at each grid point, geometric data, and images. Byte and integer data types are also supported.

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 (dials, joysticks, etc.).

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

Scientific and Engineering Data

In the engineering and scientific arena, a set of data to be processed by computer typically takes the form of a sequence of numbers. Sometimes, the numbers are generated as a real-time data stream. Many measurement instruments can produce streams of digital output (perhaps aided by an analog-to-digital converter). Often, the data is being produced by a running computational process. Sometimes, the numbers have been generated at some previous time, and are stored in a file on disk. AVS has facilities for handling both real-time data streams and disk-based data.

Each number in a data set can be represented in a variety of ways. AVS can handle the following integer and floating-point numerical formats:

byte (8 bits)
A single byte can represent an unsigned integer in the range 0..255 or a signed integer in the range –128..127.

integer (32 bits)
A single machine word can represent an unsigned integer in the range $0..2^{32}-1$ (0..4294967295) or a signed integer in the range $-2^{31}..2^{31-1}$ (-2147483648..2147483647).

single-precision (32 bits)
A single machine word can also be used to represent a floating-point quantity in IEEE 754 *single* format.

double-precision (64 bits)
Two machine words can be used to represent a floating-point quantity in IEEE 754 *double* format.

In many cases, the sequence of numbers in a data set has an implied or explicit structure. For instance, a sequence of 40,000 numbers may represent a 2D square grid (a 200x200 matrix). Similarly, a sequence of 500,000 numbers might represent a 100x200x25 lattice of data points.

Presumably, a numerical grid corresponds to a physical grid with a particular distance between grid points. In some cases, the grid may be non-regular — the distance between grid points is variable, rather than constant. It is also possible for the grid to describe a curved or arbitrarily deformed space, instead of a rectangular space. (For more on this subject, see the *Preparing Your Data for Use with AVS* later in this chapter.)

In AVS, data files always begin with a *header* that specifies the overall structure of the data. Additional structural information (for instance, the real-world coordinates that correspond to the numerical data grid) can also be included in the data file.

Other Data Formats

AVS can also use data that is in a format other than a simple stream of numbers. At many sites, purely numerical data has already been processed into a structured form, by a user-written program or by an application software package. For instance, Figure 1-1 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.

Figure 1-1. Data File in Brookhaven Protein Data Bank Format

```
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 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 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.

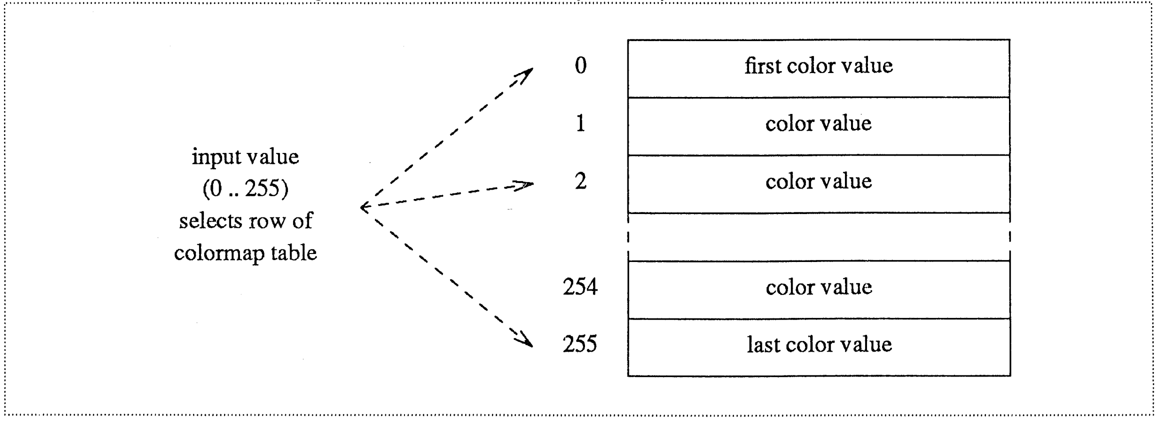
Pixel-Based Visualization

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 simple table lookup, called a *colormap*. You can define, save, and retrieve your own colormaps. AVS includes an interactive drawing tool for generating colormaps conveniently and quickly.

Colormap Lookup

An AVS colormap is a 256-row table; each row specifies a 24-bit "true-color" value (and, optionally, an 8-bit auxiliary field), as shown in Figure 1-2. 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 1-2. AVS Colormap Lookup



In general, 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.

NOTE *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 machine with only 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.

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 process, called *alpha blending* is described in more detail in the manual page for the **alpha blend** module (see the chapter entitled *AVS Module Manual Pages*).

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*.
- ❑ 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.
- ❑ Given window velocity data as above, construct flow lines to represent the motion of an object through the field.

The AVS Subsystems

AVS's capabilities are divided into a group of *subsystems*:

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

Geometry Viewer
The **Geometry Viewer** subsystem allows you to compose "scenes" that contain 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.).

The Geometry Viewer is an expanded version of the software that was delivered as Release 1 of AVS.

Volume Viewer
The **Volume Viewer** subsystem is a high-level tool for visualizing volume (3D) data. It can handle a wide variety of data sets that convey information for a sampling of points in a 3D space.

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

Exit AVS
Ends the AVS session. A pop-up window appears, prompting you to confirm your choice or cancel it.

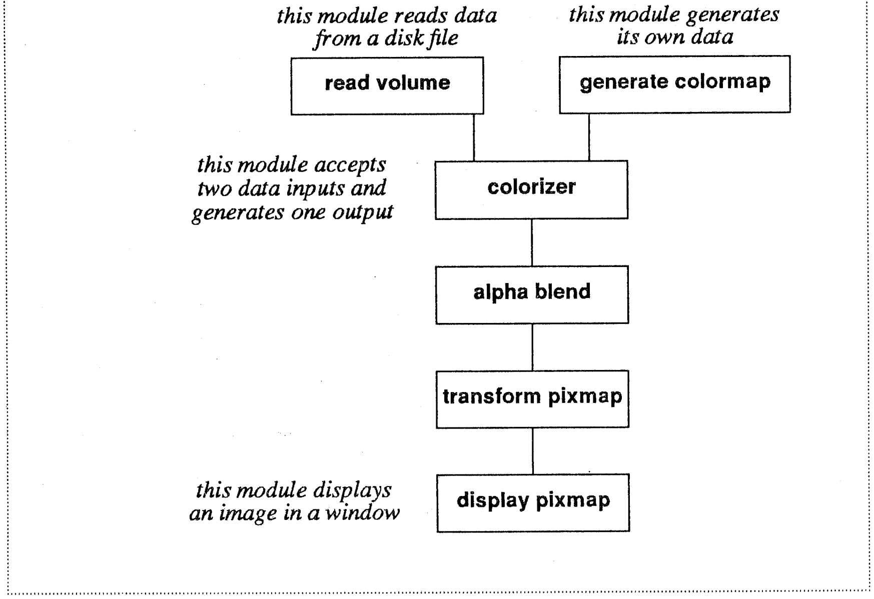
When using AVS for the first time, you should familiarize yourself with the product by using the *Image Viewer* and *Volume Viewer* subsystems. These are menu-driven and quite easy to use. Using the sample data provided in the */usr/avs/data* directory, you can perform significant visualization functions with just a few clicks of the mouse. Appendix A provides an illustrated tutorial introduction to these subsystems. The tutorial also includes a walkthrough of the *Network Editor* subsystem.

Appendix B is a tutorial introduction to the *Geometry Viewer* subsystem.

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 together 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 transformed into one or more images by a collection of modules, and finally is displayed in a window onscreen. Figure 1-3 shows a simple network of modules.

Figure 1-3. Simple Network of AVS Modules


this module reads data from a disk file
this module generates its own data
read volume
generate colormap
this module accepts two data inputs and generates one output
colorizer
alpha blend
transform pixmap
this module displays an image in a window
display pixmap

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 a few modules to do real, useful work.

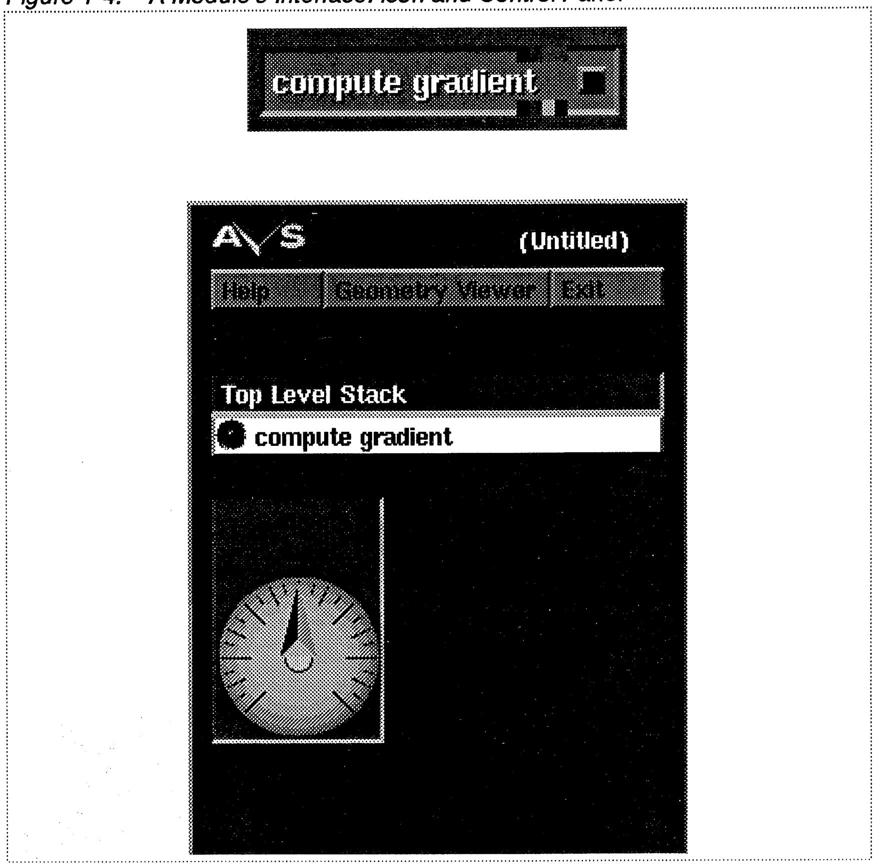
You can include a particular module in any number of AVS applications (*networks*); you can even include the same module more than once in a single network.

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.*
- ❑ A set of *input parameters*, which controls 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.
- ❑ A set of *data outputs.*

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

Figure 1-4. A Module's Interface: Icon and Control Panel



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.

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 the port accepts:

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

Port Color	Data Type
red	geometry
yellow	colormap
light blue	pixmap
multi-color	field

AVS helps you to match data types as you interactively build a network. When you begin to establish a module-to-module connection, AVS shows you the valid possibilities.

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).

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 1-3 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.

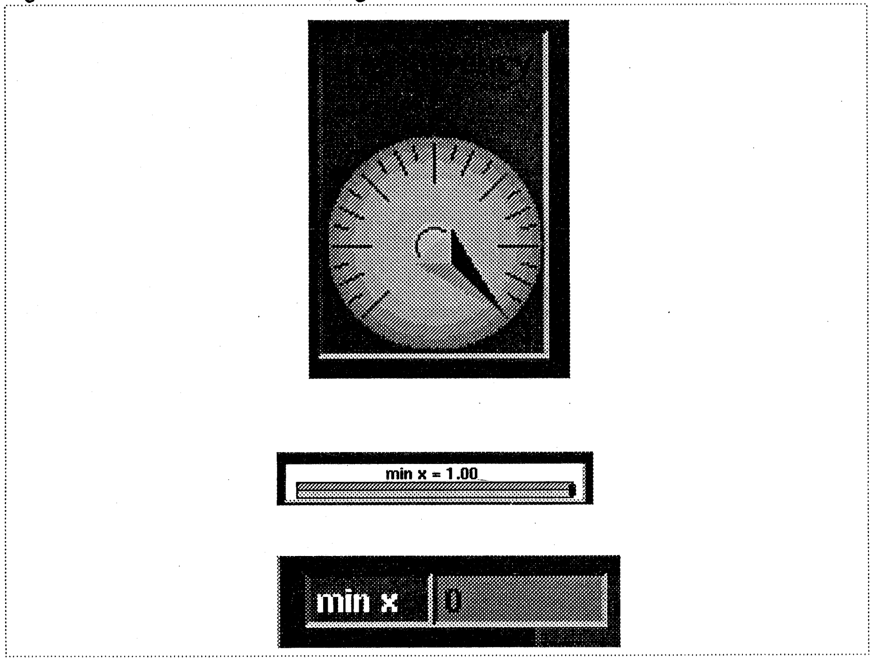
□ The **generate colormap** module creates and outputs a colormap that transforms byte values into color values. Its input parameter is implemented as an interactive "colormap editor", with which you specify the 256-entry colormap.

□ The **alpha blend** module reduces a 3D block of partially-transparent color values to a 2D image, using a projection algorithm. Its input parameters determine the orientation of the block with regard to the direction of the projection.

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, blow up an image to examine some detail, etc.

Each of a module's parameters is represented by an onscreen *control widget*. Figure 1-5 presents examples of control widgets.

Figure 1-5. Module Control Widgets



AVS implements 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.

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 UNIX 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 its output data, it passes the data back to the Flow Executive, then returns to its dormant state.
- ❑ *Coroutine* modules are active, not passive. Rather than being like a subroutine, a coroutine is a 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 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 are 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, simply by creating lists of modules with a text editor program.

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.

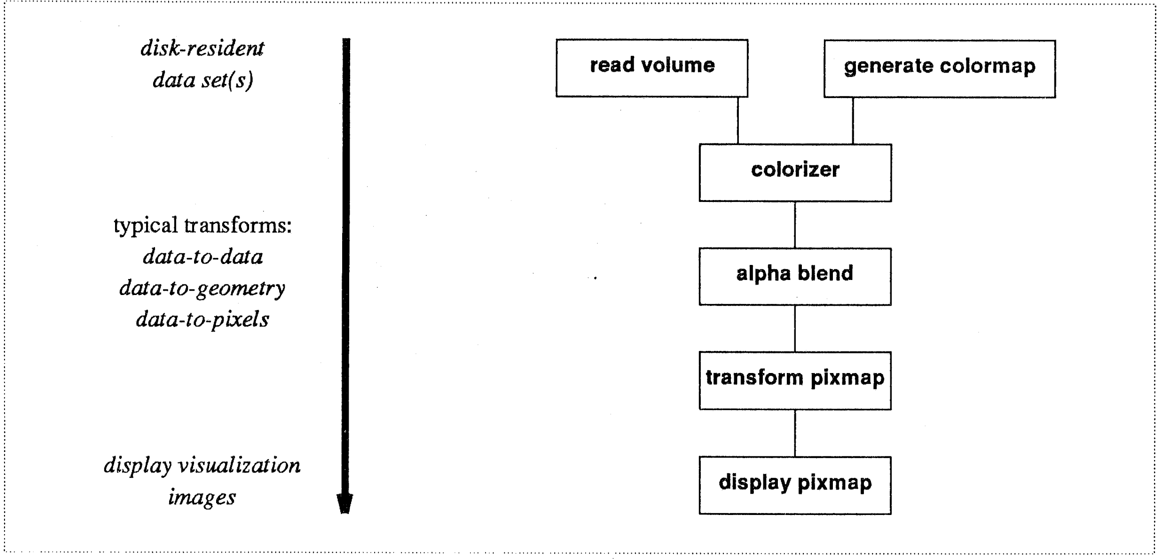
AVS Networks

As *modules* are the computational units in AVS, *networks* are the operational units. Given data that you wish 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.

Data Flow in an AVS Network

Figure 1-6 repeats the network shown at the beginning of the *Modules* section above. This time, the figure emphasizes the way data flows through a typical AVS network.

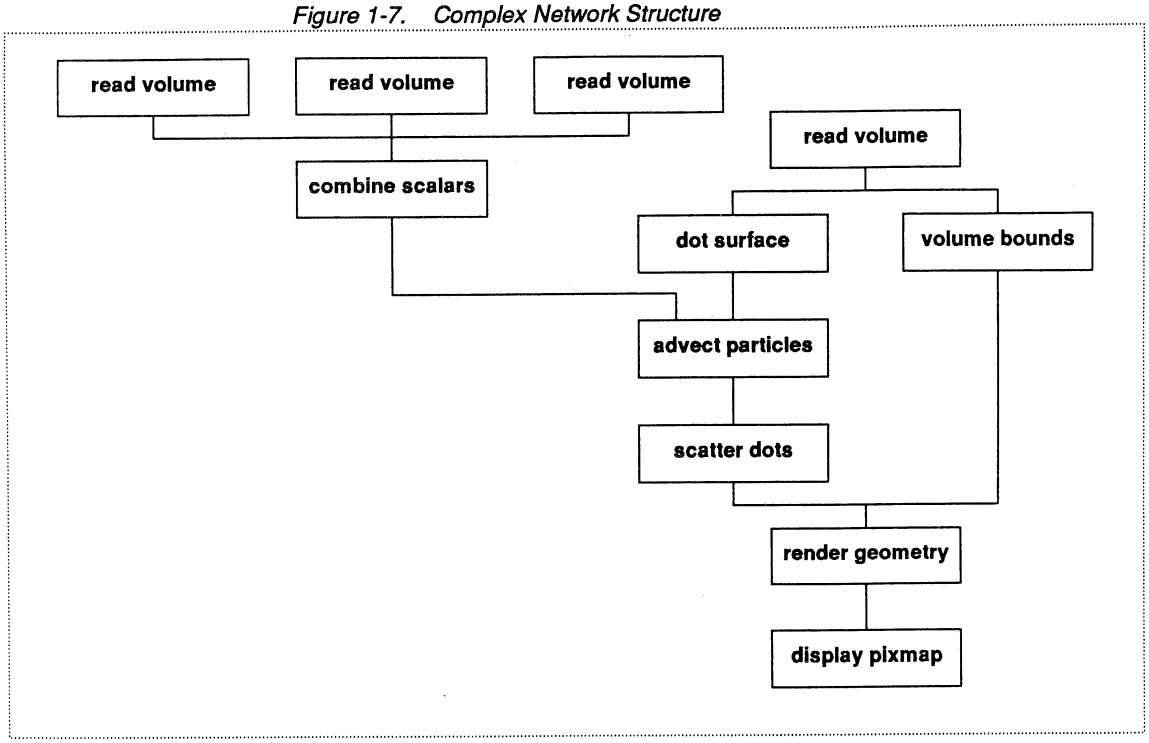
Figure 1-6. Data Flow in a Network



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 subnetworks. Figure 1-7 illustrates a more sophisticated network topology:

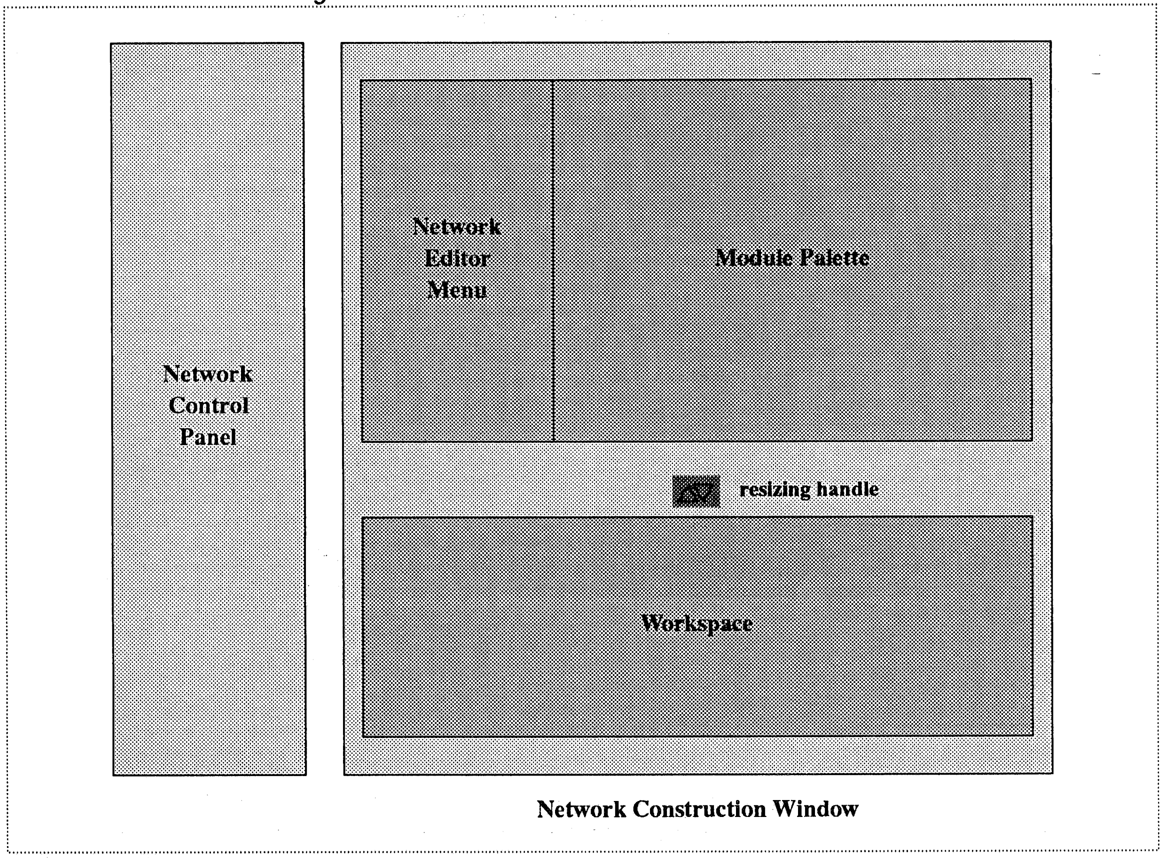
Figure 1-7. Complex Network Structure



There are limits to network complexity, however. Networks are inherently flat — AVS provides no support for creating hierarchical structures. And networks may not contain "cycles": a module's output data cannot subsequently be fed back into the module as input, directly or indirectly.

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.

Network Control Panel

A network's data-flow diagram omits 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.

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.

You can also extend the Network Control Panel to include additional physical input devices: the DIGIT™ dialbox and the Spatial Systems Spaceball™. Certain types of input parameters can be associated with a dial or the Spaceball, instead of with an onscreen control widget.

Networks in the Image Viewer and Volume Viewer

Two of the AVS subsystems, the *Image Viewer* and the *Volume Viewer*, make use of networks transparently. These subsystems are entirely menu-driven: through menu choices, you select the data to be processed along with one or more visualization techniques to be applied to the data. Each technique is implemented with a pre-existing AVS network. You can control the execution of the networks using control panels, as described in the preceding section. But you need not go through the process of creating your own networks.

This convenience is balanced by flexibility, however. Both the Image Viewer and the Volume Viewer allow you to view the networks that implement the visualization techniques, and to switch to the Network Editor in order to revise or enhance them.

AVS Display Windows

AVS creates its visualization images in *display windows* on the screen. (There is also a provision for saving 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 the X window manager. AVS also provides some window-oriented functions, such as *zoom* and *unzoom*. You can integrate display windows into the control panels of the visualization networks you build, allowing you to build predictable and space-efficient user interfaces.

Preparing Your Data for Use with AVS

AVS is designed for users who already have large numerical data sets waiting to be visualized. Inevitably, the data sets will emcompass a wide range of file formats. Some formats may be highly structured, comprising many types of data records. Other formats may be essentially unstructured: a small amount of header information followed by a stream of data.

In its current implementation, AVS can directly read several file formats, which are described below. Some of these formats (e.g. the Brookhaven Protein Data Bank format) are already in general use in the scientific/engineering community. If your data is already in one of these formats, you can start immediately to visualize it using AVS. Other formats are AVS-specific. This means you will need to do some data-conversion programming work to make such data directly usable by AVS.

File Formats that are in General Use

Files in the following formats can be used directly by AVS:

TABLE 1-2. AVS-Readable File Formats: General Use

File Format	AVS Filename Suffix
Mathematica ThreeScript	*.ts*
Movie BYU	*.byu*
Protein Data Bank	*.pdb*
UNC	*.ppoly*
Wavefront	*.wfront*

Each of these formats embodies a description of one or more geometric objects. If a file is named with the appropriate suffix from the table above, a single command in the Geometry Viewer subsystem reads the file, automatically converts its geometric descriptions to the AVS *geometry* format, and displays the object(s) in a window. (The converted data is also stored on disk, in a file with a *.geom* suffix.)

There is also an AVS module, **pdb to geom**, that reads a file in the Protein Data Bank format and outputs the data as an AVS *geometry*. (None of the other conversion routines are implemented as modules.)

For more information of conversion of these data formats to the *geometry* format, see the *Geometry Conversion Programs* appendix.

AVS-Specific File Formats

AVS has several data formats of its own, which are designed to accommodate a wide variety of scientific/engineering data sets with a minimum of conversion effort. These formats are summarized here and described in more detail in the following sections.

Geometry
As described in the preceding section, AVS has its own data format for the specification of display output in terms of geometric primitives: lines, triangles, spheres, etc. Many (but not all) numerical data sets are converted to geometries during the visualization process.

Image
AVS can read an image of any size. The file format is essentially a stream of pixel values: each pixel is specified by a red-green-blue triple.

Field
The most flexible AVS data format is the *field*, a generalization of the *n*-dimensional array construct that is commonly used to represent scientific data sets.

Volume Data
The AVS *field* construct is extremely general and powerful. As a convenience, AVS also supports a simpler format that handles one commonly-used type of field, *volume data*.

Geometry Data File Format

The AVS Geometry Viewer subsystem (also implemented as the **read geometry** module) read files in the GEOM file format. Such files can be created with routines in the special GEOM programming library (*libgeom.a*), which is included with AVS. For a description of this library and the file format, see the **geom**(3V) manual page (reproduced in Appendix B).

Geometry data files should have names that end with a *.geom* suffix.

Image Data File Format

The **read image** and **image manager** modules 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:

0 0 128 0

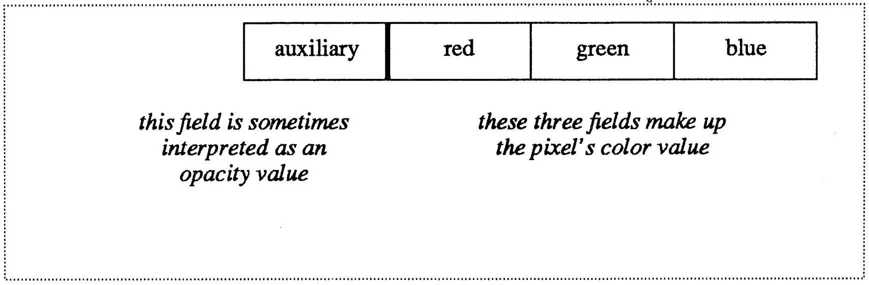
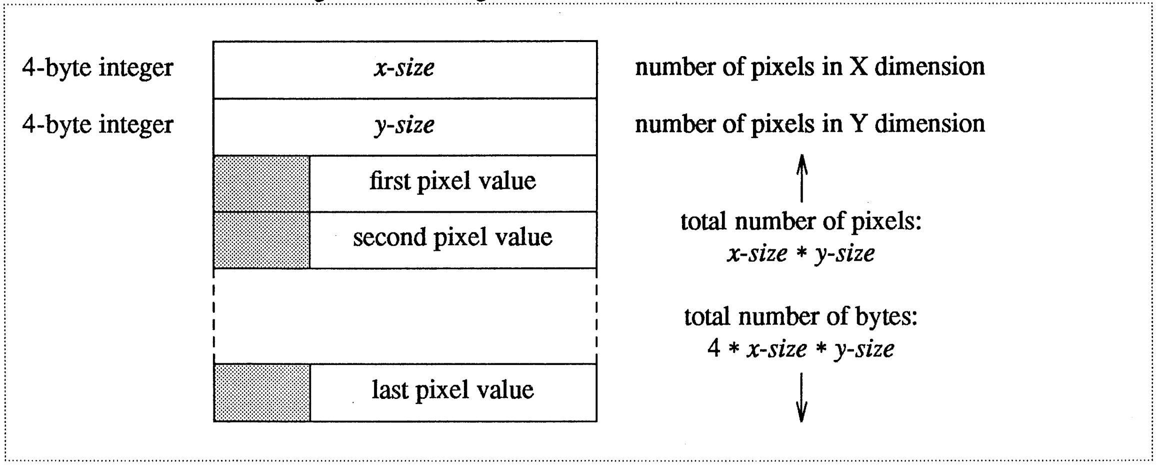

auxiliary
red
green
blue
this field is sometimes interpreted as an opacity value
these three fields make up the pixel's color value

Figure 1-9 illlustrates the AVS *image* data file format. Image data files should have names that end with a *.x* suffix.

Figure 1-9. Image Data File Format



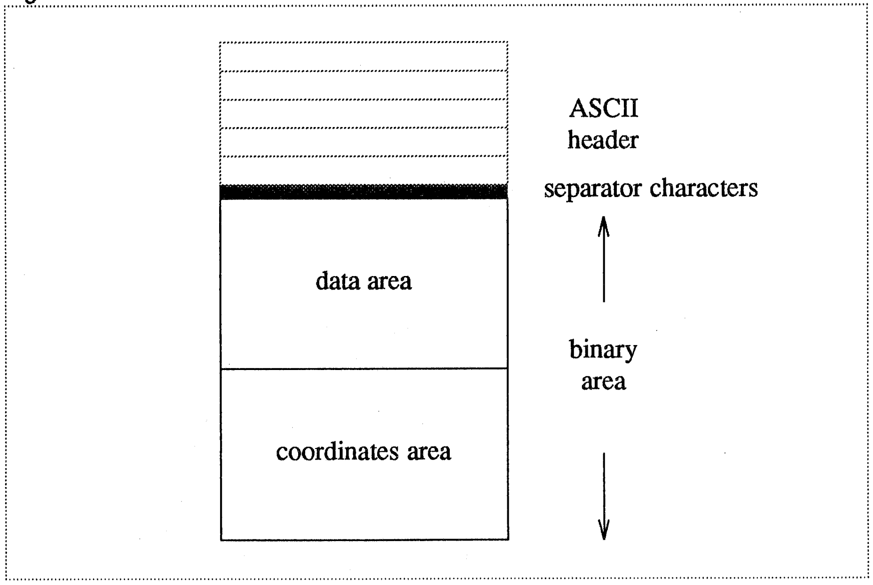
Field Data File Format

A *field* is a generalization of the familiar array structure. Whereas each element of an ordinary 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.

Figure 1-10 illustrates the top-level structure of an AVS field: Field data files should have names that end with a *.fld* suffix.

Figure 1-10. AVS Field Structure



Before describing the field file format in more detail, we present several examples of fields. This will serve to introduce terminology and to illustrate the power and flexibility of the field construct.

Example 1: Uniform 2D Field

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*:

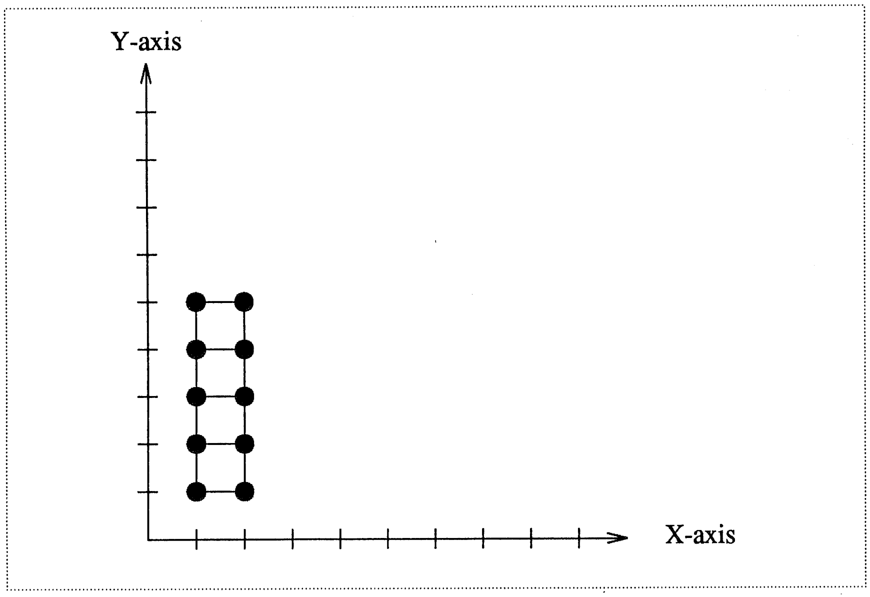
DATA(2,3) = (2.51, 1.09, 5.73)

or

DATA(2,3) = (2.51, 1.09, 5.73, 0, 1)

In the first case, 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. The 5-vector in the second case might represent the temperature-pressure-humidity measurements at each location in space, along with two boolean values to indicate the presence/absence of other atmospheric conditions.

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:



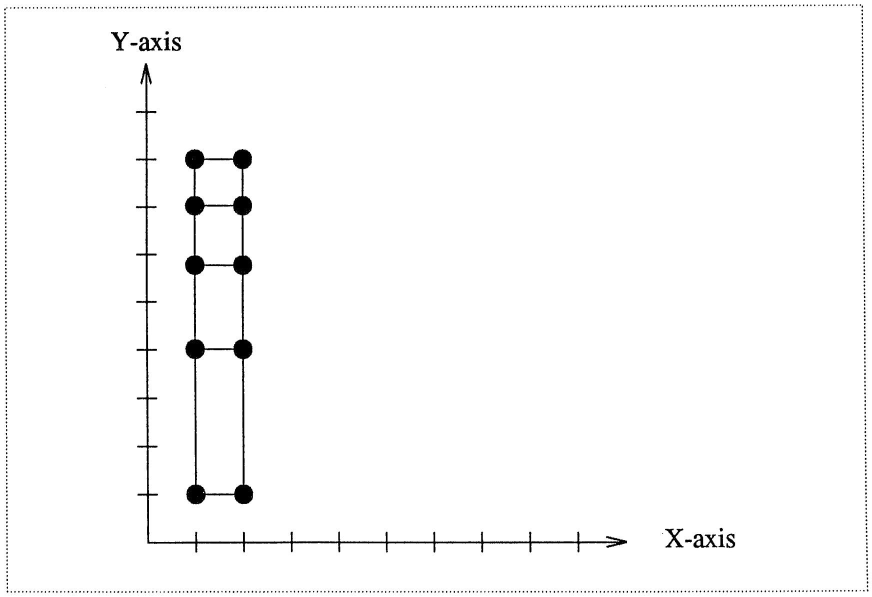
The physical space is a uniformly-spaced lattice. Accordingly, a field with no coordinate data is said to have the field type *uniform*.

Example 2: Rectilinear 2D Field

Continuing the preceding example, we can establish an explicit mapping between the computational and physical spaces by specifying coordinate data:

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

For example, array element IDATA(1,3) is 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 follows:



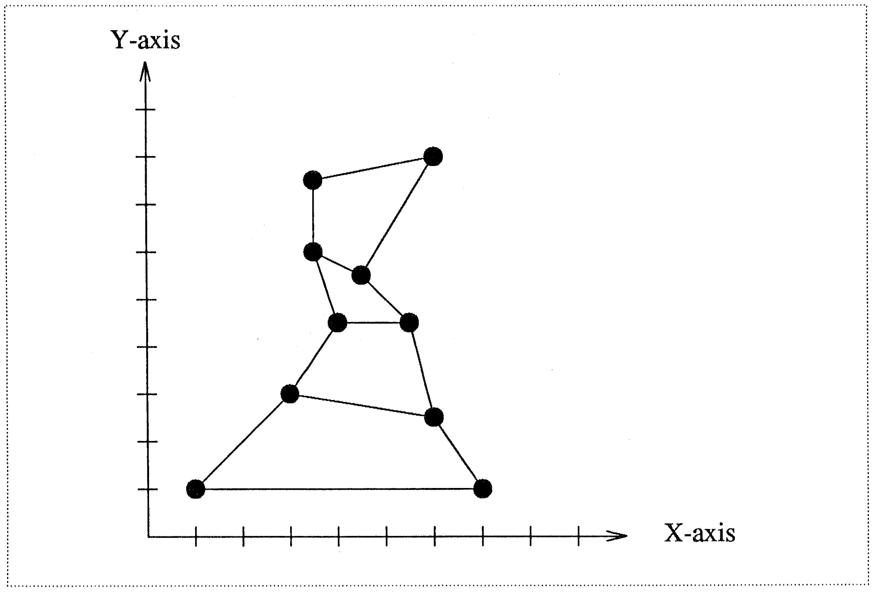
Note that in a *rectilinear* field, lines connecting the lattice points are always mutually orthogonal — all the angle are right angles.

Example 3: Irregular 2D Field

Continuing the example once more, there is another way of establishing a mapping between the computational and physical spaces. Instead of mapping the array indices, we can map individual field elements to arbitrary points in physical space:

DATA(1,1)	–>	(1, 1)
DATA(2,1)	–>	(7, 1)
DATA(1,2)	–>	(3, 3)
DATA(2,2)	–>	(6, 2.5)
DATA(1,3)	–>	(4, 4.5)
DATA(2,3)	–>	(5.5, 4.5)
DATA(1,4)	–>	(3.5, 6)
DATA(2,4)	–>	(4.5, 5.5)
DATA(1,5)	–>	(3.5, 7.5)
DATA(2,5)	–>	(6,8)

This mapping from computational space to the X-Y plane can be pictured as follows:



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.

NOTE *For additional examples, including some involving non-2D fields, see the AVS Data Types chapter in the* ***AVS Developer's Guide****.*

ASCII Header. Every field file must begin with a header in ASCII text format. This header includes a number of "keyword=value" pairs, one per line; it also may include comment lines. For example:

Figure 1-11. ASCII Header for AVS Field

```
# 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 keywords are described below, with reference to the three examples in the preceding sections.

ndim

This value specifies the number of *computational* dimensions in the field. That is, it specifies the number of dimensions in the field element array. In all the examples above, **ndim** has the value 2.

dim1, dim2, ...

The values for these keywords specify the size of the computational space. For a 2D field, you would specify **dim1** and **dim2** values. In all the examples above, **dim1** = 2 and **dim2** = 5. For a 4D field, you would specify **dim1**, **dim2**, **dim3**, and **dim4** values.

nspace

This value specifies the dimensionality of the physical space that corresponds to the computational space. In all the examples above, **nspace** = 2. At the end of Example 3, the following mapping to a 3D physical space is suggested:

DATA(2,3) -> (5.5, 4.5, –8.1)

In this case, **nspace** = 3.

veclen

This value specifies the number of data values for each field element. Example 1 above discussed two possibilities:

DATA(2,3) = –5 **veclen** = 1

DATA(2,3) = (2.51, 1.09, 5.73, 0.0, 1.0) **veclen** = 5

data

This keyword takes one of the following values: **byte**, **integer**, **real**, **double**. It indicates the type of data that is supplied for each field element. AVS fields have the restriction that all of the **veclen** data values must be of the same type.

field

This keyword takes one of the following values: **uniform**, **rectilinear**, **irregular**. The main purpose of three examples above is to illuminate these three field types. A **uniform** field (as in Example 1) 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 (as in Example 2), 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 (as in Example 3), 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.

Separator Characters. The ASCII header must be followed by two *formfeed* characters, in order to separate it from the binary area. A formfeed is expressed variously as **Ctrl-L**, octal 14, decimal 12, or hex 0C.

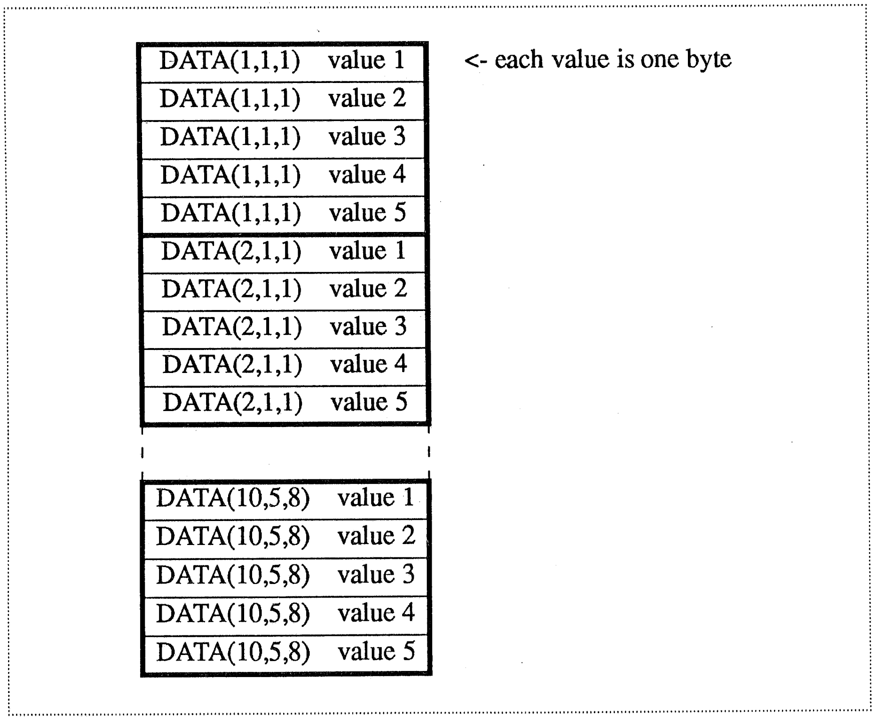
This scheme allows you to use the **more**(1) shell command to examine the header. When **more** stops at the formfeeds, press **q** to quit. This avoids the problem of the binary data garbling the screen.

Binary Area. The binary area consists of all the data that is associated with the field elements, along with all the coordinates. (For uniform fields, the coordinates area is null.)

The **data area** begins with the one or more data values for the first field element. All the data values for a field element are stored together. The first array index varies most quickly ("FORTRAN-style"). For example, suppose the ASCII header is as follows:

```
ndim = 3
dim1 = 10
dim2 = 5
dim3 = 8
nspace=3
veclen=5
data=byte
field=uniform
```

The data ordering can be illustrated as follows:



DATA(1,1,1)	value 1
DATA(1,1,1)	value 2
DATA(1,1,1)	value 3
DATA(1,1,1)	value 4
DATA(1,1,1)	value 5
DATA(2,1,1)	value 1
DATA(2,1,1)	value 2
DATA(2,1,1)	value 3
DATA(2,1,1)	value 4
DATA(2,1,1)	value 5
DATA(10,5,8)	value 1
DATA(10,5,8)	value 2
DATA(10,5,8)	value 3
DATA(10,5,8)	value 4
DATA(10,5,8)	value 5

<- each value is one byte

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* contruct.) Volume data files should have names that end with a *.dat* suffix.

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

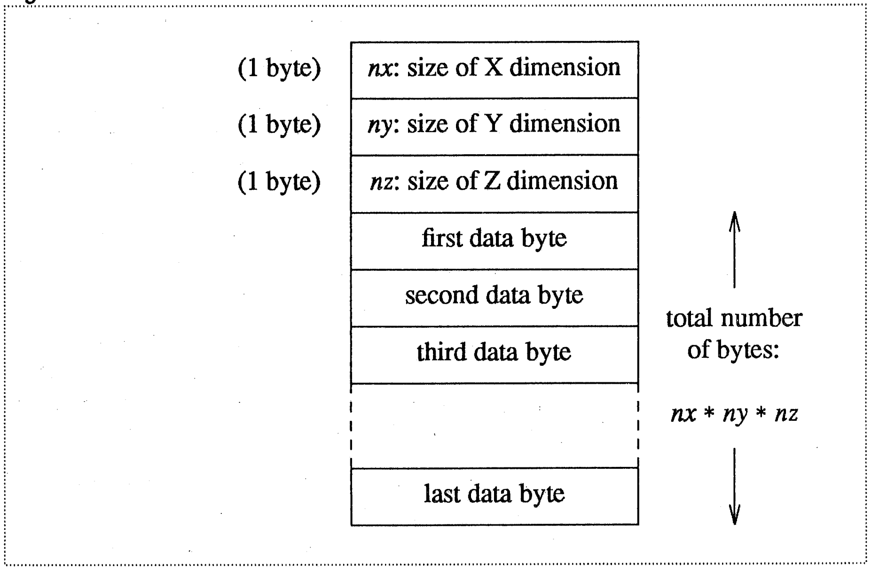
A volume data file begins with a three-byte header, which specifies the size of the array in the first (X), second (Y), and third (Z) dimensions. Since each dimension's size must be expressed as a 1-byte number, the largest array supported by this file format is 255x255x255.

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)
```

Figure 1-12 illustrates the volume data file format.

Figure 1-12. Volume Data File Format



(1 byte)	*nx*: size of X dimension
(1 byte)	*ny*: size of Y dimension
(1 byte)	*nz*: size of Z dimension
	first data byte
	second data byte
	third data byte
	last data byte

total number of bytes:

$nx * ny * nz$

Converting Your Data to an AVS Format

The preceding section contains descriptions of the file formats that are directly readable by the standard AVS modules. This section discusses strategies for converting your data to these formats.

The AVS data formats are quite simple. Each involves a short header followed by a stream of data values. There are two basic strategies for getting your data into these formats.

Writing Your Own Conversion Utility

You can use FORTRAN or C (or any other language) to write a program that converts your data to AVS's *field* format or *volume data* format. In addition to the basic conversion task, the program may need to perform some data filtering, in order to prevent problems in the AVS environment. Here are some issues that such a conversion utility should address:

□ Holes in the data set.

□ Special flag values.

□ Size of the data set.

Ideally, your conversion utility will filter out all data that could confuse AVS or seriously stress its memory allocation system. AVS includes many useful filter modules, but it is unlikely that they will cover every user's needs.

A drawback of this approach is that you may have to maintain two versions of every data set: the original one (presumably used by other analysis software), and the new one for use by AVS. With large data sets, this can have a significant impact on your system's data storage capacity.

Writing an AVS Module

A more elegant approach to converting your data is to write an AVS module that reads a data set in its original form and outputs an AVS *field*. For more information, see the *AVS Developer's Guide*.

Creating a Data Directory

You can store the data to be used with AVS anywhere in the directory hierarchy. By default, AVS initially reads its data from directory */usr/avs/data*. During program execution, you can make any other directory the current data directory.

You can also use command-line options or the AVS startup file to specify any directory as the initial data directory. This is explained further in the next chapter.

Stardent

2

Starting AVS

Table of Contents

Chapter 2
Starting AVS

Starting an AVS session involves two steps: making sure your UNIX environment is properly set, and issuing the appropriate shell command.

AVS Environment Variables

Before starting, make sure that the following UNIX environment variables are properly set.

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

SPACEBALL (optional) Indicates the serial communications port to which a Spaceball device is attached. This can also be set in the AVS startup file (see below).

DIALS (optional) Indicates the serial communications port to which a DIGIT dialbox device is attached. This can also be set in the AVS startup file (see below).

AVS_HELP_PATH
(optional) Specifies one or more locations in the file system for AVS to use when searching for on-line help files. See Appendix D of the *AVS Developer's Guide* for more on this variable.

The AVS Command and Command-Line Options

The basic command to start AVS is simple:

```
avs
```

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 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.

–data *directory*
(startup file equivalent: **DataDirectory**) Specifies the directory in which the AVS Network Editor subsystem initially will look for data files (files used an input to computational modules).

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

–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).

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

–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*

–display *display-name*

Specifies the X Window System display on which AVS is to execute. This overrides the current setting of the DISPLAY environment variable.

–geometry *[geom-option(s)]*

Automatically invokes the Geometry Viewer subsystem at startup. You can include the following options that are specific to this subsystem.

–scene *scene-file*

Automatically loads a "scene" from disk storage. This option executes the Geometry Viewer's **Read Scene** function, using the file *scene-file.scene.*

–dir *pathname*

Specifies *pathname* as the default directory used by the functions **Read Object**, **Save Object**, **Read Scene**, **Save Scene**, and the **Read** and **Save** functions in the Edit Property window.

The default data directory is */usr/avs/data* (same as the Network Editor).

–filter *pathname*

Specifies *pathname* as the directory to search for geometry conversion utilities, named *..._to_geom.* See the *Geometry Conversion Programs* appendix.

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

–defaults *filename*

Specifies a Geometry Viewer defaults file. The format of this file is described in the *Geometry Viewer Script Language* Appendix.

–geometry *geom_spec*

Specifies an X Window System geometry (e.g. **500x500-5-5**) for the initial window created by the Geometry Viewer.

–usage

Displays a usage message for the Geometry Viewer options. No AVS session is started if you type **avs –geometry –usage.**

The Geometry Viewer options correspond to the command-line options recognized by Release 1 of AVS.

–image

Automatically invokes the Image Viewer subsystem at startup.

–volume

Automatically invokes the Volume Viewer subsystem at startup.

–network *network-file*

Automatically invokes the Network Editor subsystem at startup, and loads the specified network file using the **Read Network** function.

–viewer *viewer-file*

Automatically creates a "viewer" that provides "turnkey" access to a group of existing networks. The AVS Image Viewer and Volume Viewer subsystems are implemented in this way.

–modules *directory*

(startup file equivalent: **ModulesDirectory**) Specifies the directory in which the AVS Network Editor subsystem initially will look for executable modules. All executable files in the directory are examined to determine whether they contain one or more modules.

You can use more than one **–modules** option to have AVS search through multiple directories for modules. The default modules directory is */usr/avs/avs_library*.

–usage

Displays a usage message for the AVS options (except for the Geometry Viewer option — see above). (Does not start an AVS session.)

–version

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

AVS Startup File

When it begins execution, AVS searches for a *startup file*, which specifies the locations of various directories. AVS looks for the following files, in the order listed:

./.avsrc	(current directory)
$HOME/.avsrc	(home directory)
/usr/avs/runtime/avsrc	(system directory)

At most *one* of these startup files is read. If AVS finds one of them, it ignores the others. A */usr/avs/runtime/avsrc* file is included on Stardent's AVS distribution tape.

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:

```
NetworkWindow        867x567+407+2
NetworkDirectory     /usr/johnp/avs/nets
DataDirectory        /usr/johnp/avs/data
DialDevice           /dev/tty02
```

In most cases, 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.

The AVS startup file keywords are as follows:

DataDirectory

(command-line equivalent: **–data**) Specifies the directory in which the various "read data" modules (**read field**, **read geometry**, etc.) initially will look for data files.

DialDevice

(command-line equivalent: none) Indicates the serial communications port to which a DIGIT dialbox device is attached.

This entry corresponds to the environment variable DIALS; if DIALS is set, the startup file entry (if any) is ignored.

ImageScrollbars

(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.)

NetworkDirectory

(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

(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.

Path

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

SaveMessageLog

(command-line equivalent: none) It 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).

SpaceballDevice

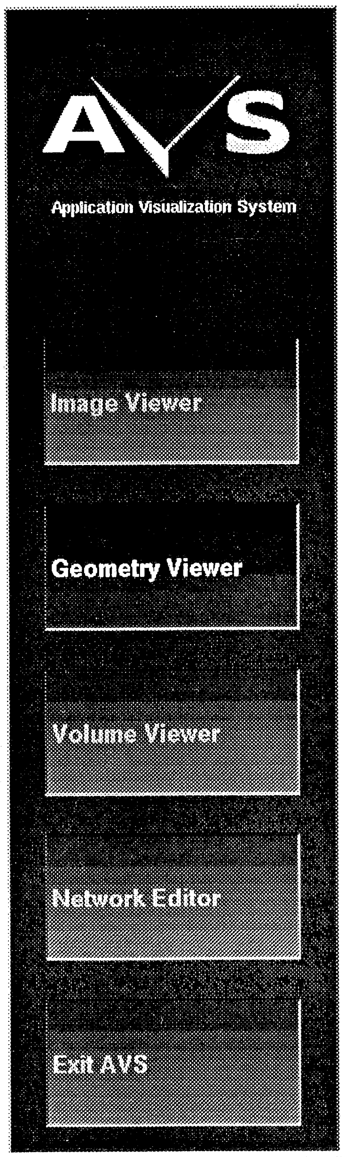
(command-line equivalent: none) Indicates the serial communications port to which a Spaceball device is attached.

This entry corresponds to the environment variable SPACEBALL; if SPACEBALL is set, the startup file entry (if any) is ignored.

The Main Menu — AVS's Subsystems

When you start AVS, the main menu appears within a control panel along the left edge of the screen (Figure 2-1).

Figure 2-1. AVS Main Menu



Each of the subsystems has its own control panel (usually, along the left edge of the screen). When you click the **Exit** button at the top of a subsystem's control panel, the AVS main menu reappears.

Switching Among the Subsystems

In general, you cannot go directly from one AVS subsystem to another. Instead, you must click **Exit** to return to the AVS main menu, then enter another subsystem. Keep in mind that when you exit a subsystem, your work is *not* automatically saved. (A dialog box will appear to remind you of this fact.) If you wish to preserve your work for later use, be sure to use the appropriate function first (e.g. **Write Network** in the Network Editor subsystem).

There are some exceptions to this rule: for instance, you can go directly to the Geometry Viewer subsystem from any of the others, and return again.

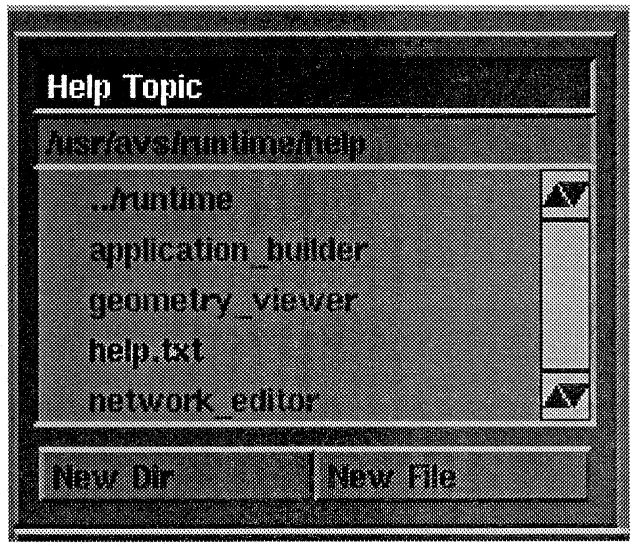
Using On-Line Help

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 Browser* window to appear (Figure 2-2):

Figure 2-2. A Help Browser



The browser 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 viewing area, which has scroll bars to allow easy perusal of the help texts:

❑ 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 the 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 browser 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 browser 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 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

Each computational module in AVS is represented onscreen 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 Browser like the one described above, displaying the complete manual page for that module.

Shell-Level Help

The manual pages are also available through the UNIX shell command **man**(1). For example, typing "man 6 colorize" at a shell prompt displays the manual page for AVS's **colorize** module.

Stardent

3

The Image Viewer Subsystem

Table of Contents

Chapter 3
The Image Viewer Subsystem

The AVS *Image Viewer* subsystem provides access to a set of visualization networks specifically created for manipulating 2D images. You don't need to do any network construction at all — just select the network you want from the Image Viewer menu system.

The Image Viewer implements these powerful features:

Fast Start
Since the networks have already been created, you can transform your data into a visualization display quickly. In most cases, just a handful of menu-choice mouse clicks is all it takes.

Replication and Comparison
It is easy to view different data sets side-by-side, using the same visualization technique. Similarly, it is easy to view the *same* data set in several windows, each using different input parameter settings.

Resource Sharing among Networks
When you invoke several networks to perform different types of visualization (or the same type on different data sets), the Image Viewer can multiplex resources among the networks. For example, a single mouse click can take the data set from one network and "plug it into" another network, too. This capability extends to the networks' display windows for visualization output and their colormaps, too.

Connections to Other AVS Subsystems
It is easy to make the transition from using the Image Viewer's pre-existing visualization networks to creating your own. At any time, you can pop up a window that shows the network being executed. Moreover, you can instantly switch to the Network Editor subsystem, in order to refine or extend the network.

In addition, you can instantly switch from the Image Viewer to the Geometry Viewer subsystem.

Overview of Image Viewer Usage

This section presents a quick overview of the Image Viewer's typical pattern of usage. It assumes that your data is already in the AVS *image* format that is directly readable by AVS data modules. Be sure to consult the *Image Data File Format* section in Chapter 1 before attempting to use your data with the Image Viewer.

Typical usage of this subsystem includes the following steps:

1. **Entering the Image Viewer Subsystem.** You can enter the Image Viewer either from the UNIX shell or from the AVS main menu.

2. **Preprocessing the Data.** In some cases, you may want to perform some transformation on your input data (e.g. extracting a subset) before visualizing it. This step reads a data file and creates a new file containing the transformed data.

3. **Selecting a Visualization Technique.** The Image Viewer offers a wide variety of techniques for processing images. When you select one by clicking its menu choice, AVS automatically loads a network that implements the technique. (An empty *display window* appears, in which the visualization output will be drawn later.)

4. **Selecting the Data.** All the image viewer networks begin by reading an *image* format data file from disk. When you select a data file (perhaps one you created in step #2 above), the data flows through the network and an

image appears in the display window.

5. **Adjusting the Parameters.** In general, each module in the underlying network has input parameters, whose values you can adjust with on-screen *control widgets*. As you work with these dials, sliders, etc., the visualization image in the display window changes accordingly.

The sections below discuss these steps in more detail. Since this pattern of usage is not the only way in which you can use the Image Viewer, there is also a section that discusses features that are not covered in the above "recipe".

Entering and Leaving the Image Viewer

There are two ways to enter the Image Viewer subsystem:

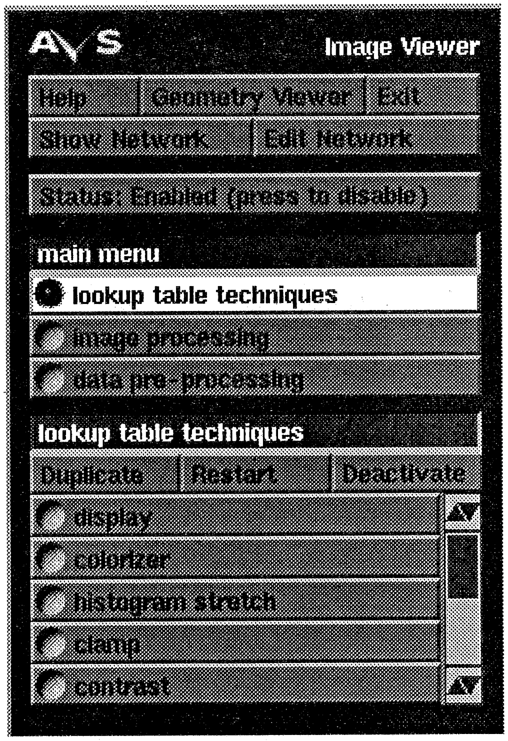
From the AVS main menu
Click the "Image Viewer" choice on the AVS main menu.

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

```
avs -image
```

The Image Viewer starts by displaying its control panel along the left edge of the screen (Figure 3-1).

Figure 3-1. Image Viewer Control Panel



To leave the Image Viewer, click the **Exit** button at the top of the control panel. Control reverts either to the AVS main menu or to the UNIX shell, depending on how you entered the subsystem.

Using the Data Preprocessors

In some situations, you may want to preprocess your data before visualizing it. With the Image Viewer, the strategy is to take a data file, preprocess it, and create another data file. The second file can then be used as input to one or more of the visualization techniques.

(In the Network Editor subsystem, you can create networks that preprocess data "on the fly", so that you need not create additional disk files containing the preprocessed data.)

To start, click **Data Preprocessors** on the Image Viewer top-level menu. Each of the preprocessing functions provides access to an AVS filter module with the same name:

crop Extracts a range of elements from a field. This is useful for discarding unwanted data. You can also use this to perform a quick analysis on a subset of the data, in preparation for a more time-consuming analysis on the complete data set.

downsize Reduces the size of a data set by sampling every *n*th element in each dimension. Like **crop**, it is useful for performing quick analyses on subsets.

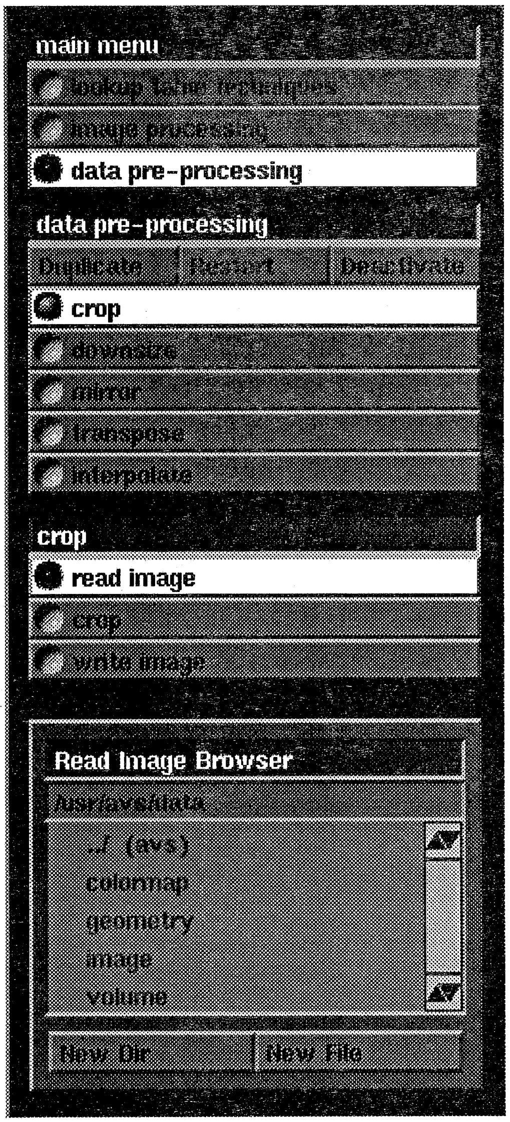
mirror Produces a mirror image of a data set.

transpose Exchanges the dimensions in a 2D or 3D data set.

interpolate Reduces the size of a 2D or 3D data set by performing an interpolation.

When you select one of these functions, a three-choice menu appears in the control panel, e.g.:

Figure 3-2. Crop Menu



The **Read Image** choice (automatically selected initially) pops up a File Browser window, in which you can specify the data set to be preprocessed.

Click the second choice (e.g. **crop**) to display a set of control widgets that interactively control the preprocessing operation (e.g. to select which part of the data set is to be cropped). As you work the controls, the potential results of the preprocessing operation are displayed in a display window.

When you have adjusted the controls to your liking, click the **Write Image** choice to pop up another File Browser. Specify a file in which to store the preprocessed data.

NOTE *The File Browser window and the control widgets are designed for ease of use. If you have any problems figuring out how to use them, see **Controlling the Execution of a Network** in the Network Editor chapter.*

Selecting Visualization Techniques

The Image Viewer's visualization techniques are organized into two submenus:

Lookup Table Techniques
display
colorizer
histogram stretch
clamp
contrast

Image Processing
gradient shade
image transform
3D mesh

Initially, the **Lookup Table Techniques** category is selected, so its five choices are visible. You can click the **Image Processing** button in the Image Viewer main menu to make the three choices in this category visible. To select one of the visualization techniques, just click on it.

Selecting a technique invokes an AVS network that includes several modules. In general, however, there is a single "main component" at the heart of the network — the module that does the "real work". For example, the **colorizer** and **histogram stretch** techniques invoke networks that are essentially packages for the like-named modules.

Getting Help on a Technique

The image viewing techniques are summarized in the sections below. For more details, consult the manual page for one or more of the modules that implement the technique. To view this manual page:

- ☐ Invoke the technique.
- ☐ Click the **Show Network** button at the top of the control panel to display the network that implements the technique.
- ☐ Use the middle or right mouse button to click the small square in one of the network's module icons. This brings up the Module Editor window.
- ☐ Click the Show **Module Documentation** button.

Lookup Table Techniques

display
Displays its input data, an AVS *image*, unchanged.

colorizer
The input data is passed through a color lookup table, whose contents are controlled by a colormap control widget.

histogram stretch
Equalizes the distribution of values between the parameter-controlled *min* and *max* values. Values outside these bounds are left unchanged.

clamp
Establishes parameter-controlled *min* and *max* values for the data. Pixel values below the *min* value are set to *min*; pixels above the *max* value are set to *max*.

contrast
Stretches range of the input data linearly between parameter-controlled *min*

and *max* output values. Can be used to increase the contrast of an image, or to invert an image.

Image Processing Techniques

gradient shade
First determines the gradient of the "surface" of the image at each pixel. (This is done by taking the ΔX and ΔY values between neighboring pixels, and using a parameter-controlled ΔZ value.) Then shades the pixel values according to the gradient. This produces an image with 3D characteristics.

image transform
Maps the input data onto a rectangle that can be arbitrarily transformed in three dimensions.

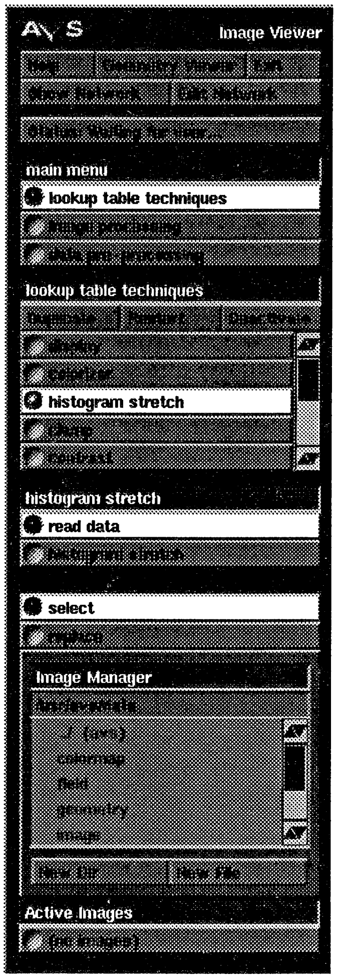
3D mesh
Creates a 3D mesh from the input data, with X and Y values determined by the pixel ordering, and Z values proportional to the value of the pixel.

Running the Network

This section discusses in more detail how you work with any one of the Image Viewer's visualization techniques. (You can also use several techniques at once — this subject is discussed in a subsequent section.)

When you select one of the visualization techniques, a choice menu ("radio buttons") appears in the control panel and an empty display window is created to the right of the control panel. For example, if you select **histogram stretch**, the control panel appears as in Figure 3-3.

Figure 3-3. Control Panel for 'histogram stretch' Technique



The choice menu has an entry for each module in the network that has input parameters. You can click the choices to switch back and forth among the *pages* of control widgets. (The empty display window created when you select a technique is not represented in the choice menu. Its controls are in a pulldown menu, accessed via the small square in the window's title bar — see next section.)

NOTE *If you wish to see the network that implements a technique, click the* ***Show Network*** *button at the top of the control panel. This creates a window showing the network. You may want to leave this window open as you use the visualization technique. Each module icon flashes as the module executes; if a module crashes, its icon turns black. If you deactivate or switch visualization techniques (described later), this window is updated accordingly.*

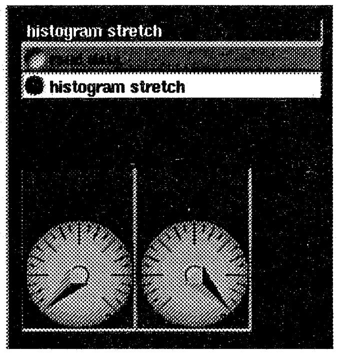
Leaving the ***Show Network*** *window open also makes it convenient to access the on-line help screens for the network's modules, as described in the* ***Getting Help on a Technique*** *section above.*

Typically, you start by selecting the data to be visualized. As shown in Figure 3-3, the **read data** choice is initially selected (corresponding to the **Image Manager** module in the network), and its File Browser control widget appears below the menu. When you specify an image to be read, it flows through the network and the image appears in the display window. This window automatically resizes itself to accommodate the image you select.

Note that whenever data is flowing through the network, the status bar near the top of the control panel turns red and displays a message that describes the current activity. If you wish to disable this feature, just click on status bar with any mouse button; another click reenables this feature.

The next typical step is to exercise the controls of the module that is the main component of the network. For the **histogram stretch** technique, click on the **histogram** choice to bring up its page of control widgets (Figure 3-4):

Figure 3-4. Control Widgets for 'histogram'

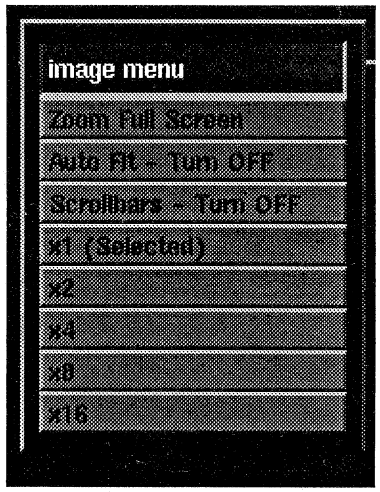


As you adjust the dials (or other control widgets) with the mouse, the image in the display window changes accordingly. If you need guidance using any of the control widgets, consult the on-line manual page for the associated module.

Working with the Display Window

The display window for a visualization technique is created by the last module in the network, either **display image** or **display pixmap**. You can move, resize, and iconify this window using the X Window System window manager. There are also AVS-level controls for this window; they are in a pulldown menu attached to the small square in the window's title bar (Figure 3-5):

Figure 3-5. Display Window Pulldown Menu



The display window is designed for flexibility: it automatically resizes itself when the image size changes. This can occur when you explicitly magnify an image or "zoom" it to full-screen size. It can also occur when you switch data sets to produce a different output image.

There may be cases in which the image does not exactly fit inside the display window. When the image is too big for the window, scrollbars automatically appear along the right and bottom edges. This allows you to move the image around within the window. You can also move the image simply by grabbing-and-dragging, using any mouse button.

(Both the scrollbars and *AutoFit* features can be turned off — see below.)

The pulldown menu choices are summarized here. For more details, see the **display image** manual page.

Zoom Full Screen
Resizes the window to fill the square working area of the screen (approximately 1024 x 1024 pixels).

Resize to Fit Image
Resizes the window to fit the image exactly at the current magnification. (You need to use this only when the *AutoFit* feature is disabled.)

Unzoom
Resizes and moves the window to return to its location before a **Zoom Full Screen** or a **Resize to Fit Image**.

AutoFit - Turn On/Off
Toggles the automatic fitting of the display window size to its image.

Scrollbars - Turn On/Off
Toggles whether scrollbars will appear along the right and bottom edges of the display window when the image is too big for the window.

x1, x2, x4, ...
Specifies a magnification for the image. The current magnification is marked as "(selected)".

Deactivating or Restarting a Technique

Just above the top-level techniques menu are three buttons: **Duplicate**, **Restart**, and **Deactivate**. When you are finished using a technique, but you don't want to exit the Image Viewer, click **Deactivate** to close down the currently active network. All its control panels and display windows disappear.

If you don't plan to return to a technique, it is advisable to deactivate it. This saves memory usage and UNIX processes, both of which are finite resources.

NOTE *When you deactivate a technique, no other technique becomes current (no technique gets the light blue ball). You must make another technique current by clicking on its submenu entry.*

Clicking **Restart** returns the current network to its initial state: no input data selected, all controls in their initial positions, and an empty display window.

The **Restart** button can be very useful in situations where one of the network's modules fails to load, hangs, or crashes. Pressing this button does *not* allow you to recover your work, however.

(The **Duplicate** button is discussed in the next section.)

Using More Than One Technique

One of the most powerful and flexible aspects of the Image Viewer is that you can invoke several visualization techniques at once. For instance, you might select **colorizer** and then select **contrast**. Note that all selected techniques are marked with a blue ball. One of the selected techniques is *current*, and is marked with a *light blue* ball.

When two or more techniques are selected, each can have its own input data set. But *all* the data sets that you have read in are accessible by *all* the techniques. The "Active Images" list includes every data set that you have read. With any technique, you can switch to another active data set simply by clicking on it in this list. This makes it easy to apply several different techniques to the same data set.

To switch back and forth among two or more active techniques, just click on the technique name in the submenu. Remember that the light blue ball always indicates the currently selected technique. There is no problem with having techniques from different submenus active at the same time.

Duplicating a Technique

In some situations, you may want to invoke the same technique more than once. For example, you might want to create three windows side by side, each showing the same image, but with different **contrast** settings. Make sure that the technique you wish to invoke more than once is currently selected (is marked with a light blue ball). Then click the **Duplicate** button once or more. For each click, a copy of the technique is entered in the submenu. (A digit is appended to the name, so that you can distinguish the different copies.) You can now switch back and forth among the copies, and share data among them, just as described above.

Example

The following sequence shows how you might use three different images with two of the visualization techniques:

1. Click **colorizer** to invoke that technique.
2. Use the **read image** file browser to load the first image from file *mandrill.x*. The image appears in the display window and the name "mandrill" is placed in the Active Images list at the bottom of the control panel
3. Click **contrast** to invoke that technique.
4. Use the **read image** file browser to load a second image from file *marble.x*. Since the **Select** button just above the file browser is highlighted, this image is *added* to the Active Images list. The image appears in a second display window; there are now two display windows, one showing a head, one showing a foot.
5. Click **colorizer** again to switch back to the control panel for that technique. (Note how the light blue ball always indicates the currently selected technique.) The Active Images list is the same as that for **contrast**, containing both *marble* (currently selected) and *mandrill.*
6. Click *marble* to make it the current image for the **colorizer**. Now, both display windows show the same image.
7. Click the **Replace** button just above the file browser, then select the third image file, *stardent.x*. In the Active Images list, *stardent* replaces the current image, *marble*, instead of being added to the list. Since *marble* had

been the current image for both techniques, *stardent* takes its place in both display windows.

Additional Image Viewer Features

The following function buttons appear at the top of the Image Viewer control panel:

Help
Pops up a Help Browser, allowing you to view a collection of help files for the Image Viewer. See Chapter 2 for information on using the Help Browser.

Show Network
Click this button after selecting a visualization technique to pop up a window that shows the AVS network used to implement the technique. Click this button again to close the window.

Edit Network
Clicking this button goes one step further than **Show Network**: it actually invokes the Network Editor on the network that implements the currently-selected technique.

Geometry Viewer
Switches immediately to the Geometry Viewer subsystem.

Stardent

4

The Volume Viewer Subsystem

Table of Contents

Chapter 4
The Volume Viewer Subsystem

The AVS *Volume Viewer* subsystem provides access to a set of visualization networks specifically created for 3D data sets. The *volume data file* format is described in Chapter 1.

This subsystem works in exactly the same way as the Image Viewer, described in the preceding chapter. The only difference is that the Volume Viewer offers a different set of visualization techniques:

Imaging Techniques
orthogonal slice
slice with gradient

Geometric Techniques
arbitrary slice
bubble viz
dot surface
isosurface tiler
volume bounds

Volume Rendering
alpha blend
shaded alpha blend
vbuffer

Data Preprocessing (same set as in Image Viewer)
crop
downsize
mirror
transpose
interpolate

These techniques are described in the sections below.

Imaging Techniques

orthogonal slice

Takes a 2D slice from the 3D input data array, by holding the array index in one dimension constant and letting the other indices vary. The resulting 2D array of values is converted to colors by passing it through a colormap.

slice with gradient

Same as **orthogonal slice**, but with additional processing of the output image. The gradient of the "surface" of the image is determined at each pixel. (This is done by taking the ΔX and ΔY values between neighboring pixels, and using a parameter-controlled ΔZ value.) Then the pixel values are shaded according to the gradient. This produces an image with 3D characteristics.

Geometric Techniques

arbitrary slice

Maps the input data to a uniform 3D lattice, then slices through this volume with a plane. The data values at the (approximated) intersection points are used to calculate a 3D mesh. The orientation of the slice plane is parameter-controlled, allowing you to position it arbitrarily within the volume. Also parameter-controlled are the resolution of the grid of points that define the slice plane and the sampling technique: *nearest neighbor* or *trilinear interpolation*.

bubble viz

Represents each data value in the volume as a (colored) sphere.

dot surface

Creates an isosurface — all the elements that have a parameter-controlled data value — representing this surface as a mesh of dots.

isosurface tiler

The input data is processed with a "marching-cubes" algorithm to create an isosurface defined by a parameter-controlled data value.

volume bounds

Draws lines that show the 3D bounding box for the input data.

Volume Rendering

alpha blend

The 3D input data is converted to a 2D image: first, the data values are converted to colors by passing them through a colormap; then, the colors are blended from back to front using the opacity field of the color value.

shaded alpha blend

Same as **alpha blend**, but with additional gradient shading, as described under **slice with gradient**.

vbuffer

Performs volumetric rendering of a 3D uniform scalar field, using interpolation, color shading, and transparency-processing algorithms.

Data Preprocessing

These same techniques are also available in the Image Viewer subsystem.

crop

Extracts a range of elements from a field. This is useful for discarding unwanted data. You can also use this technique to perform a quick analysis on a subset of the data, in preparation for a more time-consuming analysis on the complete data set.

downsize

Reduces the size of a data set by sampling every *n*th element in each dimension. Like **crop**, it is useful for performing quick analyses on subsets

mirror

Produces a mirror image of a data set.

transpose

Exchanges the dimensions in a 2D or 3D data set.

interpolate

Reduces the size of a 2D or 3D data set by performing an interpolation.

Stardent

5

The Geometry Viewer Subsystem

Table of Contents

Chapter 5
The Geometry Viewer Subsystem

The AVS Geometry Viewer subsystem is an interactive tool with which you can manipulate and view one or more 3D objects. This subsystem can be invoked in several ways:

From the shell directly
The following command line invokes the Geometry Viewer automatically when AVS starts execution:

```
avs -geometry
```

See Chapter 2 for additional command-line options that affect the manner in which the Geometry Viewer is invoked.

From the main menu
You can start the Geometry Viewer from the AVS main menu.

From another subsystem
The Geometry Viewer is the one subsystem to which there is direct access from all the other subsystems. There is a **Geometry Viewer** button at the top of each subsystem's control panel.

In a network
You can include the **read geometry** module in an AVS network.

Whichever way you invoke the Geometry Viewer, you work with objects that are represented in AVS's *geom* format. Such objects must have previously been created and stored in a file, using the AVS *libgeom* programming library. (If you invoke the **render geometry** module in a network, it can input geometries created and/or revised by upstream modules.) AVS includes a number of utility programs that read standard data formats and create *geom*-format files. The standard formats include Wavefront, Movie.BYU, Brookhaven Protein Data Bank, and others. (See the *Geometry Conversion Programs* appendix for details.)

The following sections provide an overview of the way the Geometry Viewer works.

Scenes: Objects, Lights, Cameras

Using the Geometry Viewer, you can work with one or more *scenes*. Each scene consists of:

- ❑ A collection of 3D objects, assembled into a single coordinate system (*world coordinate space*). Objects have attributes, such as surface color, various light reflectance characteristics, and a rendering method. You can selectively hide objects, 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 the 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 they display lines (you can turn depth-cueing and Z-buffering of lines on and off).

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* program — they are just like any other windows.

NOTE *If you invoke the Geometry Viewer as the* **read geometry** *module in an AVS network, you may want to integrate the view window(s) into the network's overall user interface. For details, see* ***Including Output Windows in a Reorganized Layout*** *in the* ***Network Editor*** *chapter.*

Within each view window, you can translate, rotate, and scale objects, using the system mouse. (You also can use a dialbox or Spaceball as the manipulation device.) If a scene has several view windows (cameras), then the objects move in all of them. Likewise, changing the lighting affects all the windows that belong to a scene.

Data Formats

The following sections also present an overview of the Geometry Viewer, but with a different focus — the formats in which data is represented.

Geometries

The Geometry Viewer's fundamental data structure is the *geometry*: 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 and/or normal for each vertex.)

Each geometry is stored in its own file, with a *.geom* extension. AVS includes a small library of *.geom* files. It also includes a *data filter* facility, with which you can create your own *.geom* files, either from scratch or from geometric data in a number of commonly-used formats (e.g. Movie.BYU). Note that there is no way to define a new geometry or modify an existing one from within the viewing application.

Objects

Using the Geometry Viewer, you can start with a simple object and adjust your view of it by specifying various attributes, or *properties*:

❑ The position and orientation in 3D space.

❑ The surface color.

❑ The way in which the surface reflects light (including specular highlights)

❑ The rendering method to be used in drawing the geometry.

A geometry that is customized with these property specifications is called an *object*. Each object is stored in its own file, with a *.obj* extension. This is an ASCII-format file, expressed in the *Geometry Viewer Script Language*. An object file is created when you "customize" a geometry, then save it with the **Save Object** function. (You can also store the properties separately, in a file with a *.prop* extension.)

You can also create equivalent object files with a text editor, using the Script Language directly (see Appendix B).

You can also create hierarchical *composite objects*, which include several or many geometries. You can specify the properties listed above for the individual-geometry level or at various levels of the hierarchy. If a geometry does not have its own properties set, it *inherits* them from the next level up (or some higher level).

With the Geometry Viewer, you create only a two-level hierarchy — a *top-level object* with a number of objects as its children. The Script Language gives you more flexibility (at the expense of interactivity). A script can define multiple-level hierarchies, using the **group** command. See the *Script Language* chapter for details.

Scenes

Just as you build up basic geometries into objects, you compose objects into *scenes*. Like objects, scenes are represented in the Script Language. You can build a scene interactively with the viewing application, adding and manipulating the positions of several objects, adding and positioning lights, and defining one or more cameras to view what you've built. When you select **Save Scene**, a script is written to a file with a *.scene* extension. You can also create equivalent scene files with a text editor, using the Script Language directly.

NOTE *The binary-format **geometry** files, which specify the vertices of objects, are the basic building blocks for the Geometry Viewer. These are the **only** files in which geometric definitions are stored. The ASCII-format object and scene files, written in the Script Language, include **references** to geometry files, along with other specifications. This means you must be careful not to disturb geometry files that are used as building blocks for objects and scenes. If, for instance, you rename or move the file **teapot.geom**, it will invalidate all objects and scenes that include the teapot defined therein.*

File Type Summary

The following table summarizes the types of files that the Geometry Viewer can read directly.

TABLE 5-1. Geometry Viewer File Types

Type	Filename Extension	Data Format	Contents of File
Geometry	*.geom*	binary	Describes a single geometric object (simple or complex). Can include per-vertex data: normals and/or colors.
Property	*.prop*	ASCII	Specifies a set of surface attributes, including color and light-reflectance characteristics.
Object	*.obj*	ASCII	Specifies the names of one or more geometry files, along with surface attributes and rendering methods. Can also define a hierarchy that includes geometries and other objects.
Scene	*.scene*	ASCII	Specifies objects (as in a *.obj* file), along with light(s) and camera(s).

Menu Choices, Sliders, and Function Keys

You can use any mouse button to make menu choices. A single click of the mouse button anywhere within a choice's rectangle makes the selection.

The Edit Property window includes a number of sliders, as do the **Light** and **Camera** menu selection areas. You can move a slider either by dragging it with any mouse button, or by clicking once at the place to which you want the slider to move.

AVS makes use of several of the keyboard's function keys. This is described under *Function Key Usage* below.

Transformations and the Transform Selection Area

One of the most powerful features of the Geometry Viewer is that it allows you to interactively *transform* aspects of the scenes you create. For example, you can change the positions and sizes of objects, the positions and types of lights, the placement of the camera in the scene, and so on. You control these transformations with the mouse. At any moment, the mouse is "attached" to the *current transformable*, which is one of the following:

- ❑ An object (possibly a composite object).
- ❑ A light
- ❑ A camera
- ❑ A texture mapping (see *Edit Texture* below).

The Transform Selection menu, in the upper part of the control panel, allows you to change the type of the current transformable. A one-line label just below the Transform Selection menu indicates what the current transformable is. For example:

top
The top-level object, typically consisting of several individual objects retrieved from disk storage with **Read Object**.

Directional 4
Light #4 on the lighting panel, currently specified as being a directional light.

AVS-3
The fourth view window you've created during the current AVS session. (The first one is named AVS-0.) You can use the startup file to define your own names for the windows that AVS creates. See *Startup File* above. You can also specify view window names when you create a *.scene* file with the Geometry Viewer Script Language.

jet
An object retrieved from file *jet.geom*.

A particular mouse button has approximately the same function whether you are transforming an object, a camera, or a light. Since the exact functions vary, a full (but somewhat redundant) listing is contained in the following sections.

NOTE *It is also possible to have the Spatial Systems Spaceball™ control the geometric transformations in a view window. See* ***Spaceball Usage*** *below.*

Transforming Objects

To transform objects, click the **Transform Object** button (or, equivalently, press function key **F1**). 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 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.

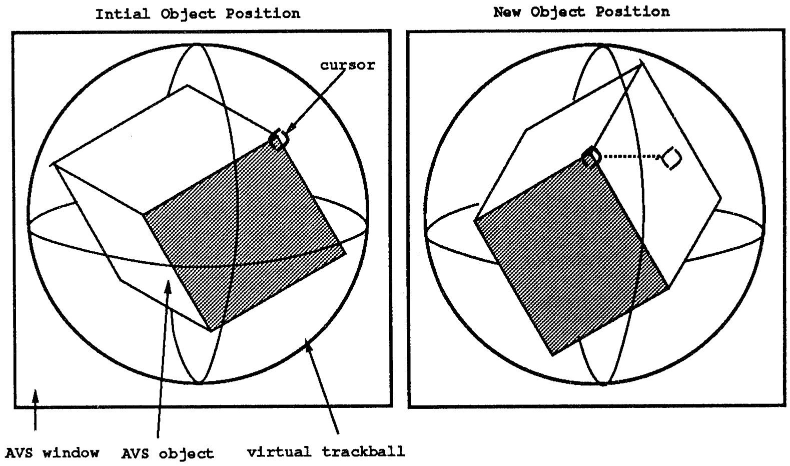
Three clicks: Expands the selection to the entire top-level object, which includes the jet and, perhaps, several other objects.

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 trackball 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.

Figure 5-1. Rotating an Object with the Virtual Trackball



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).

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. (The location of the clipping planes in world coordinates is not currently controllable by the user.)

If the scene is not drawn in perspective (see ***Cameras*** section below), there will be no change in the object's appearance until it gets clipped.

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.

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 16 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).

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.

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.

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.

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.

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.

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.

Transforming Textures

To transform the way that an image is "texture-mapped" to the surface of an object, click the **Transform Texture** button (or, equivalently, press function key **F4**). 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).

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.

Middle Mouse with SHIFT button held down
Scales the texture map.

Additional Transformations

Just below the Current Object Indicator are two additional buttons:

Reset
Restores the current transformable (object, light, or camera) to its default position. It does *not* also reset the color, surface properties, or rendering mode of the object.

Normalize
Scales the current object so that it fills its view window.

Function Key Usage

Most of 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.

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 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.

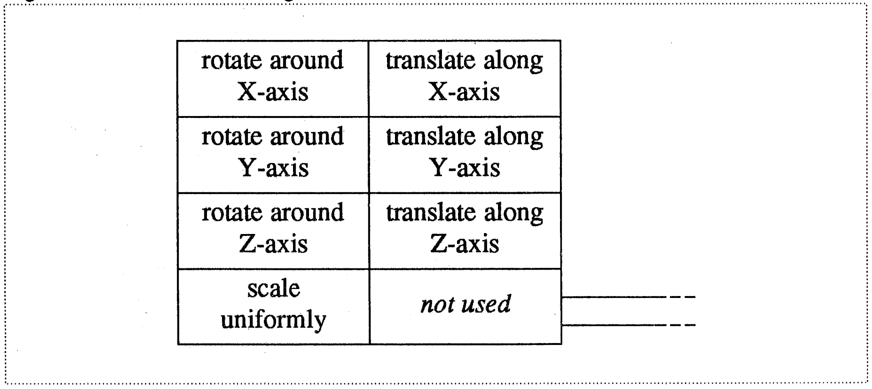
F6 Performs a **Reset**, returning the current object, light, or camera to its original position and orientation.

F7 Performs a **Normalize**, resizing the current object so that it fills the current window.

Dial Box Usage

The following diagram shows how the DIGIT dial box can be used as an alternative to the mouse for moving objects, lights, and cameras. To enable the dial box, use the **–geometry –dials** *device-name* option when starting AVS.

Figure 5-2. Dial Box Usage



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*

Spaceball Usage

To enable the Spaceball, use the **–geometry –spaceball** *device-name* option when starting AVS. The spaceball should beep as AVS starts up.

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.

Occasionally, the Spaceball loses track of its "at rest" position. When this occurs, press the **8** function key to recalibrate the ball.

Geometry Viewer Menu Reference

The Menu Selection part of the Geometry Viewer control panel provides access to most functions for creating 3D "scenes". Each scene includes a combination of objects, lights, and cameras. (Each camera you define shows the scene in a different window.) The Menu Selection area 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:

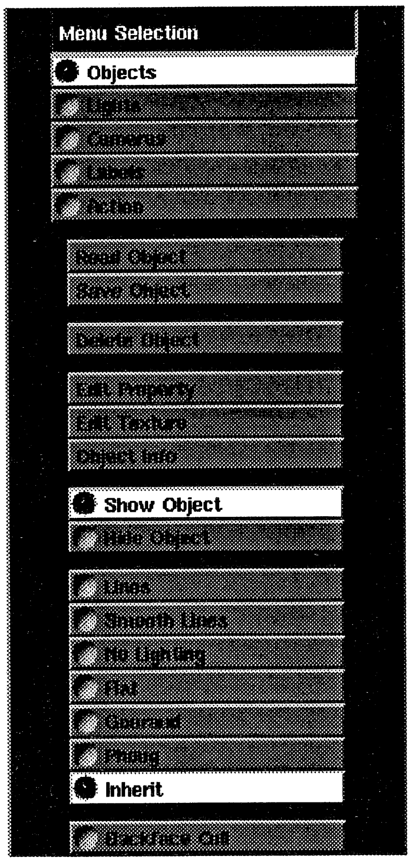
Objects
Lights
Cameras
Labels
Action

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 as follows:

Figure 5-3. Objects Menu Selections

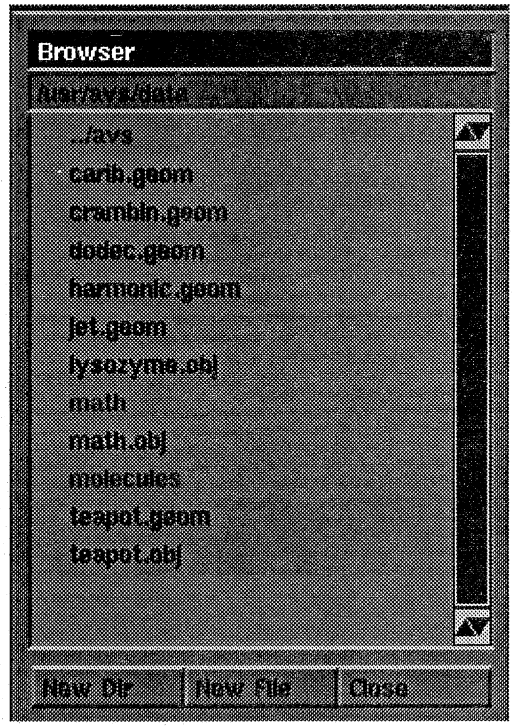


Read Object

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:

Figure 5-4. The File Browser



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 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:

❑ 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.

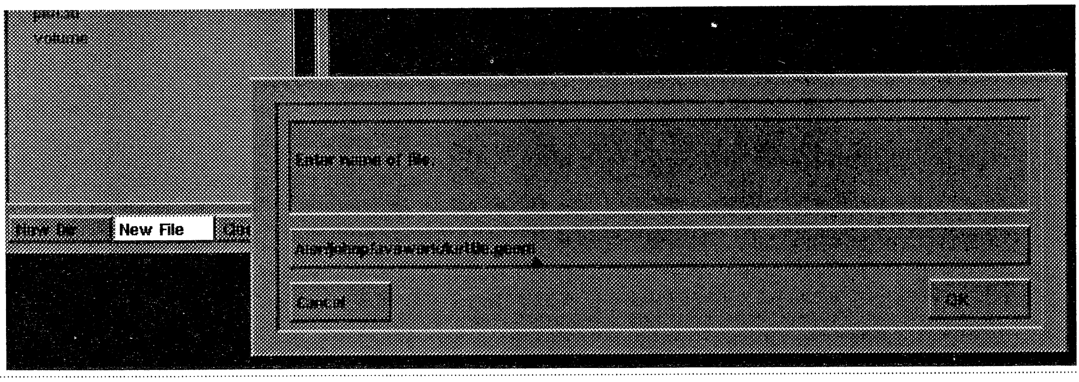
Selecting one of the red (directory) entries changes the working directory. The names of the Geometry Viewer object files in that directory are displayed, along with the names of any subdirectories.

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. (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 5-5. Entering a Filename



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 RETURN key or click **OK** with the mouse.

NOTE *File browser windows are also used in many places throughout the other AVS subsystems.*

Save Object

This function saves the current object (shown by the Current Object Indicator) in a *.obj* file. 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).

NOTE *The* ***.obj*** *file contains references to one or more* ***.geom*** *files (which define simple objects). That is, the* ***.obj*** *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.*

A window pops up so that you can type a filename. Be sure the mouse cursor is in the one-line text-entry area before you start typing the filename.

You don't need to type the *.obj* extension when you enter a filename — AVS adds this extension automatically (unless you type it yourself). To finish entering the filename, press the RETURN 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**.

AVS actually uses a special Script Language to create the object file (see Appendix B).

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.

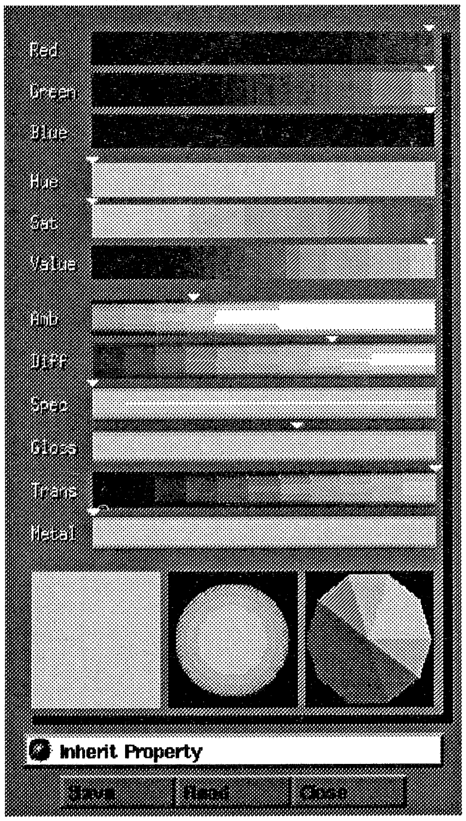
Since there is no way to "undo" deleting an object, you may want to perform a **Save Object** before deleting something that might be useful later on.

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.

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 5-6. The Edit Property Window



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:

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. White is specified by a zero saturation (slider all the way to the left). Black is specified by a zero value.

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.

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.
- ❑ 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 restore them, just click on **Inherit** again. For longer-term storage of properties settings, use the **Save** and **Read** buttons.

Edit Texture

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 three-dimensional object to which 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 maps the remainder of the texture to the object surface by means of linear interpolation between vertices.

Steps in Using AVS Texture Mapping. Here's a procedure for doing 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.
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*.

This procedure causes the texture to appear on the object. Whenever you transform the object (translate, rotate, scale), the texture transforms accordingly. 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 **render geometry** module. This module takes an image as an optional input. Clicking **Set Dynamic Texture** causes this image to be selected as the texture to be mapped to the current object.

Object Info

Clicking this button displays a window of information pertaining to the current object:

- ❑ 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

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.

Lines

The object is drawn as a wire-frame, using non-anti-aliased lines.

Smooth Lines

The object is drawn as a wire-frame, using anti-aliased lines.

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 drawn using filled polygons, each of which is flat shaded.

Gouraud Shading

The object is drawn using filled polygons, each of which is Gouraud shaded.

Phong Shading

The object is drawn using filled polygons, each of which is Phong shaded.

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 Culling

Causes certain faces of the object not to be drawn. Successive mouse clicks in the menu box cause the slot to be filled with a dark blue ball, a light blue ball, or nothing:

Dark blue ball
Indicates backface culling: surfaces of the object whose surface normals point *away from* the eyepoint are not drawn.

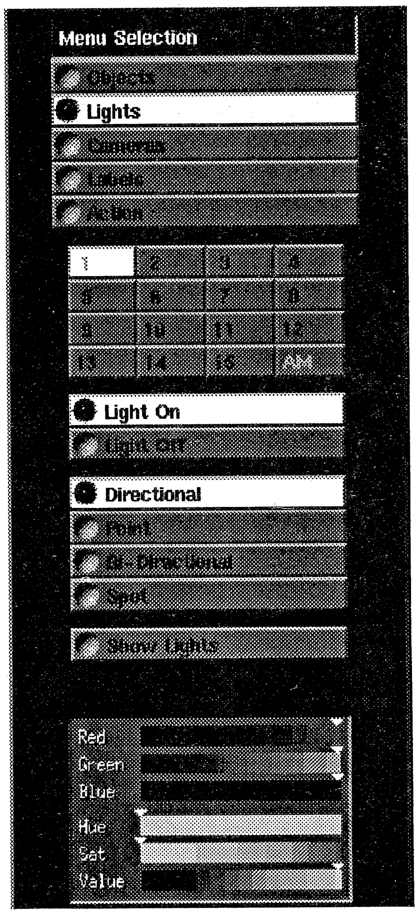
Light blue ball
Indicates frontface culling: surfaces of the object whose surface normals point *toward* the eyepoint are not drawn.

If not selected
No culling takes place; all surfaces of the object are drawn.

Lights

Selecting **Lights** causes the Menu Selection area to appear as follows:

Figure 5-7. Lights Menu Selections



The grid of numbers is a "lighting panel". 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:

- ❑ Ambient light, with white color.
- ❑ 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.

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 â 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 indicate 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:

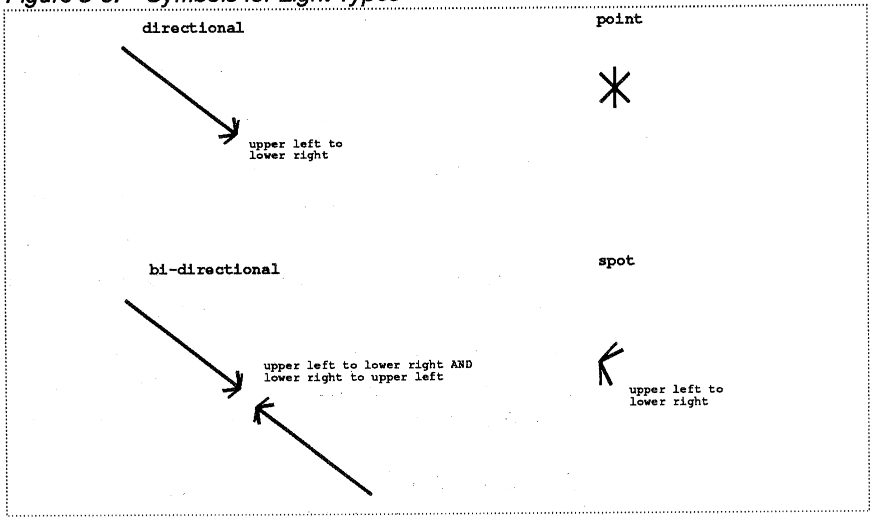
- ❑ **Directional**: (default light type) A light source whose rays all point in the same direction (are parallel). The sun is the canonical directional light source.
- ❑ **Bi-Directional**: A pair of directional light sources that point in exactly opposite directions. This type of light can be used 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.

  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.
- ❑ **Point**: A light source whose rays emanate in all directions from a particular point. A bare light bulb is the canonical point light source.
- ❑ **Spot:** A light source whose rays emanate from a particular point and are restricted to a particular cone. A flashlight is the canonical spot light source.

Show Lights

Displays a symbol for each light source, indicating its (position and) direction. 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 to help indicate its distance from the view plane.

Figure 5-8. Symbols for Light Types



To move (the direction of) a light, make sure that **Lights** is selected in both the Menu Selection and Transform Selection parts of the control panel. Then use the mouse to rotate and/or translate the light. See the **Transform Selection** section for details.

Color of Light

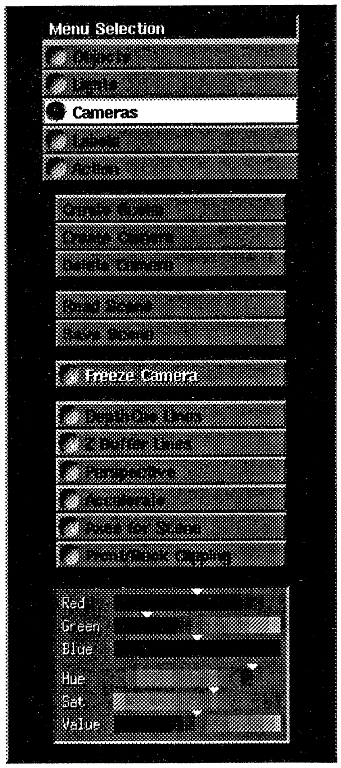
At the bottom of the **Lights** menu selection area, there are RGB and HSV sliders for setting the color of the light source. See *Edit Property* above for an explanation of how to use the sliders.

The default color of all lights is white.

Cameras

Selecting **Cameras** causes the Menu Selection area to appear as follows:

Figure 5-9. Cameras Menu Selections



These 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.

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.

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 *.scene* 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.

See also the *Geometry Viewer Script Language* appendix. This language allows you to define scenes in ASCII files.

NOTE *Similarly to **.obj** files, **.scene** files contain references to objects and geometries, rather than copies of them For this reason, be careful not to disturb **.geom** and **.obj** files that store the "building blocks" for your objects.*

Freeze Camera

Use this function when you have several windows on a scene (**Create Scene Dup**). When you freeze one of the windows, you can still manipulate the objects, lights, and camera in any of the other windows. The changes are reflected only in the unfrozen window(s) -- the image in the frozen window remains the same.

DepthCue Lines

This setting causes lines to "fade away" as they get more distant from the viewing position. If affects objects drawn with Lines or Smooth Lines.

Z Buffer Lines

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).

Perspective

This setting causes the current window to use a *perpective* viewing projection (the default is to use a *parallel* projection). 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.)

Accelerate

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 X-Y-Z axes in the current scene. The right-hand coordinate system indicated by these axes is the "world coordinate system" for the scene.

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. They are so distant, however, that for all practical purposes, no front/back clipping takes place.

Color of Window Background

At the bottom of the **Cameras** menu selection area, there are 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 *Getting Started* chapter for details.

Labels

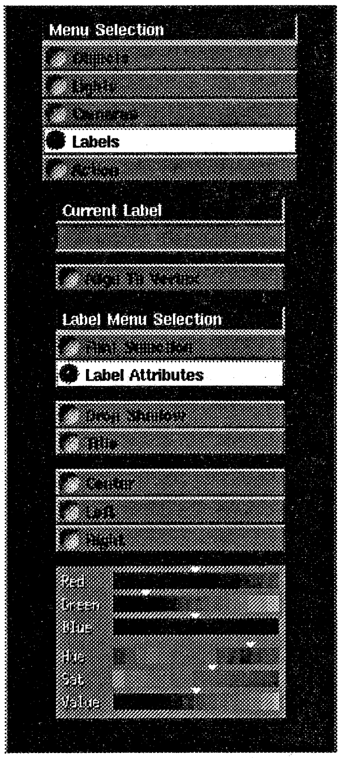
The **Labels** menu selection provides access to the Geometry Viewer's annotation text facility. You can attach one or more *labels* to any object. Each label consists of a single line of text. As you manipulate the object — move it, resize it, temporarily hide it, permanently delete it, etc. — the object's 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 following display:

Figure 5-10. 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.

Be sure to press **Return** 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**.

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.

Attaching a Label to a Vertex. 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.

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 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.

Changing Label Attributes — the 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 the following choices:

Courier
Helvetica
Schoolbook
Times
Charter
Symbol
Selects the X Window System 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.

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. The **xlsfonts**(1) utility program lists all the X Window System fonts available on your machine.

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.

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.

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.

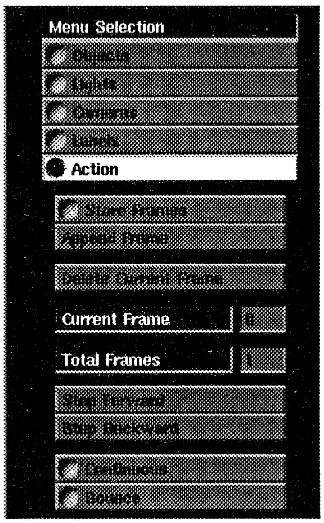
Color Editor
An RGB-HSV color editor, similar to ones used elsewhere by the Geometry Viewer, allows you to specify the color of the label.

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 also define the sequence of geometries as a "cycle" in the Geometry Viewer Script Language. (See the *Script Language* appendix for details.)

When you select **Action**, the following submenu appears:

Figure 5-11. Action Menu Selections



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 output window will be appended to the frame sequence. The **Current Frame** and **Total Frames** counters are updated automatically.

WARNING *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 output window, it will also be added to the sequence.

This feature has some restrictions. Each time you read a new geometry, the existing animation sequence (if any) is discarded and a new sequence is begun. (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.)

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.

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.

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 Foreward** 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**.

Stardent

6

The Network Editor Subsystem

Table of Contents

Chapter 6
The Network Editor Subsystem

AVS's Network Editor subsystem is your main tool for creating, testing, and revising AVS networks. You can also use the Network Editor to refine the user interface to a network, so that others can perform visualization tasks without having to be knowledgeable about network construction.

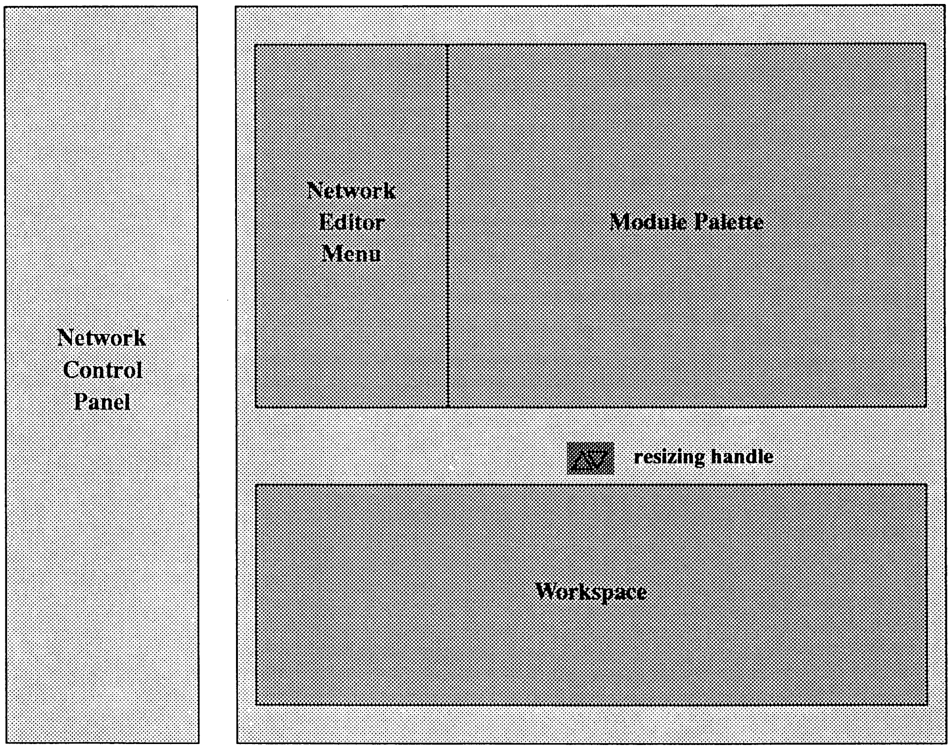
Starting the Network Editor

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 **Exit** 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



Network Construction Window

Getting Help

On-line help for the Network Editor is available via two **Help** buttons. The 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.

The **Help** button in the *Network Construction* window brings up a help browser window. See the *Using On-Line Help* section in the *Starting AVS* chapter for details on using on-line help.

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 close down the Network Editor and return to the AVS main menu. 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 to the Geometry Viewer

In many situations, you'll want to switch to the Geometry Viewer subsystem 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. There are two ways to go directly from the Network Editor to the Geometry Viewer:

- ❏ Click the **Geometry Viewer** button at the top of the Network Construction Window. You can return to the Network Editor by clicking the **Close** button in the Geometry Viewer control panel.
- ❏ If a network includes the **render geometry** module, use the left mouse button to click the small square box in the icon for this module.

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

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 scalar data**) 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

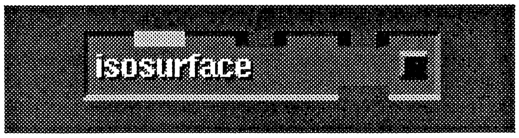
These modules perform the "visualization" step — converting a numerical data set to a description of one or more geometric objects. For instance, the **field to mesh** module creates a 2D surface in 3D space. It does so by interpreting each scalar value of a 2D array as the height of a point above a base plane. The collection of points defines (an approximation to) a 2D surface above the plane.

Renderer 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.

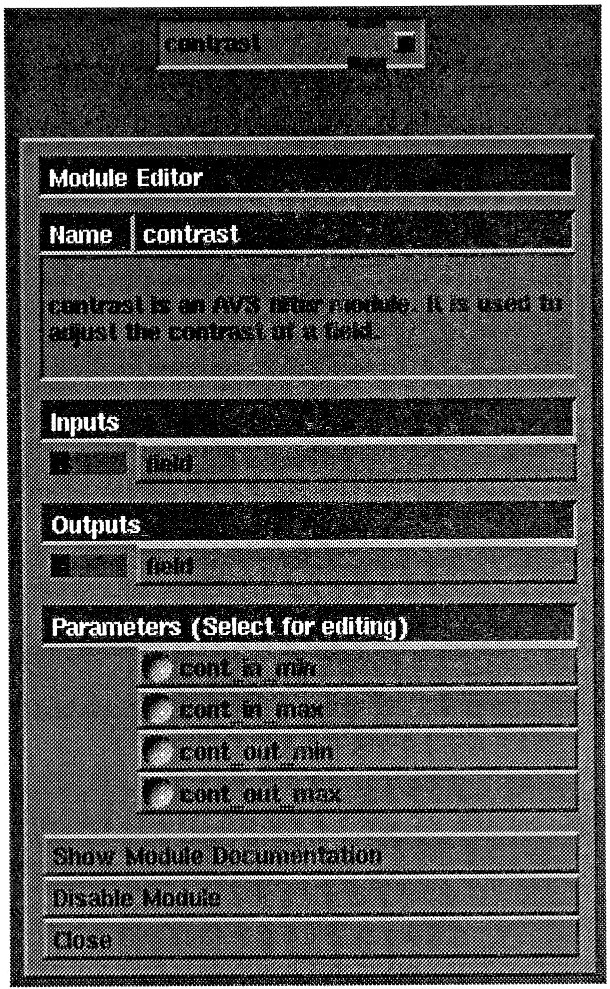
Each module icon shows the module's name, along with *input ports* and *output ports* to indicate the types of data that the module handles (Figure 6-2). The ports are color-coded to indicate the type of data that can pass through the port.

Figure 6-2. 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, 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. 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 (Figure 6-3).

Figure 6-3. The Module Icon and the Module Editor Window



Module Editor

Name contrast

contrast is an AVS filter module. It is used to adjust the contrast of a field.

Inputs

field

Outputs

field

Parameters (Select for editing)

cont_in_min

cont_in_max

cont_out_min

cont_out_max

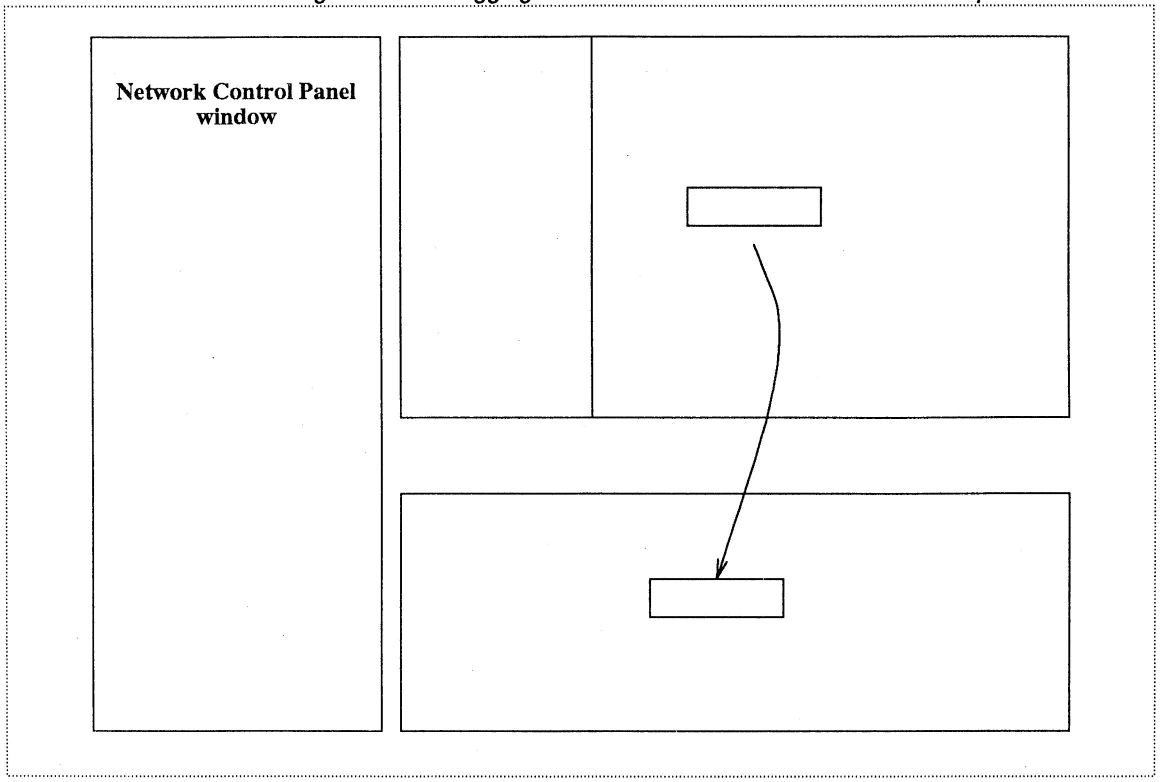
The next few sections describe how to work with module icons using the mouse. For quick reference, here's a listing of how the mouse buttons work in this context:

- ❑ **Left Mouse:** Move one or more icons.
- ❑ **Middle Mouse:** Establish a connection between two icons.
- ❑ **Right Mouse:** Break an existing connection between two icons.

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 (Figure 6-4).

Figure 6-4. Dragging a Module From the Palette Into the Workspace



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. You can 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 with the left mouse button.)

NOTE *The **render geometry** module is an exception. When you drag it into the Workspace, its control panel does **not** appear, nor is its name added to the Network Control Panel menu. The control panel for **render geometry** is the entire Geometry Viewer menu system described in Chapter 5.*

*When you click the small square on the **render geometry** module icon with the left mouse button, the Geometry 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 Geometry Viewer control panel aside, using the X Window System window manager. This allows you to see both the Geometry Viewer/**render geometry** 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, again using the left mouse button. You can also drag a rectangular "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 background part of the lasso area.)

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.

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.

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 port, 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*. (Most *renderer* 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.)

The ports are color-coded to represent the type of data that can pass through — an output port can only be connected to an input port with a matching color:

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 *GEOM Format* appendix). 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. See the *Geometry Conversion Programs* appendix for details.

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.

light blue = *pixmap*

A *pixmap* is an image stored in memory allocated by the X server. When a geometric description (a *geometry*) is rendered to produce pixel values, the pixmap format is used to hold the resulting image. Thus, the output of the **render geometry** module is often sent to the **display pixmap** module (or another module that handles pixmap input).

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. See the preceding chapter for a discussion of fields. (And for more information, see the *AVS Data Types* chapter in the *AVS Developer's Guide*.)

"Matching" has a special meaning in the case of the multi-colored field ports. For more information, see *Connecting Field Ports* below.

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.

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.

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 can help you to smooth over field-level incompatibilities. 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 **alpha blend**). Similarly, there are modules for handling dimension-based conversions. The **orthogonal slicer** takes a 2D slice from any 3D field. You can extract one dimension from a multi-dimension field using **extract scalar**; you can assemble a multi-dimension field from its component dimensions using **combine scalar**.

As an aid in matching field subtypes, the color bars for field input and output ports are divided into four parts (Figure 6-5 and Table 6-1):

Figure 6-5. Color-Coding for Field Input/Output Ports

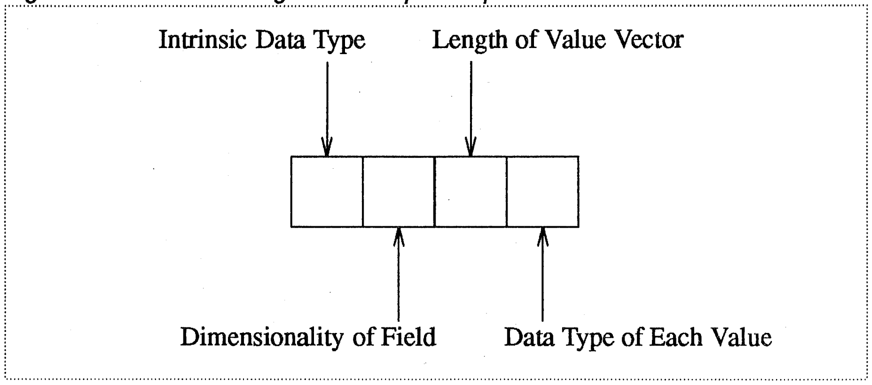
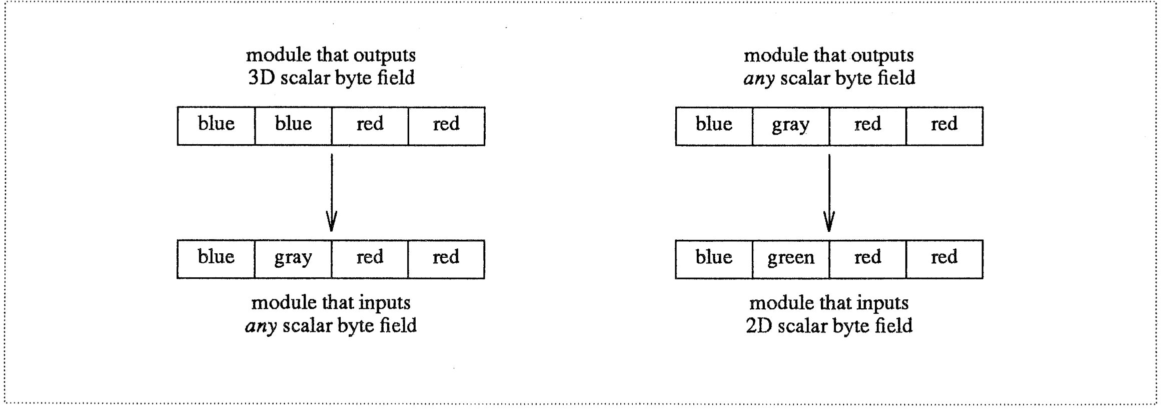


TABLE 6-1. Color-Coding for Field Input/Output Ports

	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 (i.e. value is a scalar quantity)
	green	3
	blue	4
	gray	either of the above
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 6-6).

Figure 6-6. Gray Color-Coding as Wildcard Value



WARNING *You must be careful when exploiting the flexibility illustrated in the above diagram. In fact, this diagram shows how you can connect three modules in such a way that a 3D scalar byte field is sent to a module that expects a 2D scalar byte field. Such a network will compute incorrect results.*

More generally, it is important to keep in mind that the amount of type-checking performed by AVS, both when you construct a network and when you run data through a network, is quite limited. It is possible to construct networks that will pass AVS's checks, but will perform incorrect or meaningless computations.

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.

The Module Editor and Parameter Editor Windows

Each module icon has a small square *Module Editor* button at its right edge. You can click this button to bring up the Module Editor window (see Figure 6-3).

NOTE *When an icon is in the Palette, you can use any mouse button to bring up the Module Editor. When the icon is in the Workspace, only the middle and right mouse buttons perform this function. The left mouse button acquires a new function — raising the module's control panel to the top of the Network Control Panel stack.*

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.

The Module Editor window also includes these function buttons:

Show Module Documentation
Displays the compete manual page for the module in a help browser window. This is the same page that you can view from the shell with the **man**(1) command.

Disable Module
Temporarily disconnects the module from its network, preventing it from receiving or sending data. This often has the effect of freezing the entire network. The module icon turns red to indicate its disabled state.

To reenable the module, click this button again.

The Parameter Editor

When you open 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, instead. This would allow you to enter an exact value, such as 48.2, rather than using the mouse to fine-tune a dial setting.

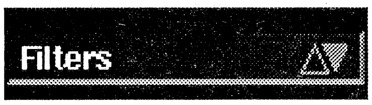
Finding the Module You Want

The Network Editor's standard module library includes more than 60 modules. This number is large enough so that you may not immediately spy 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 alloted 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 alloted space, it adds a scroll widget to the category's title bar:

Figure 6-7. 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. 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 explicit way to 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.

NOTE *Any time the cursor is in the title bar of a category, you can press BACKSPACE to scroll the category back to the top.*

Changing the Partitioning of the Network Construction Window

In some cases, you may find it desirable 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).

Module Libraries

When it begins execution, the Network Editor locates a set of executable modules, looking in directory */usr/avs/avs_library* and in any directories specified with **–modules** command-line options. It does not, however, automatically install all the modules it locates in the Module Palette. Instead, it installs only those modules specified in the *default module library*, which is defined in file */usr/avs/avs_library/SupportedModules*

You may find that this collection of modules 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.

To meet these needs, you can define you own module libraries. The **Read Module Library** function reads a module library file and replaces the current Palette contents with the new library. If you've already read in several libraries, you can switch back and forth among them instantly using the **Select Module Libary** function.

Module library files are ASCII text files, the format of which is described in the *File Formats* appendix.

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.

You are now ready to control the execution of the network using the Network Control Panel window.

Controlling the Execution of a Network

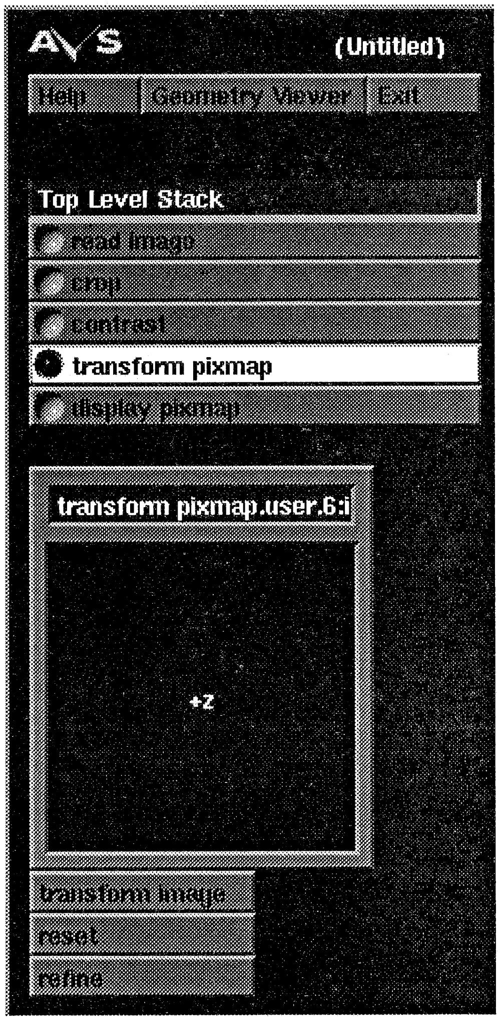
As you build a network, its modules start to execute. In most cases, nothing useful will occur until the network is complete and you specify the input data to be visualized. Thereafter, the network re-executes every time ...

- ❑ ... 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. (There is also a **Disable Flow Executive** funtion that 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 (Figure 6-8):

- 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.
- 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.)

Figure 6-8. Organization of the Network Control Panel



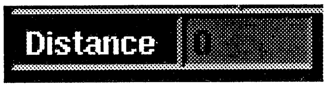
Using Control Widgets

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.

Using Type-In Controls

Figure 6-9 shows a typical type-in control widget.

Figure 6-9. 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:

Backspace Erases the last character currently in the type-in area.

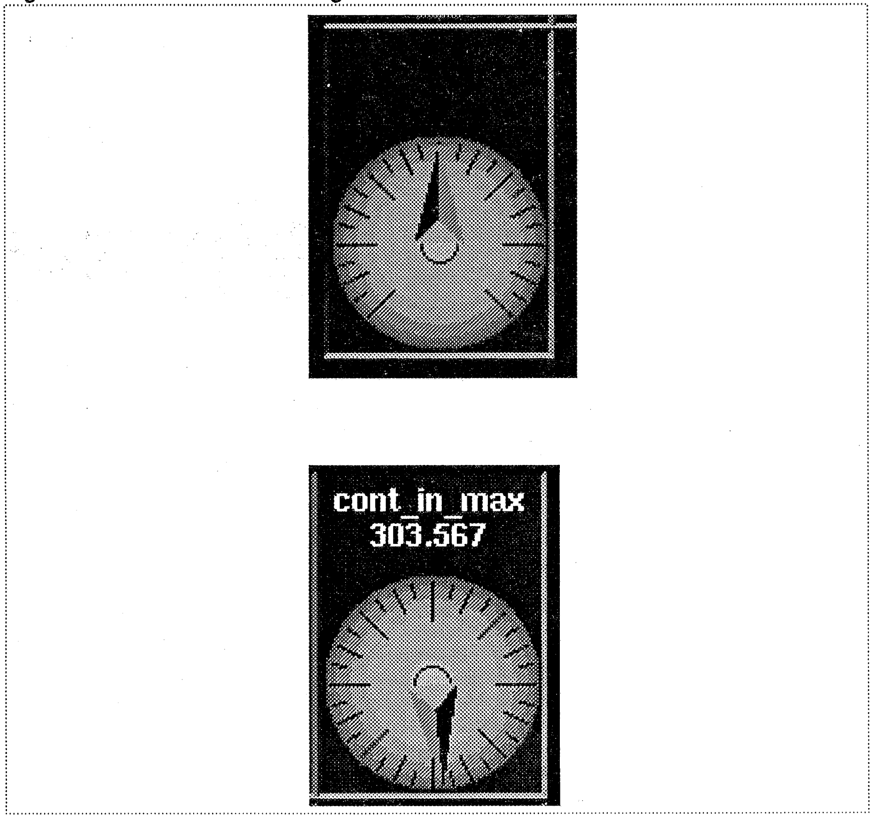
Ctrl-U Completely erases the type-in area.

When you press **Return** to finish entering the new value, AVS checks it 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.

Using Dial Controls

Figure 6-10 shows a typical dial control widget.

Figure 6-10. Dial Control Widget



You can use a dial either by *clicking* or by *dragging*:

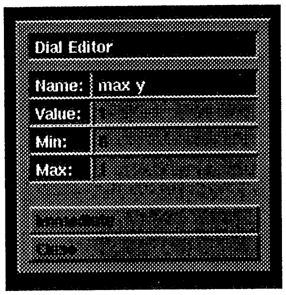
□ If you click with any mouse button at a location along the edge of the dial, the needle jumps immediately to that location and the current value indicator changes accordingly.

□ 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 — the needle stops moving when you reach the limit.

The Dial Editor. Dial control widgets are special in that they have an associated control window called the Dial Editor (Figure 6-11). 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.

Figure 6-11. Dial Editor



Dial Editor

Name: max y

Value:

Min:

Max:

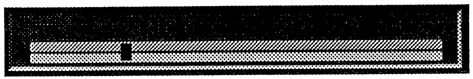
You can use this window to specify an exact value for the parameter (which may be easier than trying to move the dial needle by microscopic amounts). You can also change the dial's minimum and maximum bounds.

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 image that depends on the parameter to change continuously, also. (This may not be advisable for compute-intensive networks — changes to the image may lag behind the movement of the needle.)

Using Slider Controls

Figure 6-12 shows a typical slider control widget.

Figure 6-12. Slider Control Widget



As with a dial, you can use a slider either by *clicking* or by *dragging*:

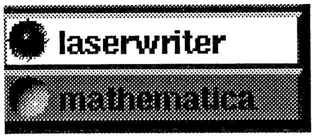
- ❑ If you click with any mouse button at a location along the slider, the crosshair jumps immediately to that location and the current value indicator changes accordingly.
- ❑ Alternatively, click and hold down any mouse button near the crosshair; then drag it to the left or right. As you do so, the 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 6-13 shows a typical *choice* ("radio buttons") control widget, which allows you to select from a mutually-exclusive set of choices.

Figure 6-13. A Set of Radio Buttons



● laserwriter

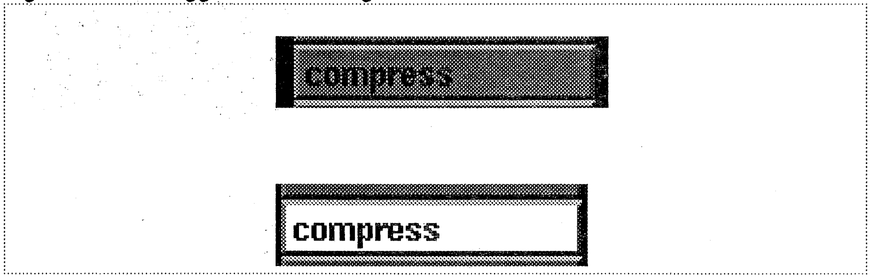
○ mathematica

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 6-14 shows a toggle control widget, in both the *off* state and the *on* state:

Figure 6-14. Toggle Control Widget

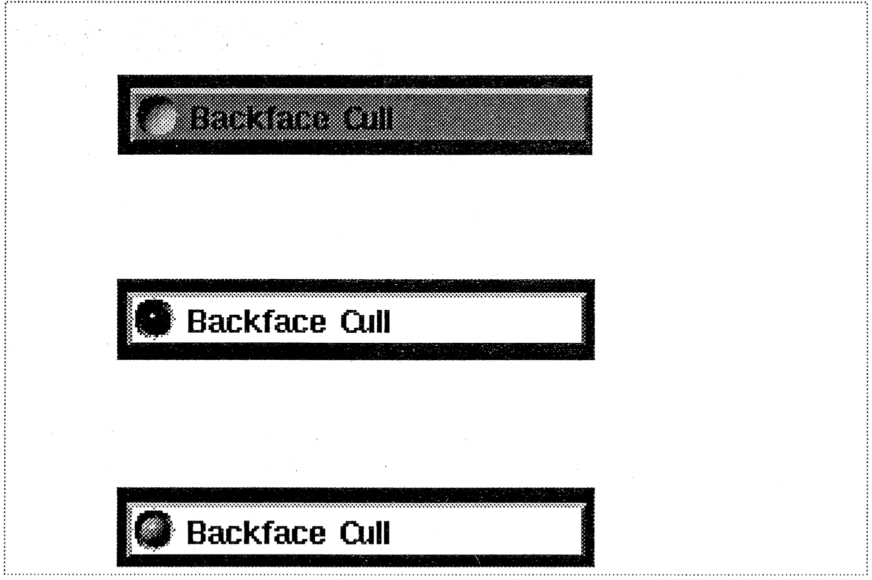


To change the state of a toggle switch, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button.

Using Tristate Controls

Figure 6-15 shows a typical tristate control widget, in its three states. This type of control is used for parameters that can assume three values, not just two.

Figure 6-15. Tristate Control Widget

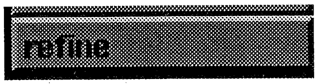


To change the state, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button. Successive clicks cycle the widget through its three states.

Using Oneshot Controls

Figure 6-16 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 6-16. Oneshot Control Widget

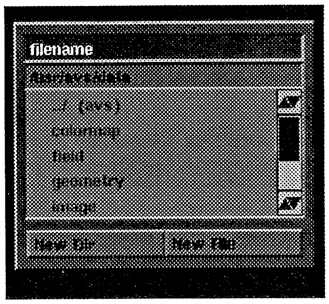


To use a oneshot control, move the cursor anywhere within its box. (The label inside the box "lights up" to indicate the cursor's presence.) Then click any mouse button to make the box flash.

Using File Browser Controls

Figure 6-17 shows a typical file browser control widget.

Figure 6-17. 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 (or leftward).
- ❑ 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 (or rightward).

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.

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.

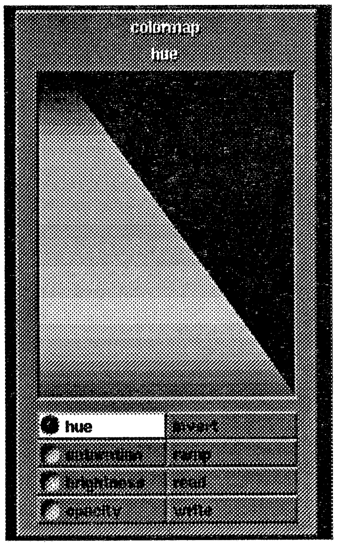
New File Pops up a dialog box that works the same way as the **New Dir** box. This allows you to specify the file to be processed, either with a full pathname or a name relative to the current directory. Note that the current directory does *not* change, no matter what name you enter.

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.

Using the Colormap Control

Figure 6-18 shows the control widget that generates a colormap. You can also use it to maintain a set of colormaps in disk files.

Figure 6-18. Colormap Control Widget



An AVS colormap is used by a number of modules to translate integers in the range 0..255 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 generator control 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 generator widget also includes the following buttons:

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.

Ramp
Restores the default colormap, which is a linear ramp starting in the blue range and ending in the red range.

Read
Write
These functions implement a system for maintaining 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 images and pixmaps:

Images
An *image* is originally defined in terms of pixels. The only scaling AVS performs on images 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. AVS chooses a scaling for the image that makes it too big for the window, rather than too small. 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.

❑ AVS refuses to scale an image as large as the new window size if such a scaling would severely strain system resources (e.g. main memory). In such cases, it chooses a smaller image size, so that part of the window remains unused.

Pixmaps
In most cases, a *pixmap* is originally defined as a geometry. AVS can scale pixmaps continuously. When you resize a pixmap window, the picture always resizes accordingly.

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. (The standard window manager on Stardent systems is **uwm**(1).)

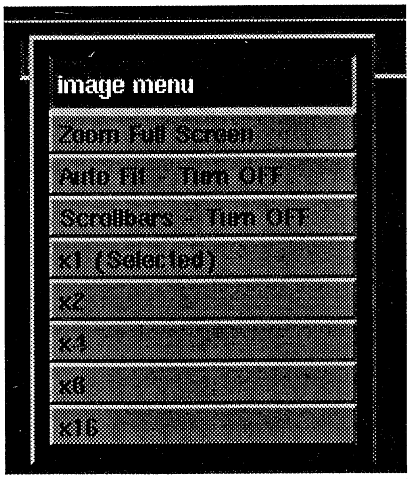
If you use the **Edit layout** function of the Network Editor to reorganize a network's control widgets, you may want to include the network's display windows in the reorganization. This topic is discussed in section *Including Display Windows in a Reorganized Layout* below.

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 (Figure 6-19). 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.

Figure 6-19. Display Window Pulldown Menu



The menu choices are as follows:

Zoom Full Screen
Enlarges the display window to be (approximately) the largest possible square size. The image is scaled up accordingly.

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).

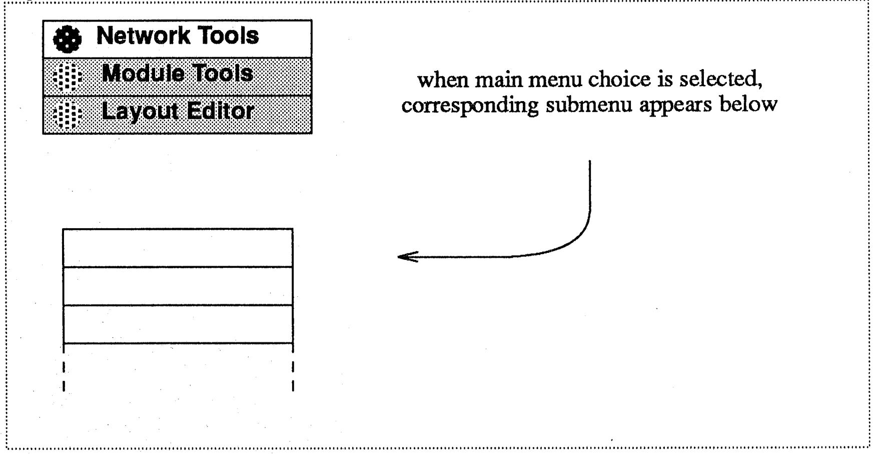
Resize Window to Fit Image
Use this choice when **Auto-Fit** is turned off and an image is too big for its window. (If you don't use this function, you can scroll the image in order to see different parts of it.) This choice is also useful when AVS has refused to scale up an image (as described above), 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 (Figure 6-20). At any moment, one of the menu choices is selected, and the corresponding submenu appears below the main menu.

Figure 6-20. Network Editor Main Menu


Network Tools
Module Tools
Layout Editor
when main menu choice is selected,
corresponding submenu appears below

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.

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/AVSnetwork.ps_**proc*** (where ***proc*** 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.

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_snapshot.parms_**proc***. (***proc*** is the process 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

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 Module Library

Empties the Module Palette, then adds all the modules in a specified *module library*. 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.

A library file names some combination of Stellar-supplied modules, user-written modules, and directories that contain modules. For example:

```
builtin         render geometry
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 *File Formats* appendix.

Write Module Library
Writes a module library file, consisting of the modules currently in the Palette.

Select Module Library
Invokes a pop-up menu, 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. This function provides a convenient way to switch back and forth among different sets of modules.

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.

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 **xterm** window from which you typed the **avs** command.

Layout Editor

Selecting **Layout Editor** places the Network Editor in a mode that allows you to "redesign" the user interface of the current network. 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:

- ❑ 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 DIGIT Dialbox input devices.
- ❑ Moving widgets around within their control panels.
- ❑ 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.

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.

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 DIGIT Dialbox device.

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_undoN.lyt_**proc*** to implement the undo feature. (*N* is a small integer and ***proc*** 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.**

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.

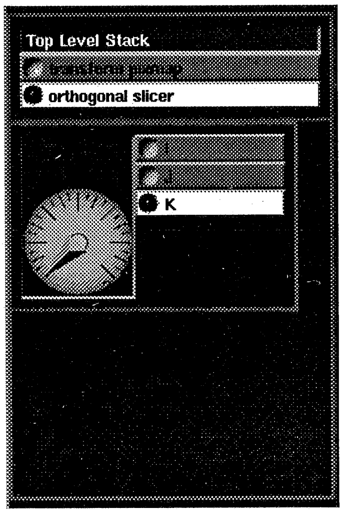
Note that in the Layout Editor, the mouse buttons modify the *layout* of control widgets, pages, and stacks rather than changing the *values* of parameters. Red borders around the control widgets, pages, and stacks remind you that you are in this Layout Editor mode.

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 (Figure 6-21); 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.

Figure 6-21. Window Borders in the Layout Editor



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 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 title. 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.

Including Display Windows in a Reorganized Layout. You can include the *display windows* created by the **display image** and **display pixmap** 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.

Using the Spaceball and Dialbox Managers. 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 Figures 6-22 and 6-23.

Figure 6-22. Spaceball Manager

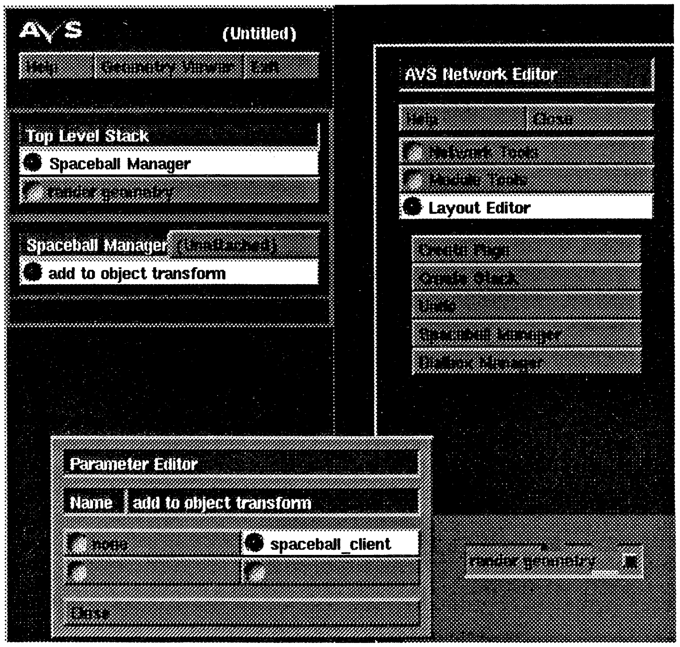
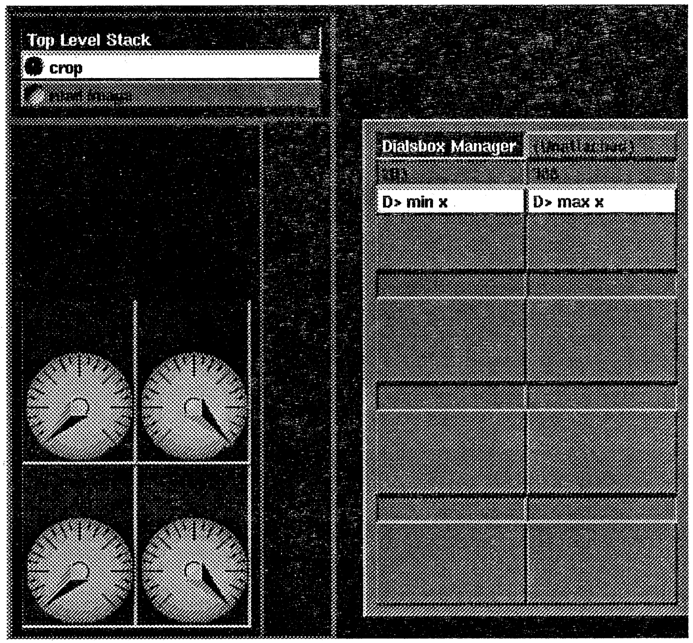


Figure 6-23. Dialbox Manager



AVS needs to know which serial ports the Spaceball and Dialbox are attached to. You can use the UNIX 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 Chapter 2).

When you click **Spaceball 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.

Whenever the name of a serial port *is* displayed, you can click on 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

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.

Stardent

7

AVS Module Manual Pages

Table of Contents

Chapter 7
AVS Module Manual Pages

avs

Application Visualization System

SYNOPSIS

avs *option(s)*

DESCRIPTION

The Application Visualization System (AVS) is an interactive tool for scientific visualization. It includes the following subsystems:

- ❑ **Image Viewer.** A high-level tool for manipulating and viewing images.
- ❑ **Geometry Viewer.** Allows you to compose "scenes" that contain 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.).

  The Geometry Viewer is an expanded version of the software that was delivered as Release 1 of AVS.
- ❑ **Volume Viewer.** A high-level tool for visualizing volume data. It can handle a wide variety of data sets that convey information for a sampling of points in a 3D space.
- ❑ **Network Editor.** A tool for connecting computational modules together into networks that perform visualization functions.

OPTIONS

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 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 *AVS STARTUP FILE* section for details.

–data *directory*

(startup file equivalent: **DataDirectory**) Specifies the directory in which the AVS Network Editor subsystem initially will look for data files (files used an input to computational modules).

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

–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).

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

–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*

–display *display-name*

Specifies the X Window System display on which AVS is to execute. This overrides the current setting of the DISPLAY environment variable.

–geometry *[geom-option(s)]*

Automatically invokes the Geometry Viewer subsystem at startup. You can include the following options that are specific to this subsystem:

–scene *scene-file*

Automatically loads a "scene" from disk storage. This option executes the Geometry Viewer's **Read Scene** function, using the file *scene-file.scene.*

–dir *pathname* Specifies *pathname* as the default directory used by the functions **Read Object, Save Object, Read Scene, Save Scene**, and the **Read** and **Save** functions in the Edit Property window.

The default data directory is */usr/avs/data* (same as the Network Editor).

–filter *pathname* Specifies *pathname* as the directory to search for geometry conversion utilities, named *..._to_geom.* See the *Geometry Conversion Programs* appendix.

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

–defaults *filename*

Specifies a Geometry Viewer defaults file. The format of this file is described in the *Geometry Viewer Script Language* Appendix.

–geometry *geom_spec*

Specifies an X Window System geometry (e.g. **500x500-5-5**) for the initial window created by the Geometry Viewer. **–usage**

Displays a usage message for the Geometry Viewer options. No AVS session is started if you type **avs –geometry –usage.**

The Geometry Viewer options correspond to the command-line options recognized by Release 1 of AVS.

–image Automatically invokes the Image Viewer subsystem at startup.

–volume Automatically invokes the Volume Viewer subsystem at startup.

–network *network-file*

Automatically invokes the Network Editor subsystem at startup, and loads the specified network file using the **Read Network** function.

–viewer *viewer-file*

Automatically creates a "viewer" that provides "turnkey" access to a group of existing networks. The AVS Image Viewer and Volume Viewer subsystems are implemented in this way.

–modules *directory*

(startup file equivalent: **ModulesDirectory**) Specifies the directory in which the AVS Network Editor subsystem initially will look for modules. All executable files in the directory are examined to determine whether they contain one or more modules.

You can use more than one **–modules** option to have AVS search through multiple directories for modules.

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

–usage Displays a usage message for the AVS options (except for the Geometry Viewer option — see above). (Does not start an AVS session.)

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

AVS STARTUP FILE

When it begins execution, AVS searches for a *startup file*, which specifies the locations of various directories. AVS looks for the following files, in the order listed:

./.avsrc	(current directory)
$HOME/.avsrc	(home directory)
/usr/avs/runtime/avsrc	(system directory)

At most *one* of these startup files is read. If AVS finds one of them, it ignores the others. A */usr/avs/runtime/avsrc* file is included on the AVS distribution tape. Each line of the AVS startup file consists of keyword-value pair, with whitespace separating the keyword and the value. For example:

```
NetworkWindow        867x567+407+2
NetworkDirectory     /usr/johnp/avs/nets
ModulesDirectory     /usr/johnp/avs/modules
DataDirectory        /usr/johnp/avs/data
DialDevice           /dev/tty02
```

In most cases, 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.

The AVS startup file keywords are as follows:

DataDirectory

(command-line equivalent: **–data**) Specifies the directory in which the various "read data" modules (**read field**, **read geom**, etc.) initially will look for data files. **DialDevice** (command-line equivalent: none) Indicates the serial communications port to which a DIGIT dialbox device is attached.

This entry corresponds to the environment variable DIALS; if DIALS is set, the startup file entry (if any) is ignored.

ImageScrollbars

(command-line equivalent: none) If set to the value **NO**, 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.)

ModulesDirectory

(command-line equivalent: **–modules**) Specifies one or more directories in which the AVS Network Editor subsystem initially will look for executable modules. You can specify several directories, separating the pathnames with commas (and optionally, whitespace).

NetworkDirectory

(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

(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.

Path (command-line equivalent: **–path**) Specifies the directory tree in which AVS itself is installed. **SaveMessageLog** (command-line equivalent: none) It 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). **SpaceballDevice** (command-line equivalent: none) Indicates the serial communications port to which a Spaceball device is attached.

This entry corresponds to the environment variable SPACEBALL; if SPACEBALL is set, the startup file entry (if any) is ignored.

AVS Modules

introduction to manual pages for AVS modules

DESCRIPTION

This section includes a manual page for each Stellar-supplied computational module in the AVS system. These manual pages are also available on-line. You can view them either within AVS itself, or from a shell.

Within AVS

Click on the small square in any module icon to open its Module Editor window. Then click the **Show Module Documentation** button to view the complete manual page for the module.

From a shell

Issue a command in the form:

man 6 *module_name*

Note that in *module_name*, you should replace any SPACE characters that appear on the module icon with underscore characters. For instance, to display the manual page for the **read volume** module, issue this command:

```
man 6 read_volume
```

MODULE LISTING

The modules included in this release of AVS are:

avs	Application Visualization System — command-line and startup file
AVS Modules	this manual page
alpha blend	generate 2D image from 3D colored data
arbitrary slicer	map 3D scalar field to 3D mesh
bubbleviz	generate spheres to represent values of 3D field
clamp	restrict values in data field
colorizer	convert field of data values to color values
colormap manager	share colormaps among subnetworks
combine scalars	combine scalar fields into a vector field
compute gradient	compute gradient vectors for 2D or 3D data set
contrast	perform linear transformation on range of field values
crop	extract range of elements from a field
display image	show image in a display window
display pixmap	show pixmap in a display window
dot surface	generate points that define an isosurface
dot to geometry	map a list of points
downsize	reduce size of data set by sampling
extract scalar	extract scalar field(s) from a vector field
field to byte	transform any field to an byte-valued field
field to double	transform any field to a field of double-precision floating point values
field to float	transform any field to a field of single-precision floating point values

AVS Modules

field to int	transform any field to an integer-valued field
field to mesh	transform a 2D scalar field to a surface in 3D space
flip normal	change direction of each vertex normal
generate colormap	output AVS colormap
geom to scatter	convert geometry to point list
gradient shade	apply lighting and shading to colored data set
hedgehog	show vectors on a slice of a 3D 3-vector field
histogram stretch	balance the histogram of a data set
image manager	share images among subnetworks
image to pixmap	convert image to pixmap
interpolate	change the size of the data
isosurface	generate an isosurface for a volume of data
mirror	reverse array indices in a 2D or 3D data set
offset	make an object smaller
orthogonal slicer	slice through volume data with plane perpendicular to coordinate axis
output postscript	convert pixmap to PostScript™ and store in file
particle advector	release grid of particles into velocity field
pdb to geom	create molecule geometry from PDB file
pixmap to image	transform AVS pixmap to AVS image
read field	read AVS field from a disk file
read geom	reads a data file containing an AVS ´geometry´
read image	read image file from disk into a field
read volume	read volume file from disk into a field
render geometry	convert geometric description to image (Geometry Viewer)
render manager	share geometries among subnetworks
scatter dots	generate spheres at points in 3D space
shrink	make an object smaller
stream lines	generate stream lines for a vector field
stream mesh	generate stream lines for a vector field as a polygonal mesh
threshold	restrict values in data field
transform pixmap	perform 3D transformation on pixmap
transpose	exchange dimensions in a 2D or 3D data set
tube	convert lines to cylindrical tubes
vbuffer	perform volumetric rendering on volume data
vector curl	compute the curl of a vector field
vector div	compute the divergence of a vector field
vector grad	compute the vector gradient of a scalar field
vector mag	compute the magnitude of a vector field

vector norm	normalize a vector field
volume bounds	generate bounding box of 3D 3-vector field
volume manager	share volumes among subnetworks
wireframe	convert object from surface to wireframe representation
write field	write a field description to disk
write image	store image data in a file
write volume	write volume data to a file

alpha blend

generate 2D image from 3D colored data

SUMMARY

Name	alpha blend
Type	renderer
Inputs	field 3D 4-vector byte
Outputs	pixmap

Parameters

Name	*Type*	*Default*	*Min*	*Max*
X-Rot	float	0.0	*none*	*none*
Y-Rot	float	0.0	*none*	*none*

DESCRIPTION

The **alpha blend** module generates an image (2D grid of pixels) from a 3D block of voxels. (*Voxels* are the 3D analogue of *pixels*.) The *alpha blending* technique treats the voxel block as a set of 2-dimensional images, stacked on top of one another. For each line of sight, you can see though layers that contain semi-transparent voxels, up to the nearest layer with an opaque voxel.

The voxel color values are blended from back to front, using each voxel's opacity value:

auxiliary	red	green	blue

this field interpreted as voxel's opacity value

these three fields make up voxel's color value

This produces cloud-like images, with the densities of the clouds controlled by the **Opacity** ramp of the colormap that assigned the color values.

INPUTS

Data Field (required; field 3D 4-vector byte)
The input data must be a 3D block of voxels. That is, the data at each point of the 3D field must be a 4-vector of bytes in the alpha-red-green-blue format used in images.

PARAMETERS

By default, the "front" from which the block is viewed is the direction of the positive Z-axis. You can change the direction by rotating the block about the X-axis and/or Y-axis, using these parameters:

X-Rot A floating point value that simulates rotating the data set around the X-axis (horizontal).

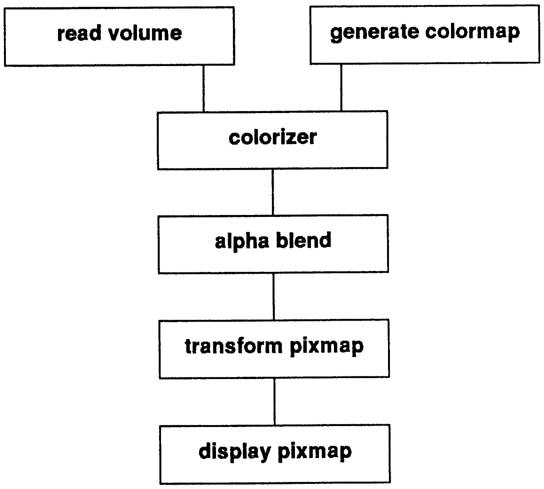
Y-Rot A floating point value that simulates rotating the data set around the Y-axis (vertical).

OUTPUTS

Pixmap The output data is in the form of an AVS pixmap.

EXAMPLE 1

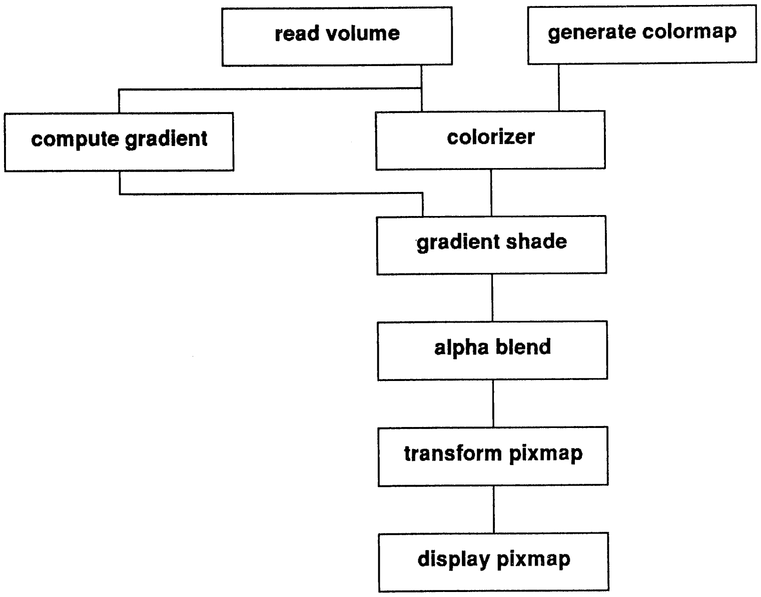
The following network shows how 3D data can be colored using the **colorizer** module, then blended into a 2D image using the **alpha blend** module:



Note that this network uses the **transform pixmap** module to allow the user to resize the image with the window manager. Otherwise, the generated image will be a fixed size, determined by the size of the original data set. For instance, a 64x64x64 data set would produce a fixed-size 128x128 pixel image. (The extra pixels accommodate rotation of the data, which produces a larger image.)

EXAMPLE 2

Another interesting technique is to apply a light source to the data. In order, to do this, the gradient of the data (which approximates the "surface normal") must be computed. A network for doing this "gradient shading" is:



LIMITATIONS

Because of the shearing technique used to simulate axis rotations, there are certain X- and Y-axis angles for which the image breaks up and eventually disappears completely. Complete rotations around one axis only (zero rotation around the other axis) always work correctly.

RELATED MODULES

Modules that could provide the **Data Field** input:

colorizer
gradient shade

Modules that could be used in place of **alpha blend**:

vbuffer
orthogonal slicer

Modules that can process **alpha blend** output:

transform pixmap
display pixmap

arbitrary slicer

map 3D scalar field to 3D mesh

SUMMARY

Name	arbitrary slicer				
Type	mapper				
Inputs	field 3D scalar byte (optional) colormap				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	X rotation	dial	0	0	360
	Y rotation	dial	0	0	360
	distance	dial	0	–2	2
	mesh res	integer	36	8	144
	trilinear	toggle	off		

DESCRIPTION

The **arbitrary slicer** module extracts a 2D slice from a 3D volume of data. The slice place can be oriented arbitrarily — it need not be parallel to any of the coordinate axes.

The volume of data is represented as a 3D scalar field (which defines a uniform lattice within the volume). The slice plane is represented as a 2D grid, with a parameter-controlled resolution. The intersection of the volume and the grid is a *mesh* of vertices in 3D space.

Each vertex in the mesh is assigned a color that corresponds to one or more values of the 3D scalar field. Since, in general, the mesh vertices do *not* coincide with the original lattice points, an interpolation method can be used — see the *trilinear* input parameter below.

By default, the volume is placed at the origin and the slice plane is the X-Y plane. The orientation of the slice plane is controlled by the **slice xform** parameter.

You can control the resolution of the mesh using the **mesh res** parameter. At lower resolutions, fewer original data points are used in the computations; at higher resolutions, more points are used.

The optimal way to use this module is to start off with a low resolution mesh, position it as desired, then increase the resolution and turn on trilinear mapping.

INPUTS

Data Field (required; field 3D scalar byte uniform)

The input data must be a 3D field, with a byte value at each location in the field. The field must be uniform.

Colormap (optional; colormap)

By default, the value computed for each vertex of the mesh is used as the hue in HSV space. If you specify a colormap, the values are transformed to the range 0..255, and are then used as indexes into the colormap.

PARAMETERS

trilinear (toggle) Controls the way in which each vertex of the output mesh is assigned a color:

- ❑ **If OFF**, a nearest-neighbor algorithm is used. Each mesh vertex is assigned the byte value of the nearest point in the lattice.
- ❑ **If ON**, a trilinear interpolation is performed. The value at each vertex depends on the byte values at the eight lattice points that are the corners of the "enclosing cube".

The trilinear interpolation method is more accurate but takes longer to compute, particularly with larger meshes.

X rotation Rotates the mesh along the first axis of the volume.

Y rotation Rotates the mesh along the second axis of the volume.

distance Moves the mesh along its normal.

mesh res Controls the resolution of the slice plane mesh. Higher resolution meshes result in higher quality representations, but take longer to compute and render. The default mesh is 8x8.

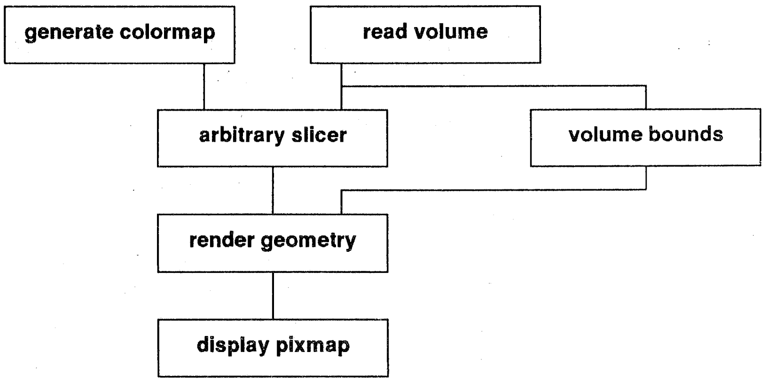
OUTPUTS

Geometry (geometry)

The output is an AVS *geometry*.

EXAMPLE

This example shows a common usage of the **arbitrary slicer** module:



The **volume bounds** modules gives a reference frame for orienting the slice plane. Often, an **isosurface tiler** is also input to the **render geometry** module.

LIMITATIONS

The **arbitrary slicer** module does not handle either rectilinear or irregular field data.

bubbleviz

generate spheres to represent values of 3D field

SUMMARY

Name	bubbleviz				
Type	mapper				
Inputs	field 3D scalar byte colormap				
Outputs	field 1D 3-coord 4-vector real				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Radius	float	1.0	0.0	100.0

DESCRIPTION

The **bubbleviz** module generates spheres of various radii and colors at the element locations of a 2D or 3D field. This is a "cuberille" style of volume visualization, except that it uses spheres rather than cubes.

This module can be used for non-uniform input fields (rectilinear or irregular).

INPUTS

Data Field (required; field 2D/3D scalar byte)

The principal input data for the **bubbleviz** module is a 2D or 3D field. The data at each point of the field must be a single byte. The byte values will be interpreted as numbers in the range 0..255.

Color Map (colormap)

The optional colormap may be of any size. Since each input datum is a byte, the natural size for the colormap is 256. If you specify a larger colormap, its entries beyond the 256th are unused.

PARAMETERS

Radius A multiplier factor for the sphere radii. This is particularly useful for irregular fields, for which the computational-to-physical mapping often makes the default spheres too small.

OUTPUTS

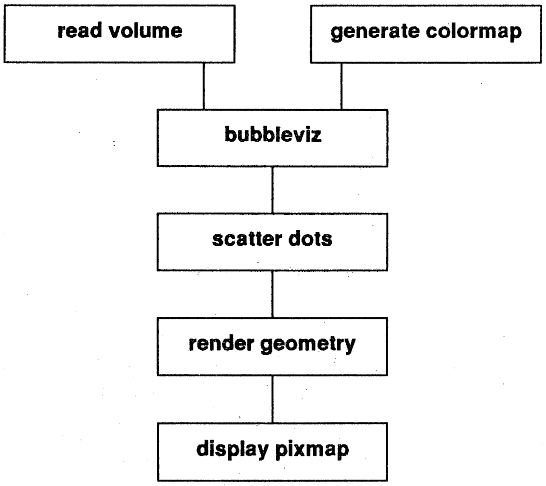
Data Field (field 1D 3-coord 4-vector real)

The output is a list of points in 3D space, with a 4-vector of reals at each point:

- □ The first three elements of the lookup value specify the red-green-blue components of the sphere's color (0.0 = no color; 1.0 = maximum color).
- □ The fourth element is interpreted as the sphere's radius. If the radius value is 0.0, no sphere is generated as output.

EXAMPLE

A typical network using this module looks like this:



Note that the list of points generated by the **bubbleviz** module is converted to a geometry by the **scatter dots** module.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume

Modules that could be used in place of **bubbleviz**:

colorizer
gradient shade

Modules that can process **bubbleviz** output:

scatter dots

LIMITATIONS

The **bubbleviz** module can generate extremely large databases (one sphere per voxel for volume data). Use 0.0 values in the last field of the input colormap ("opacity" field) to eliminate unnecessary data.

clamp

restrict values in data field

SUMMARY

Name	clamp				
Type	filter				
Inputs	field *any-dimension any-data*				
Outputs	field of same type as input				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	clamp_min	float	0.0	none	none
	clamp_max	float	255.0	none	none

DESCRIPTION

The **clamp** module transforms the values of a field as follows:

- ❑ Any value less than the value of the **clamp_min** parameter is set to **clamp_min**.
- ❑ Any value greater than the value of the **clamp_max** parameter is set to **clamp_max**.
- ❑ All values within the **clamp_min**-to-**clamp_max** range are not changed.

After being **clamp**'ed, a data set's values are all in this range:

clamp_min ≤ *value* ≤ **clamp_max**

Note the difference between the **clamp** and **threshold** modules:

- ❑ **threshold** sets values outside the specified range to be zero.
- ❑ **clamp** sets values outside the specified range to be the range's minimum and maximum values.

INPUTS

Data Field (required; field *any-dimension any-data*)
The input data may be any AVS field.

PARAMETERS

clamp_min A floating-point number that specifies the minimum output value.

clamp_max A floating-point number that specifies the maximum output value.

OUTPUTS

Data Field (field *any-dimension any-data*)
The output field has the same dimensionality and type as the input field.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume
any other filter module

Modules that could be used in place of **clamp**:

threshold

Modules that can process **clamp** output:

colorizer
any other filter module

colorizer

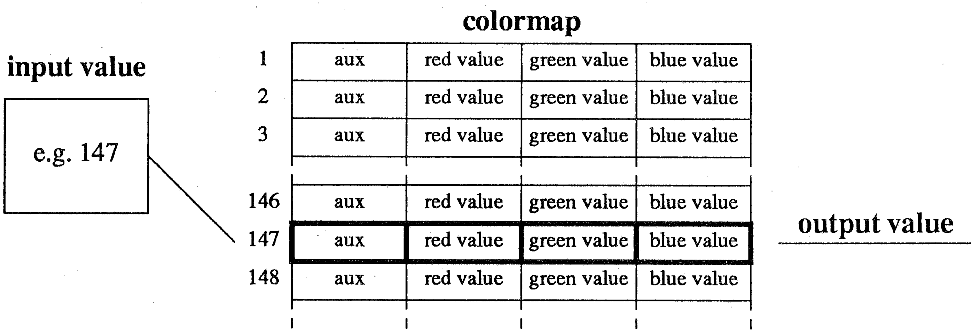
convert field of data values to color values

SUMMARY

Name	colorizer
Type	filter
Inputs	field *any-dimension* scalar byte colormap
Outputs	field *any-dimension* 4-vector byte
Parameters	*none*

DESCRIPTION

The **colorizer** module converts the data at each point of a scalar field from a *byte* to a *color* (4-vector of bytes). The conversion is accomplished by using the byte value (0–255) as an index into a *colormap*:



1	aux	red value	green value	blue value
2	aux	red value	green value	blue value
3	aux	red value	green value	blue value
146	aux	red value	green value	blue value
147	aux	red value	green value	blue value
148	aux	red value	green value	blue value

INPUTS

Data Field (required; field *any-dimension* scalar byte)

The principal input data for the **colorizer** module is a field, which can be of any dimensionality. The data at each point of the field must be a single byte. The byte values will be interpreted as numbers in the range 0..255.

Color Map (optional; colormap)

The optional colormap may be of any size. Since each input datum is a byte, the natural size for the colormap is 256. If you specify a larger colormap, its entries beyond the 256th are unused. **Default:** If this input is omitted, a gray-scale colormap is used (0 = black; 255 = white).

OUTPUTS

Field of Colors (field *any-dimension* 4-vector byte)

Each input byte is transformed into a color value, which is structured as four bytes, as illustrated above. The red, green, and blue bytes specify a true-color pixel value. The *auxiliary* byte is typically used to specify an opacity value (0 = completely transparent; 255 = completely opaque).

The dimensionality of the output field is the same as that of the input field. The output field is four times as large as the input field, since each byte (8 bits) is converted to a color value (32 bits).

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume
field to byte

Modules that could provide the **Color Map** input:

read colormap
generate colormap

Modules that could be used in place of **colorizer**:

arbitrary slicer

Modules that can process **colorizer** output:

alpha blend
gradient shade
display image

colormap manager

share colormaps among subnetworks

SUMMARY

Name	colormap manager	
Type	data	
Inputs	none	
Outputs	colormap	
Parameters	*Name*	*Type*
	Colormap Manager	colormap
	Colormap Choices	choice

DESCRIPTION

The **colormap manager** module produces an AVS *colormap* data structure, for use by modules that transform input data into color values. These modules include:

colorizer
arbitrary slicer
bubbleviz
field to mesh
isosurface

colormap manager works exactly like **generate colormap**, with one exception: separate active subnetworks, each with its own **colormap manager** module, share a single "pool" of colormaps.

A menu of all the active colormaps appears in a choice menu below each *colormap manager*'s editing widget. All the menus have the same entries — different maps can be selected in different managers.

PARAMETERS

Colormap Manager

A colormap generator widget. See the **generate colormap** manual page for details on using this widget.

Colormap Choices

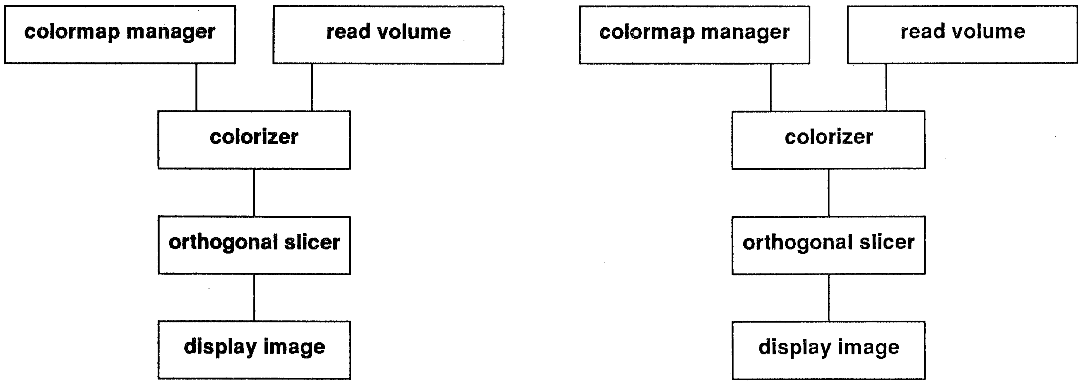
A set of choices, listing each of the currently active colormaps.

OUTPUTS

colormap The output is an AVS colormap.

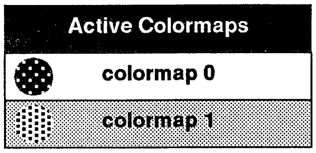
EXAMPLE

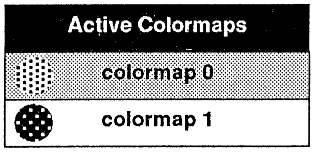
Suppose the following two subnetworks are active, created to slice through two different databases:



colormap manager

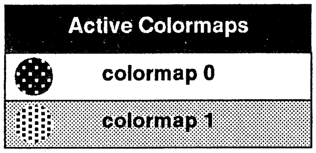
Each **colormap manager** module has its own *colormap editor* control widget. Below the two colormap editors are two choice menus:

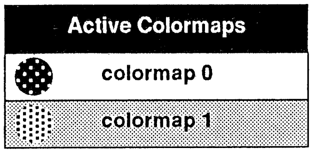




The same "pool" of colormaps is shown in each menu, but a different colormap is currently selected for each subnetwork.

By default, each new **colormap manager** that is instantiated from the module Palette has it's own unique colormap editor. You can click on the **colormap 0** button for the second subnetwork in order to have both subnetworks use the same colormap:





Now, editing the colormap in *either* **colormap manager** module is reflected in both subnetworks.

You can extend the sharing of colormaps to any number of currently active subnetworks. Each must have its own **colormap manager** module.

NOTE **colormap manager** modules are used in both the *Image Viewer* and *Volume Viewer* subsystems.

combine scalars

combine scalar fields into a vector field

SUMMARY

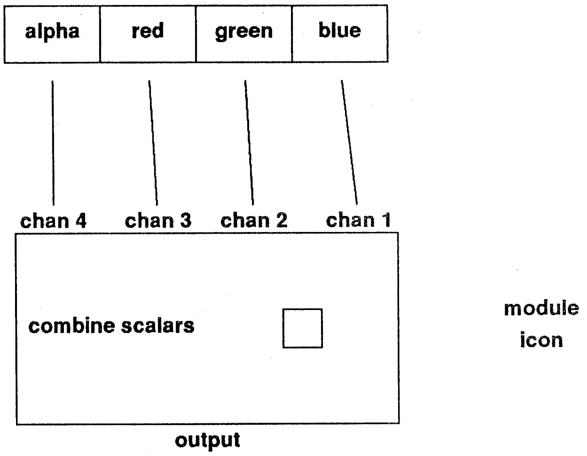
Name	combine scalars
Type	filter
Inputs	field *any-dimension* scalar *any-data* (channel 0 — optional) field *any-dimension* scalar *any-data* (channel 1 — optional) field *any-dimension* scalar *any-data* (channel 2 — optional) field *any-dimension* scalar *any-data* (channel 3 — optional)
Outputs	field *same-dimension* 4D *same-data*
Parameters	none

DESCRIPTION

The **combine scalars** module combines up to four fields with scalar data values into a field whose data values are 4D vectors. The input field must be of like dimension and the scalar values must be of the same type.

This module is generally most useful for constructing images or gradient fields from separately computed components.

The order of the inputs ports on this module's Network Editor icon corresponds to the diagram of a 4-byte color value (an image is essentially a sequence of these color values):



INPUTS

None of the following inputs is required, but at least one of them must be present. If an input channel is omitted, zero values are output.

Channel 4 (optional; field *any-dimension* scalar *any-data*) A set of values to be output in the *fourth* dimension of the output vectors.

Channel 3 (optional; field *any-dimension* scalar *any-data*) A set of values to be output in the *third* dimension of the output vectors.

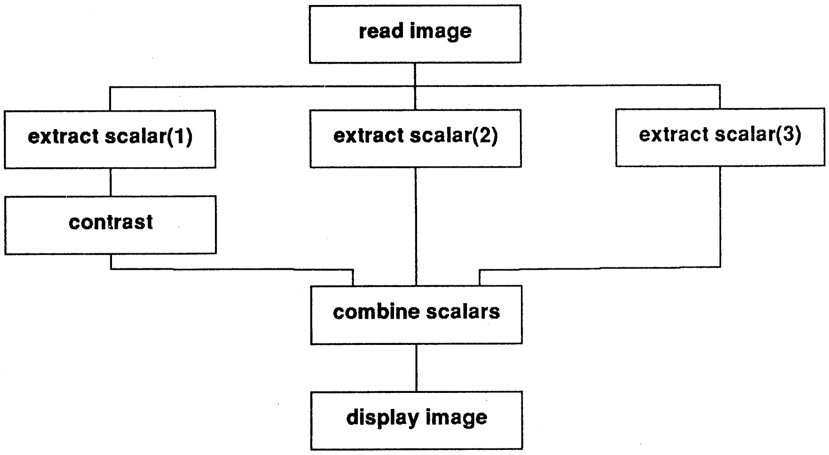
Channel 2 (optional; field *any-dimension* scalar *any-data*) A set of values to be output in the *second* dimension of the output vectors.

Channel 1 (optional; field *any-dimension* scalar *any-data*) A set of values to be output in the *first* dimension of the output vectors.

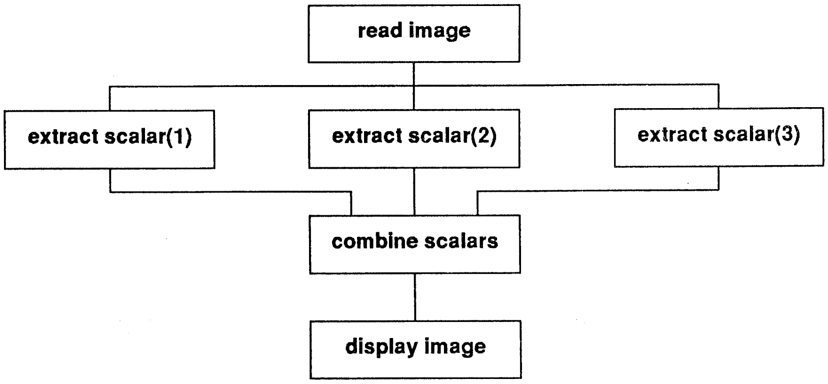
OUTPUTS

Field (field *same-dimension* 4D *same-data*) The four scalar input streams are assembled into a single output stream consisting of 4D vectors.

EXAMPLE 1 The following network performs contrast stretching on only the *red* band of an image.



EXAMPLE 2 The following network swaps the *green* and *blue* bands of an image:



RELATED MODULES

extract scalar

contrast

perform linear transformation on range of field values

SUMMARY

Name	contrast
Type	filter
Inputs	field *any-dimension* scalar *any-data*
Outputs	field of same type as input

Parameters

Name	*Type*	*Default*	*Min*	*Max*
cont_in_min	float	0.0	*none*	*none*
cont_in_max	float	255.0	*none*	*none*
cont_out_min	float	0.0	*none*	*none*
cont_out_max	float	255.0	*none*	*none*

DESCRIPTION

The **contrast** module transforms all the values in a scalar field. Two different types of transformation take place:

- ❑ **Linear transform:** All values that fall within the "input range" specified by the **cont_in_min** and **cont_in_max** parameters are transformed linearly to the "output range" **cont_out_max .. cont_in_max**.

$$new\_value = \frac{(cont\_out\_max - cont\_out\_min) * (value - cont\_in\_min)}{cont\_in\_max - cont\_in\_min}$$

(More precisely, this is an *affine transformation*.) In essence, this transformation "stretches" or "compresses" one specified range of data to fit another specified range.

- ❑ All values that fall outside the specified input range are "clamped" to the limit values of the output range.

The **contrast** module typically is used to remove low-level noise from images and volumes, or to increase the contrast in faded images and volumes.

INPUTS

Data Field (required; field *any-dimension* scalar float)

The input data may be an AVS field of any dimensionality.

PARAMETERS

cont_in_min

Specifies the bottom of the range of input values that will be transformed linearly.

cont_in_max

Specifies the top of the range of input values that will be transformed linearly.

cont_out_min

Specifies the bottom of the range of output values. All values ≤ **cont_in_min** will be transformed to this value.

cont_out_max

Specifies the top of the range of output values. All values ≥ **cont_in_max** will be transformed to this value.

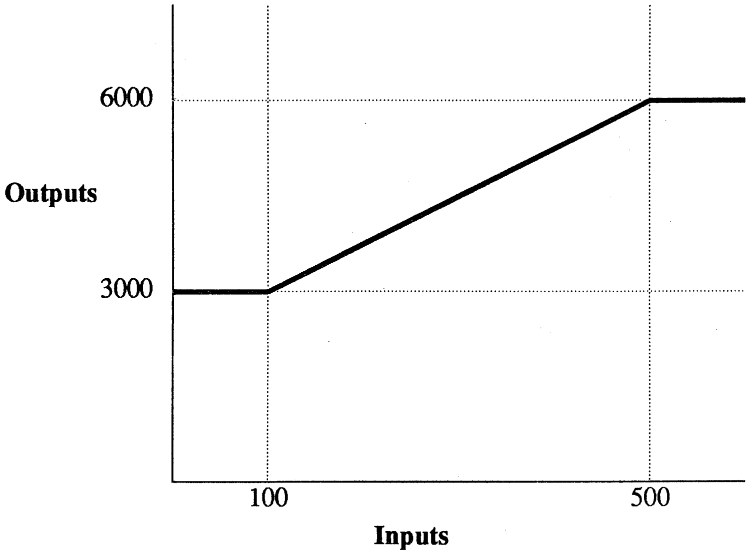
OUTPUTS

Data Field The output field has the same dimensionality as the input field, but the number of elements in each dimension is reduced.

EXAMPLE

The following diagram shows how field values are transformed given these parameters:

cont_in_min = 100
cont_in_max = 500
cont_out_min = 3000
cont_out_max = 6000



You can use **contrast** to make a negative out of an image by "flipping" the output values (e.g. **cont_out_min** = 255; **cont_out_max** = 0).

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume

Modules that could be used in place of **contrast**:

threshold

Modules that can process **contrast** output:

any other filter module

crop

extract range of elements from a field

SUMMARY

Name	crop
Type	filter
Inputs	field *any-dimension any-data*
Outputs	field of same type as input

Parameters

Name	*Type*	*Default*	*Min*	*Max*
min x	int	1st indx	1st indx	last indx
max x	int	last indx	1st indx	last indx
min y	int	1st indx	1st indx	last indx
max y	int	last indx	1st indx	last indx
min z	int	1st indx	1st indx	last indx
max z	int	last indx	1st indx	last indx

DESCRIPTION

The **crop** module changes the size of a field by extracting the data within a specified range of elements. This process is analogous to "cropping" a photographic image.

This module is useful for subsampling the data without changing it (e.g. by interpolation). It preserves the resolution of the data, but may change its aspect ratio. Typical uses are to eliminate uninteresting portions of the data and to increase processing speed by reducing the amount of data.

INPUTS

Data Field (required; field *any-dimension any-data*)
The input data may be any AVS field.

PARAMETERS

All the input parameters are normalized to the range 0.0–1.0. Thus, specifying a parameter value of 0.75 means "three-quarters of the way" along a particular dimension of the input field. Note that the parameters indicate *positions* of elements in the field — they have nothing to do with the *values* of field elements.

min x Specifies the lower bound array index in the field's first dimension.

max x Specifies the upper bound array index in the field's first dimension.

min y Specifies the lower bound array index in the field's second dimension.

max y Specifies the upper bound array index in the field's second dimension.

min z Specifies the lower bound array index in the field's third dimension. (Has no effect for 2D input data sets.)

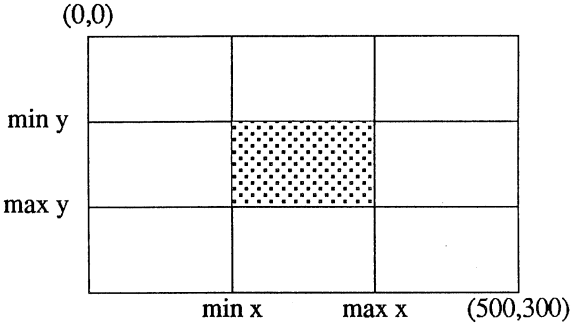
max z Specifies the upper bound array index in the field's third dimension. (Has no effect for 2D input data sets.)

OUTPUTS

Data Field The output field has the same dimensionality as the input field, but the number of elements in each dimension is reduced.

EXAMPLE

Suppose you want to process the middle third of a field that contains an 500x300 pixel image:



Set the x-axis and y-axis parameters as follows:

- **min x** = 167
- **max x** = 333
- **min y** = 100
- **max y** = 200

Note that all these settings are on the normalized scale (0.0–1.0). The actual 500x300 size of the data set is irrelevant.

RELATED MODULES

Modules that could provide the **Data Field** input:

- read volume
- filter modules

Modules that could be used in place of **crop**:

- downsize
- interpolate

Modules that can process **crop** output:

- colorizer
- gradient shade
- arbitrary slicer
- orthogonal slicer
- any other filter module

LIMITATIONS

crop works for 2D and 3D data sets only.

display image

show image in a display window

SUMMARY

Name	display image				
Type	renderer				
Inputs	field 2D 4-vector byte				
Outputs	*none*				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Magnification	choice	x1	x1	x16
	Automag_Size *(internal)*	integer	256	50	1024
	Max Image Dimension *(internal)*	integer	1280	100	4096

DESCRIPTION

The **display image** modules takes an input image and displays it in a display window. This window has a pulldown menu, accessed via the small square in the window's title bar. The menu allows you to control image magnification, window resizing, and other options relating to the display window.

When the image is larger than the display window, you can scroll it with the mouse, either by "dragging" the image itself or by using horizontal and vertical scrollbars.

You can resize the display window manually, using the X Window System window manager. You can also have the window resize itself automatically, in response to a change in the image contents or a magnification selected from the display window's pulldown menu.

INPUTS

Data Field (required; field 2D 4-vector byte)

The input field must be in the AVS *image* format.

PARAMETERS

Magnification

A choice to specify a power of 2 (1,2,4,8,16) by which to multiply each dimension of the image.

Automag_Size

(for internal use only) This is used as a communications port to handle resizing of the image. Do not change this parameter.

Maximum Image Dimension

(for internal use only) Controls resource allocation — see description below.

MAGNIFICATION

You can magnify an image for closer examination, although the magnified image will provide no new detail. Magnification is implemented by duplicating the pixels in the original image. The result is "blockier" but provides a closer look at the image. There are several magnification levels (x1,x2,x4,x8,x16) in the pulldown menu, with the current magnification marked as (selected).

Magnification may result in views that require more pixmap resources than the X server can provide. An "invisible" *Maximum Image Dimension* parameter provides a limit to how large an overall view can be safely produced. If the image size times the magnification exceeds this parameter in one of its dimensions, the module will present a warning and automatically reduce the magnification to produce a size within the limits. While intended as an internal limit, you could attach a widget to this parameter to increase this limit when required; exceeding the available resources of the X server will cause a crash of AVS as an X client process.

RESIZING

The display window can be resized in several ways. You can use the X window manager's *resize* window operation to enlarge or shrink the display window. An approximate image magnification is automatically chosen that makes the image at least as large as the window.

The pulldown menu also provides several ways to resize the window to certain fixed sizes:

- □ **Zoom Full Screen.** Resizes the window to fill the square working area of the screen (approximately 1024 x 1024), and magnifies the image to fit. If the window is embedded in a page or stack (see *Layout Editor* in the Network Editor chapter), it becomes a top-level window that can be freely resized and moved using the X window manager.
- □ **Resize to Fit Image.** Resizes the window to fit the image exactly at the current magnification. (The maximum size window is the full screen window described above.) As with **Zoom Full Screen**, an embedded display window becomes a top-level window.
- □ **Unzoom.** Resizes and moves the window to return to its location before a **Zoom Full Screen** or a **Resize to Fit Image.** If the window originally was embedded in a page or stack, it will be re-embedded there.
- □ **AutoFit - Turn On/Off.** This toggle switch controls the automatic fitting of the display window size to its image. When this feature is enabled (the default), **display image** automatically resizes the display window whenever the image size changes. This can occur when you select a new magnification or when an entirely new image is input to **display image.** The new display window size exactly fits the new image size (unless the window is currently embedded in a page or stack).

SCROLLING

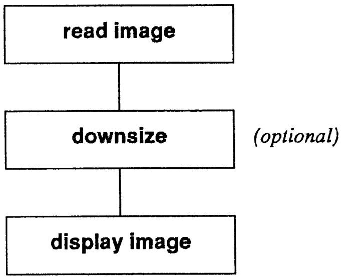
Whenever the image is larger than the display window, only a portion of the image is visible. You can "pan" over the entire image in two ways:

- □ Using the horizontal and vertical scrollbars that automatically appear. These scrollbars work the same way as those on File Browser windows.
- □ By dragging the image itself. Place the mouse cursor anywhere in the image, click and hold down any mouse button, and drag the mouse. The image moves continuously, and the scrollbars are updated when you release the mouse button. The image automatically stops scrolling when it hits its borders.

The **Scrollbars – Turn On/Off** selection on the pulldown menu allows you to disable or reenable the appearance of scrollbars along the right and bottom edges of the display window. (The "drag-the-image" method is always enabled.) You may want to suppress the scrollbars to reduce distraction or to provide additional viewing space.

The **ImageScrollbars** parameter in the AVS startup file (see Chapter 2) determines whether image windows get scrollbars by default when they contain oversize images. If you do not use this startup parameter, scrollbars are initially enabled.

EXAMPLE



RELATED MODULES

display pixmap

LIMITATIONS

This module can fail for large images that require more resources than the X Server process can provide. While the *Maximum Image Dimension* protects against this in many cases, it may fail if the original image itself is too large. For large images, the **downsize** module may be useful in producing a more manageable original image size.

display pixmap

show pixmap in a display window

SUMMARY

Name	display pixmap			
Type	renderer			
Inputs	pixmap			
Outputs	*none*			
Parameters	*Name*	*Type*	*Default*	*Choices*
	Store Frames	toggle		
	Append Frame	one shot		
	Delete Current	one shot		
	Replay	choice	Off	Continuous Bounce Off
	Current Frame	integer slider		
	Max Frames	integer typein		
	Replace Speed	integer slider		
	Save Image	string typein		

DESCRIPTION

The **display pixmap** module displays its input pixmap in a display window. It automatically sizes the pixmap to fit the window.

In addition, you can:

❑ Save the pixmap as an AVS image in a file.

❑ Create and play back a "flipbook" of consecutive images.

These capabilities are invoked using the module's input parameters, as described in the sections below.

INPUTS

pixmap The input data must be an AVS *pixmap*, typically created by the **render geometry** module.

PARAMETERS

The following parameters control a pixmap-animation capability. Note that this is independent of the animation facility in the Geometry Viewer (**render geometry** module), and works somewhat differently. See the *ANIMATION* section below for more information.

Store Frames

This toggle controls whether all new frames are automatically added to the animation sequence.

AppendFrames

Explicitly adds the currently displayed pixmap to the animation sequence. (Use when **Store Frames** is off.)

Delete Current

Deletes the currently displayed pixmap from the animation sequence.

Replay This choice widget controls how the animation sequence is to be played back: The choices are **Continuous, Bounce**, and **Off.**

Current Frame

The number of the current frame in the animation sequence (first frame = 0). This field is a typein — change the number to jump directory to another frame.

Max Frames

A typein field that specifies the ceiling for the number of frames that you can place in an animation sequence.

Replay Speed

Controls the rate at which an animation is played back. The larger the value, the greater the delay between frames.

Save Image This is a typein field. If you type a filename or pathname into this field, the current pixmap is written to a file when you press **Return.**

SAVING AN IMAGE

To save an image in a file, type the filename as the value of the **Save Image** parameter. When you press **Return**, the file is created. To save another image under the same name, you can move the mouse cursor to the **Save Image** input area and press **Return** again.

ANIMATION

By changing the input data or by adjusting the parameters of upstream modules (e.g. **transform pixmap**), you can have the **display pixmap** window show a sequence of images. You can create an animation ("flip book") by designating certain images to be "frames". Then, you can play back the images, adjusting the speed with a control widget.

Because each of the images in a flip book takes up a significant amount of system memory, there is a *Max Frames* parameter. Be sure that its value is low enough so that your system can comfortably keep all of the images in memory at the same time. AVS requires roughly 4 bytes of memory per pixel of each your image. The larger the display window, the greater the memory requirements.

There are two ways to create a flip book:

- ❑ To save *all* the images that appear in the window (actually, just the last *Max Frames* that are produced — see below), turn on the **Store Frames** toggle. As each image is drawn, it will be appended to the end of the flip book. If *Max Frames* images have already been saved, this new pixmap will replace the oldest pixmap in the cycle.
- ❑ If you want to selectively add images to the flip book, modify the image until it is as you want it, then select the one-shot **Append Frame.** This appends the image to the end of the existing flip book. This method allows you to carefully construct a flipbook animation.

The **Replay** parameter controls the way in which the flip book is displayed. It has three selections:

- ❑ **Continuous** plays through all of the frames in the animation, wrapping around when it reaches the end.
- ❑ **Bounce** plays forward through the last *Max Frames* or fewer frames. When it reaches the end, it plays backwards through those frames.
- ❑ **Off** turns off the animation facility

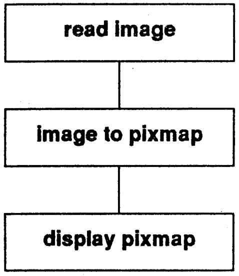
The **Replay Speed** parameter controls the rate at which flip book frames are displayed.

The **Current Frame** parameter allows you to select a particular frame "manually". It is normally updated to display the current frame, but for cases in which such updating would impact animation performance, it is not updated. Note that since only the last *Max Frames* frames are stored, the animation can begin at a frame other than 0.

After you select a particular frame, you can delete it with the one-shot **Delete Frame.**

display pixmap

EXAMPLE



RELATED MODULES

transform pixmap, alpha blend, render geometry

LIMITATIONS

There is no way to store the "first *max frames*" frames of an animation loop.

dot surface

generate points that define an isosurface

SUMMARY

Name	dot surface				
Type	filter				
Inputs	field 3D scalar *any-data*				
Outputs	field 1D scalar (irregular 3-space)				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Stepsize	real	.01	1.0E–5	1.0
	Threshold	real	.02	–100	100

DESCRIPTION

The **dot surface** module accepts a 3D scalar field as input and generates a list of points that defines an isosurface. The input field is composed of cells, where each cell is defined as a subvolume composed of six faces. Each cell is processed checking for a possible intersection of the surface. If the cell does contribute to the surface it is then subdivided until the maximum physical dimension of the resulting subcell is ≤ the value of the **Stepsize** parameter. A smooth surface can be generated in this manner, given a sufficiently small **Stepsize** value.

The running time of this module is directly proportional to the number of cells processed and the number of cells that contribute to the surface. It is inversely proportional to the **Stepsize** value.

If the input field is uniform, then a physical grid is generated mapping the data volume into a canonical size. The largest dimension of the volume is mapped into the interval: [-1.0, +1.0]. Other dimensions are scaled accordingly, thus if a uniform volume consisting of 100 nodes in the x direction, 50 in the y direction and 20 in the z direction will have a bounding volume of: x=[-1.0, +1.0], y=[-0.5,+0.5], z=[-0.2,+2.0]. The distance between each node is then approximately equal to 0.02. The Stepsize parameter is relative to this length scale.

INPUTS

Data Field (required; field 3D)

This module uses a scalar data value for each field element. If the input is a vector-valued field, then the first component of the vector is used as the scalar value.

PARAMETERS

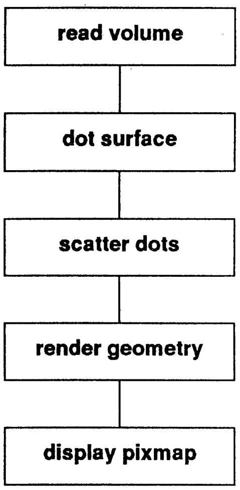
Stepsize A floating-point value that determines the resolution of the isosurface. The smaller this value, the smoother the surface.

Threshold A floating-point value that specifies the common data value on the isosurface: for each point on the isosurface, the field element's data value equals the **Threshold** value.

OUTPUTS

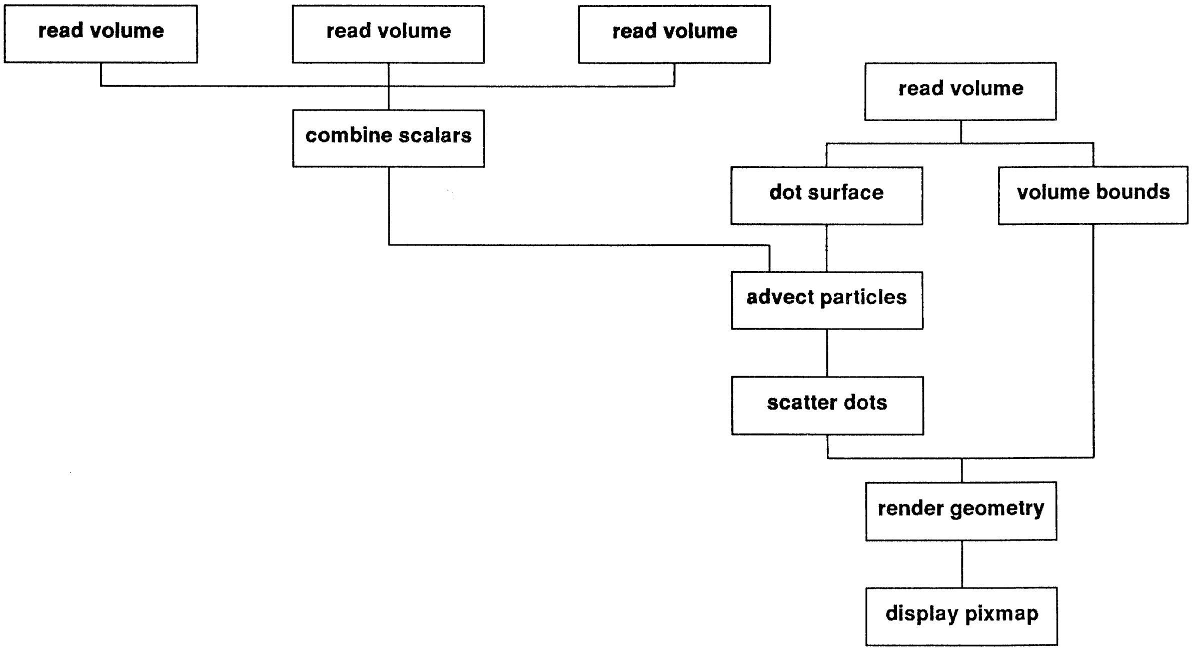
Point List (field 1D scalar irregular 3-space). The scalar data value for each output field element is unused. The only useful information is the 3D coordinate data.

EXAMPLE 1



EXAMPLE 2

In the following network, **dot surface** is used to generate a point list for use with **particle advector**. The three **read volume** modules create a vector velocity field, which is used as the driving force for the new locations of the point list originally generated by **dot surface**.



LIMITATIONS

The number of points may be inadequate to represent areas of small surface curvature with respect to the cell's local coordinate system.

A maximum of 80,000 points will be generated. Once the module calculates this number of points, it returns leaving all other cells unprocessed.

RELATED MODULES

Modules that could provide the Data Field input: read volume combine scalars

Modules that could be used in place of dot surface:

isosurface
vbuffer

alpha blend

Modules that can process dot surface output: scatter dots

downsize

reduce size of data set by sampling

SUMMARY

Name	downsize				
Type	filter				
Inputs	field *any-dimension any-data*				
Outputs	field of same type as input				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	downsize	integer	1	1	16

DESCRIPTION

The **downsize** module changes the size of the input data set by subsampling the data. It extracts every *N*th element of the field along each dimension, where *N* is the value of the **downsize factor** parameter. This technique preserves the aspect ratio of the input data.

This module is useful for operating on a reduced amount of data, in order to adjust other processing parameters interactively. After the parameter values have been set, you can remove the **downsize** module, so that the full data set is used for final processing.

Alternatively, retain the **downsize** module in the network, so that you can interactively choose between image quality (**downsize factor** = 1 for highest-resolution data) and execution speed (**downsize factor** > 1 for lower-resolution data).

INPUTS

Data Field (required; field *any-dimension any-data*)
The input data may be any AVS field.

PARAMETERS

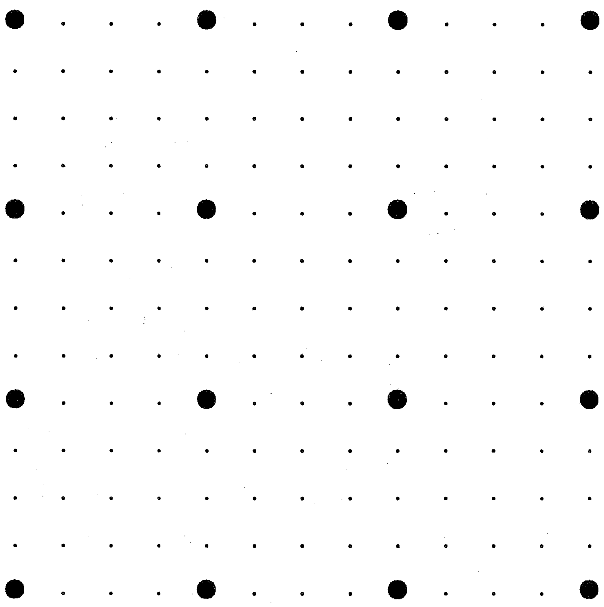
downsize Determines how data elements from the field are sampled. Increasing this parameter causes more elements to be skipped over, thus *decreasing* the size of the output.

OUTPUTS

Data Field The output field has the same dimensionality as the input field, but the number of elements in each dimension is reduced by the **downsize factor**.

EXAMPLE

The following diagram shows how a **downsize factor** of 4 reduces a 2D field. Each element of the field is represented by a dot. Only the larger dots are included in the output field.



LIMITATIONS

downsize works for 1-D, 2D, and 3D data sets only.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume
filter modules

Modules that could be used in place of **downsize**:

interpolate (arbitrary sampling)
crop (subset at high resolution)

Modules that can process **downsize** output:

colorizer
gradient shade
arbitrary slicer
orthogonal slicer
any other filter module

extract scalar

extract scalar field(s) from a vector field

SUMMARY

Name	extract scalar				
Type	filter				
Inputs	field *any-dimension n*-vector *any data* (*n* = 1..9)				
Outputs	field *same-dimension* scalar *same-data*				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Channel	int	0	0	9

DESCRIPTION

The **extract scalar** module inputs a field whose data values are vectors (1D to 10D), and outputs one of the dimensions ("channels") as a scalar-valued field. The output field has the same structure as the input field, except that its data values are scalars (vector length of 1).

This module is useful for performing operations on individual channels of vector fields. It is frequently used with the **combine scalars** module, which composes vector fields from individual scalar fields.

INPUTS

Data Field (required; field *any-dimension n*-vector *any data*) The input data may be any field whose data values are vectors with 10 or fewer dimensions. Even scalar fields may be used, since their data values are considered to be 1D vectors.

PARAMETERS

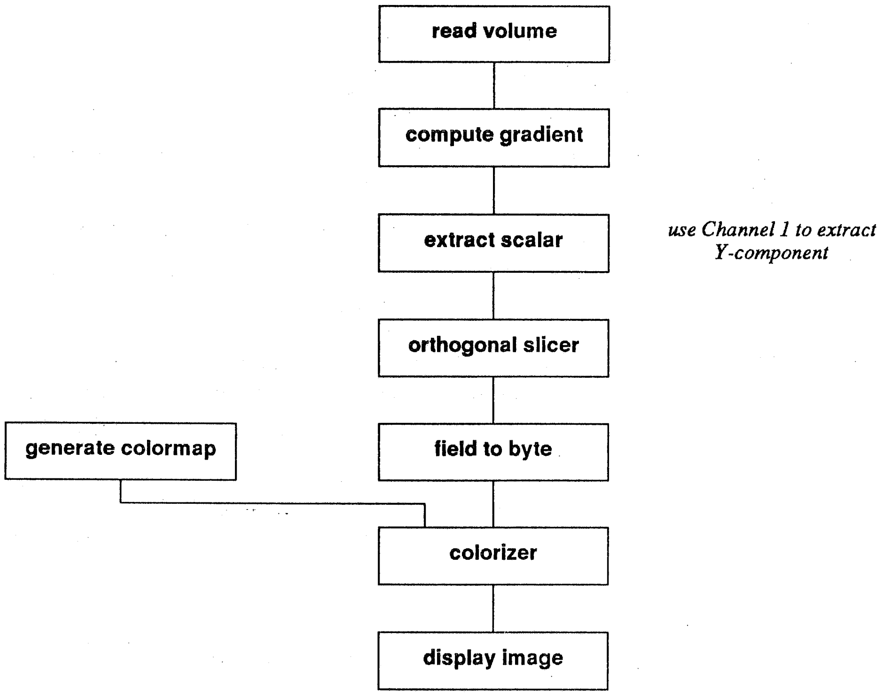
Channel Selects the dimension of the input data values to be output. The bounds of the control widget are automatically adjusted according to the vector length of the input field.

OUTPUTS

field (*same-dimension* scalar *any-data*) The output field has the same dimensionality as the input field. The data for each element is reduced from a vector to a scalar.

EXAMPLE 1

This examples displays a slice of the Y-component of the gradient field of a volume:



use Channel 1 to extract Y-component

For additional examples, see the **combine scalars** manual page.

RELATED MODULES

combine scalars

field to byte

transform any field to an byte-valued field

SUMMARY

Name	field_to_byte			
Type	filter			
Inputs	field *any-dimension any-data*			
Outputs	field *same-dimension* byte			
Parameters	*Name*	*Type*	*Default*	*Choices*
	byte normalize	toggle	off	on,off

DESCRIPTION

The **field_to_byte** module takes a field of data (*integer*, *real*, *double*, or *byte*) and converts it to an *byte* field. It can be used in conjunction with volume visualization modules that have a bias towards byte fields (**colorizer**, **compute gradient**, **arbitrary slicer**, etc.).

By default, the input data is normalized to the range 0..255 If the toggle parameter **byte_normalize** is turned off, the data is "clamped" to that range instead. (See below for details.)

INPUTS

Data Field (required; field *any-dimension any-data*)

The input data may be any AVS field.

PARAMETERS

byte_normalize

This is a toggle parameter:

❑ **If on:** The data is transformed linearly into the range 0..255:

$$new\_value = \frac{(value - min) * 255}{(max - min)}$$

❑ **If off:** The data is "clamped" so that no value falls outside the range 0..255:

If $value < 0$	$new\_value = 0$
If $0 \le value \le 255$	$new\_value = value$
If $value > 255$	$new\_value = 255$

OUTPUTS

Data Field (field *same-dimension* byte)

The output field has the same dimensionality as the input field, but each scalar value is forced to be a byte.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume

Modules that could be used in place of **field_to_byte**:

field to int
field to float
field to double

Modules that can process **field_to_byte** output:

read volume

field to double

transform any field to a field of double-precision floating point values

SUMMARY

Name	field_to_double			
Type	filter			
Inputs	field *any-dimension any-data*			
Outputs	field *same-dimension* double			
Parameters	*Name*	*Type*	*Default*	*Choices*
	double normalize	toggle	off	on,off

DESCRIPTION

The **field_to_double** module takes a field of data (*byte*, *real*, *double*, or *double*) and converts it to an *double* field. This may be useful for computing fields at greater data resolutions.

By default, the input data is simply cast (re-typed) to be double-precision floating point. If the toggle parameter **double_normalize** is turned on, the data is also normalized to the range 0..1. (See below for details.)

INPUTS

Data Field (required; field *any-dimension any-data*)

The input data may be any AVS field.

PARAMETERS

double_normalize

This is a toggle parameter:

- ❑ **If on:** The data is transformed linearly into the range 0..1:

$$new\_value = \frac{(value - min)}{(max - min)}$$

- ❑ **If off:** The data is converted to double-precision floating point format (IEEE 754).

OUTPUTS

Data Field (field *same-dimension* double)

The output field has the same dimensionality as the input field, but each scalar value is forced to be a double-precision number.

RELATED MODULES

read volume

field to float

transform any field to a field of single-precision floating point values

SUMMARY

Name	field_to_float			
Type	filter			
Inputs	field *any-dimension any-data*			
Outputs	field *same-dimension* float			
Parameters	*Name*	*Type*	*Default*	*Choices*
	float normalize	toggle	off	on,off

DESCRIPTION

The **field_to_float** module takes a field of data (*byte*, *real*, *double*, or *float*) and converts it to an *float* field. It can be used in conjunction with volume visualization modules that have a bias towards *float* fields (**isosurface, dot_surface**).

By default, the input data is simply cast (re-typed) to be single-precision floating point. If the toggle parameter **float_normalize** is turned on, the data is also normalized to the range 0..1. (See below for details.)

INPUTS

Data Field (required; field *any-dimension any-data*)

The input data may be any AVS field.

PARAMETERS

float_normalize

This is a toggle parameter:

- ❑ **If ON**, the data is transformed linearly into the range 0..1:

$$new\_value = \frac{(value - min)}{(max - min)}$$

- ❑ **If OFF**, the data is converted to single-precision floating point format (IEEE 754).

OUTPUTS

Data Field (field *same-dimension* float)

The output field has the same dimensionality as the input field, but each scalar value is forced to be a single-precision number.

RELATED MODULES

read volume, isosurface, dot surface

field to int

transform any field to an integer-valued field

SUMMARY

Name	field_to_int			
Type	filter			
Inputs	field *any-dimension any-data*			
Outputs	field *same-dimension* integer			
Parameters	*Name*	*Type*	*Default*	*Choices*
	int normalize	toggle	off	on,off

DESCRIPTION

The **field_to_int** module takes a field of data (*byte*, *real*, *double*, or *int*) and converts it to an *int* field. This may be useful for performing integer math with greater precision (–65536..65535) than that offered by byte fields (0..255).

By default, the input data is "clamped" to the range 0..65535. If the toggle parameter **int_normalize** is turned on, the data is normalized to that range instead. (See below for details.)

INPUTS

Data Field (required; field *any-dimension any-data*)

The input data may be any AVS field.

PARAMETERS

int_normalize

This is a toggle parameter:

❑ **If ON**, the data is transformed linearly into the range 0..65535:

$$new\_value = \frac{(value - min) * 65535}{(max - min)}$$

❑ **If OFF**, the data is "clamped" so that no value falls outside the range 0..65535:

If $value < 0$	$new\_value = 0$
If $0 \leq value \leq 65535$	$new\_value = value$
If $value > 65535$	$new\_value = 65535$

OUTPUTS

Data Field (field *same-dimension* integer)

The output field has the same dimensionality as the input field, but each scalar value is forced to be an integer.

RELATED MODULES

read volume

field to mesh

transform a 2D scalar field to a surface in 3D space

SUMMARY

Name	field_to_mesh				
Type	mapper				
Inputs	field 2D scalar colormap (optional)				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Z scale	float	1.0	0.0	5.0

DESCRIPTION

The **field to mesh** module converts a two-dimensional field into surface in 3D space, represented as a GEOM-format *mesh*. Each element of the field is mapped to a point in a base plane. The height of the mesh above each point in this plane is proportional to the scalar value of the field.

For irregular fields, the "base plane" need not actually be planar. The 2D grid of field elements is mapped into 3D space using the coordinate array included in the field description.

INPUTS

Data Field (required; field 2D scalar)

The input data must be a 2D field with a scalar data value at each element. The data value may be of any primitive type: byte, integer, float, or double.

Colormap (optional)

Colors each vertex of the mesh, according to the data value (0..255) at that point. (Non-byte data values are first mapped to the range 0..255, then translated to colors.) If no colormap is supplied, the vertices are colored white.

PARAMETERS

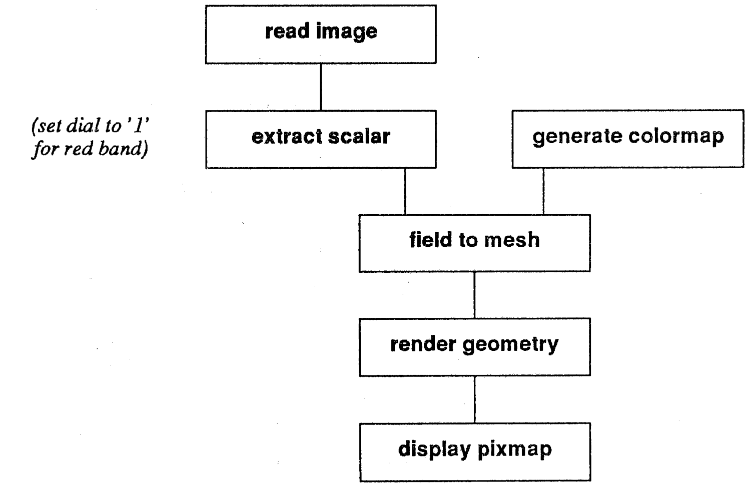
Z scale With uniform input fields, determines the height of the mesh. With rectilinear and irregular input fields, this parameter is unused.

OUTPUTS

Geometry The output is an AVS *geometry*.

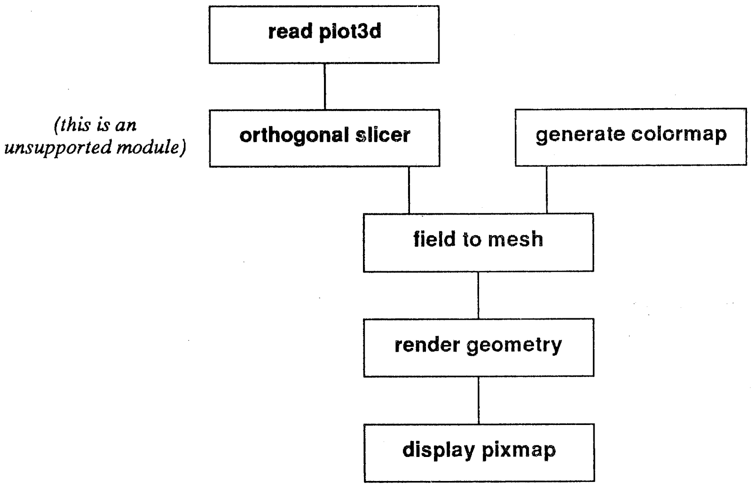
EXAMPLE 1

This example uses the "red band" (red component of the RGB color) of an image as a 2D field. It then converts this field to a mesh, using a colormap, and displays the mesh.



EXAMPLE 2

This example shows how to take orthographic slices through a curvilinear data set, showing them as XYZ meshes:



LIMITATIONS

This module can output meshes that are too big for the **render geometry** module to handle, causing AVS to crash. Use the **downsize** filter module to reduce the size of the input data.

flip normal

change direction of each vertex normal

SUMMARY

Name	flip normal
Type	filter
Inputs	geometry
Outputs	geometry
Parameters	none

DESCRIPTION

The **flip normal** module transforms an AVS *geometry* so that all the vertex normals point in the opposite direction. This is most often used to correct normals that have been calculated incorrectly.

When its normals are backwards, a 3D object appears unaffected by light sources; it frequently appears as a grey silhouette.

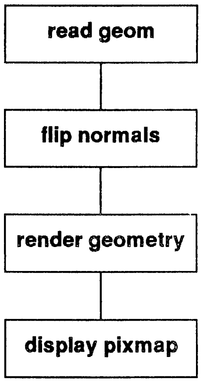
INPUTS

Geometry The output can be any AVS *geometry.*

OUTPUTS

geometry The output is an AVS *geometry* that represents the same object.

EXAMPLE



RELATED MODULES

read geom, offset, shrink, tube, render geometry

NOTES

Some filter modules (e.g. **offset**) sometimes produce bad normals, which can be corrected with **flip normal.**

generate colormap

output AVS colormap

SUMMARY

Name	generate colormap				
Type	data				
Inputs	none				
Outputs	colormap				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	colormap	colormap	(-- NOT APPLICABLE --)		
	lo value	float	0	none	none
	hi value	float	255	none	none

DESCRIPTION

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

colorizer
arbitrary slicer
bubbleviz
field to mesh
isosurface

This module bases its output colormap on the state of the *colormap editor* control widget. The colormap editor uses a *hue-saturation-brightness* (HSB) color space model:

hue	0.00 = red
	0.16 = yellow
	0.33 = green
	0.50 = cyan
	0.66 = blue
	0.83 = magenta
saturation	0.00 = white
	1.00 = **hue**
brightness	0.00 = black
	1.00 = **hue**

The HSB color space can be thought of as an inverted cone:

❑ The **hue** axis runs circularly around the cone.

❑ The **saturation** axis runs from the center of the cone (white) to its perimeter (fully saturated color).

❑ The **brightness** axis runs from the tip of the cone (black) to the base (white).

PARAMETERS

colormap The state of the colormap editor control widget specifies the colormap to be generated. This widget has an *editing panel* and eight buttons:

hue Raises the **hue** editing panel. The default panel is a linear ramp: 0=blue through 255=red.

saturation Raises the **saturation** editing panel. The default panel has all colors fully saturated: 0–255 = 1.0.

brightness Raises the **brightness** editing panel. The default panel has all colors at full brightness: 0–255 = 1.0.

opacity Raises the **opacity** editing panel. (The **opacity** value is placed in the *auxiliary* field of the colormap.) The default panel is a linear ramp: 0=0.0 through 255=1.0.

invert Inverts the currently raised editing panel, swapping high values with low values: the value at 0 becomes the value at 255, the value at 1 becomes the value at 254, and so on.

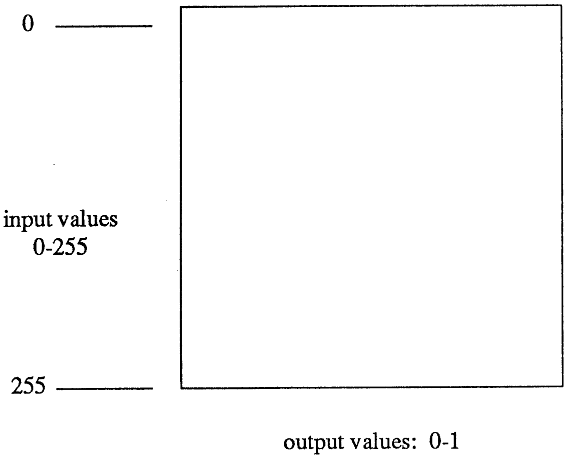
ramp Generate a linear ramp on the currently raised editing panel: 0=0.0 through 255=1.0.

read Reads a colormap from disk storage. Pressing this button pops up a File Browser widget, allowing you to specify a filename. You can also change the working directory.

write Write the current colormap to a disk file. Pressing this button pops up a File Browser widget, allowing you to specify a filename. You can also change the working directory.

You can change an editing panel from its current setting by scribing a curve with the mouse. Place the mouse cursor anywhere within the editing panel, hold down any mouse button, and drag upward or downward.

Each editing panel is organized as follows:



lo value (see *LIMITATIONS* below) Specifies the minimum data value that can be used as input to the colormap (the value at the top of the editing panel). The default low value is 0.

hi value (see *LIMITATIONS* below) Specifies the maximum data value that can be used as input to the colormap (the value at the bottom of the editing panel). The default high value is 255.

OUTPUTS

colormap The output is an AVS *colormap*.

COLORMAP FILE FORMAT

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
```

The hue, saturation, brightness, and opacity values are normalized to the range 0.0 – 1.0. The default colormap has 255 entries, with the hue, saturation, brightness, and opacity default values as described above.

LIMITATIONS

The **colormap** can only generate colormaps with 255 entries.

geom to scatter

convert geometry to point list

SUMMARY

Name	geom to scatter
Type	filter
Inputs	geometry
Outputs	field 1D 3-coordinate real irregular
Parameters	radius

DESCRIPTION

The **geom to scatter** module converts an AVS *geometry* to a point list of its vertices (a "scatter"). This is represented as a "field 1D 3-coordinate float irregular".

INPUTS

Geometry The input can be any AVS *geometry.*

PARAMETERS

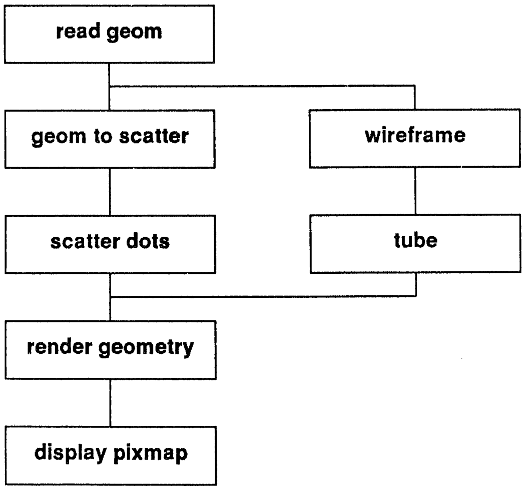
radius A value to be placed in the output field as the data value of each element. (In the example below, this value is not used by the **scatter dots** module.)

OUTPUTS

Data Field (field 1D 3-coordinate real irregular)
The output field is a list of points in 3D space — the vertices of the input geometry.

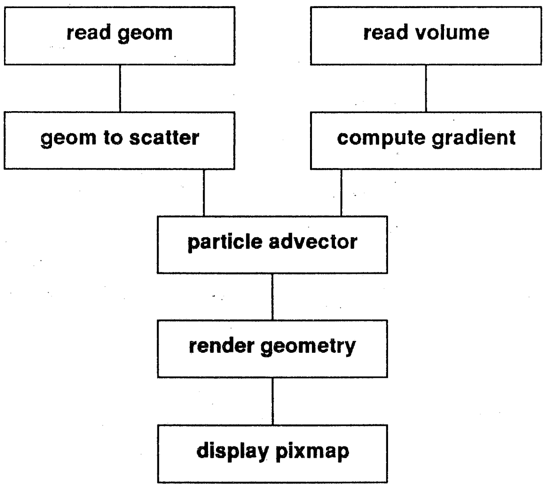
EXAMPLE 1

This example turns a geometry into a ball-and-stick representation, using spheres and tubes:



EXAMPLE 2

This example uses **geom to scatter** to provide the input "particles" (points) to the **particle advector** module. (It is also use for supplying input to the **hedgehog** module.)



RELATED MODULES

read geom, offset, shrink, tube, wireframe, hedgehog, particle advector, render geometry

gradient shade

apply lighting and shading to colored data set

SUMMARY

Name gradient shade

Type filter

Inputs field 4-vector byte *(colorized data)*
field 3-vector real *(gradient supplied by* **compute gradient***)*

Outputs field 4-vector byte *(shaded version of colorized data)*

Parameters

Name	*Type*	*Default*	*Min*	*Max*
ambient		0.1	0.0	1.0
diffuse		0.8	0.0	1.0
specular	float	0.0	0.0	1.0
gloss		20.0	0.0	50.0
lt theta	float	0.0	none	none
lt off-ctr	float	0.0	none	none

DESCRIPTION

The **gradient shade** module accepts a colored 2D or 3D data set, along with its gradients (supplied by the **compute gradient** module). It applys a single light source to the colored data, then shades it.

The gradient at each location in the data field substitutes for the *surface normal*, which is used in traditional algorithms for lighting and shading surfaces. (A surface normal at a particular point on a surface is a vector perpendicular to the surface.)

Various shading styles are achievable using the lighting controls (see *PARAMETERS* below). These include creating shiny and matte surfaces, and controlling the location of the light source.

INPUTS

Data Field (required; field 2D/3D 4-vector byte)
The input field is an image (2D pixel array) or a block of voxels (3D pixel array).

Gradient (required; field 3D 3-vector real)
This field is the gradient of the **Data Field.**

PARAMETERS

The way in which all the following parameters determine the coloring of an object is described below.

ambient The contribution of ambient (uniform background) lighting to the color. When this is set to 0.0, all surfaces facing away from the light source are black. When this is set to 1.0, surfaces appear in their own colors, with no shading information present.

diffuse The contribution of diffuse (directional) lighting to the color.

specular The contribution of specular lighting to the color.

gloss The sharpness of the specular highlight. The larger this value, the smaller and sharper the specular highlights.

lt off-ctr The angle between the light source and the positive Z axis (which comes out of the screen at a right angle).

lt theta The angle between (1) the projection of the light source on the X-Y plane and (2) the positive Y axis. This value measures how much an off-center light source "swings around" the Z-axis.

With *lt theta* = 0.0 and *lt off-ctr* = 0.0, the light source is coming straight from the eye perpendicular to the data. A positive *off-ctr* value moves the light source "up" (in the positive Y direction); a negative value moves it "down".

The equation for calculating the intensity of light reflected by a spot of surface is: $(int_{amb} * ambient) + (int_{diff} * diffuse * \cos(phi)) + (int_{diff} * specular * \cos^{gloss}(lt\ off\text{-}ctr))$
In performing this computation, **gradient shade**:

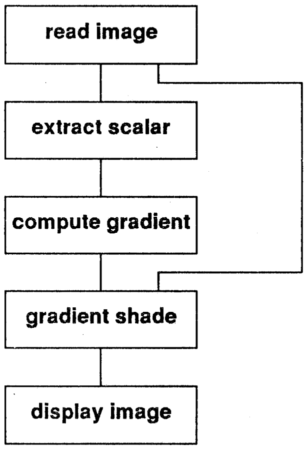
- ❑ Assumes that int_{amb} and int_{diff} are both maximal (1.0).
- ❑ Uses *lt theta* and *lt off-ctr* to compute *phi*, the angle between the surface normal (gradient vector) and the light source. The quantity cos(*phi*) is the attenuation (reduction) factor for the directional (diffuse) light.
- ❑ Computes the quantity $\cos^{gloss}(\alpha)$, the attenuation factor for the specular highlight.

OUTPUTS

Data Field (field *same-dimension* 4-vector byte)
The output field has the same form as the **Data Field** input.

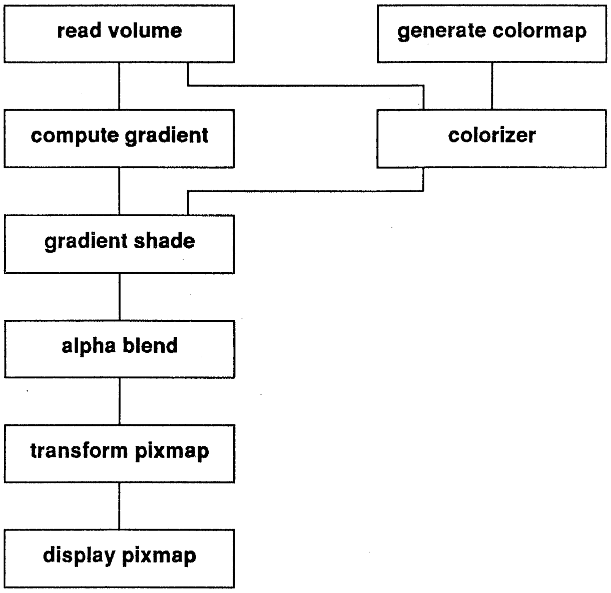
EXAMPLE 1

The following network shades a 2D image:



EXAMPLE 2

The following network shades a 3D image:



RELATED MODULES

Modules that could provide the **Data Field** input:

read volume

Modules that could provide the **Gradient** input:

compute gradient

Modules that could be used in place of **gradient shade**:

colorizer

Modules that can process **gradient shade** output:

display image (2D data)
alpha blend (3D data)

See also **extract scalar**, which gets a single scalar height field from an image.

hedgehog

show vectors on a slice of a 3D 3-vector field

SUMMARY

Name	hedgehog
Type	mapper
Inputs	field 3D 3-vector float
Outputs	geometry

Parameters

Name	*Type*	*Default*	*Min*	*Max*
Vector Scale	float	1.0	0.01	10.0
Mesh Res	integer	16	2	128
X Rotation	float	0.0	0.0	360.0
Z Rotation	float	0.0	0.0	360.0
Distance	float	0.0	–2.0	2.0

DESCRIPTION

The **hedgehog** module takes as input a 3D field whose values are 3-vectors of *floats*. That is, the data represents a volume of lattice points, each point having a 3D vector of *float* values. This 3D-vector value can be visualized as a small line segment with a particular length and direction.

The **hedgehog** module takes an arbitrarily-oriented (parameter-controlled) slice through the volume, outputting the line segments that fall on the slice plane. The collection of line segments resembles the coat of a hedgehog — hence the module's name.

The slice plane is represented as a 2D grid, with a parameter-controlled resolution. The intersection of the data volume and the grid is a *mesh* of vertices in 3D space.

For each vertex in the mesh, a small line segment is created. (If a mesh point falls outside of the volume, no segment is created.) The length and orientation of the segment depends on the nearby 3D vector values in the volume. (Since, in general, the mesh vertices do *not* coincide with the lattice points in the data volume, a trilinear interpolation method is used to average the 3D-vector values of nearby lattice points. See **arbitrary slicer** for more on trilinear interpolation.)

INPUTS

Volume Data (required; field 3D 3-vector float)

The input data must be a 3D field, representing a volume of points. The data value for each point must be a 3D vector of *floats*.

PARAMETERS

Vector Scale

The lengths of the line segments output by this module are proportional to this value.

Mesh Res Controls the resolution of the slice plane mesh. Higher resolution meshes result in higher quality representations, but take longer to compute and render. The default mesh is 16x16.

X Rotation Rotates the mesh along the X-axis (first dimension) of the volume.

Y Rotation Rotates the mesh along the Y-axis (second dimension) of the volume.

Distance Performs a translation perpendicular to the mesh plane.

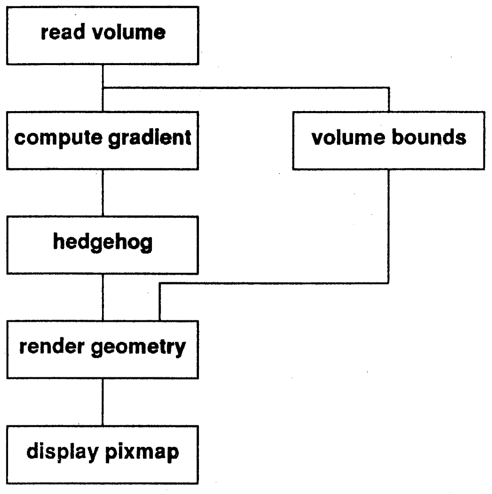
OUTPUTS

Hedgehog (geometry)

The output *geometry* is a collection of line segments that represent the 3D-vector values along the slice plane.

EXAMPLE

The following network visualizes the output of the **compute gradient** module.



NOTES

The **hedgehog** module is useful for visualizing the magnitude of wind velocities. No supported modules generate this data type, however.

RELATED MODULES

Data input:
  read volume, volume manager
Gradient computation:
  compute gradient
Vector operations:
  vector curl, vector div, vector grad, vector mag, vector norm
Additional geometries:
  volume bounds, isosurface
Geometric rendering:
  render manager,
  render geometry,
  display pixmap

histogram stretch

balance the histogram of a data set

SUMMARY

Name	histogram stretch				
Type	filter				
Inputs	field *any-dimension* scalar byte				
Outputs	field of same type as input				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	histr_min	int	0	0	255
	histr_max	int	255	0	255

DESCRIPTION

histogram stretch is an image/volume processing module that balances the "histogram" of a data set between specified values. This operation combines histogram balancing (also called "histogram normalization" or "histogram equalization") and contrast stretching.

Finding the *histogram* of an image (or volume) consists of tallying the number of pixels (voxels) of each value into "bins". Byte data typically generates 256 bins (1 bin for each possible data value).

The *histogram equalization* process consists of trying to establish the same number of pixels (voxels) per bin by translating the pixel (voxel) values, using a well-chosen lookup table. This has the effect of creating an even distribution of values throughout the data set. It typically used to enhance low-contrast images (volumes) or images in which the data is "bunched up" at one end of the spectrum.

Equalization is applied only to values within the range specified by the parameters **histr_min** and **histr_max**. Data outside this range is not included in the histogram generation, and is eliminated.

INPUTS

Data Field (required; field *any-dimension* scalar byte)

The input data may be an AVS field of any dimensionality, each of whose values is a scalar *byte*.

PARAMETERS

histr_min Specifies the bottom of the range of input values that will be histogrammed, then transformed.

histr_max Specifies the top of the range of input values that will be histogrammed, then transformed.

OUTPUTS

Data Field The output field has the same form as the input field.

LIMITATIONS

This module works for *byte* fields only. (For other data types, there is no general way to determine the "right" number of bins to generate.) To apply this module to non-*byte* data, use the **field_to_byte** module to pre-process the data.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume
field to byte

Modules that could be used in place of **histogram stretch**:

contrast stretch

histogram stretch

Modules that can process **histogram stretch** output:

field to integer
field to float
field to double
any other filter module

image manager

share images among subnetworks

SUMMARY

Name	image manager		
Type	data		
Inputs	none		
Outputs	field 2D 4-vector byte (image)		
Parameters	*Name*	*Type*	*Choices*
	IMAGMGR Select	choice	Select, Replace
	Image Manager	browser	
	Image Choices	choice	

DESCRIPTION

The **image manager** module reads an image file from disk and outputs the image as a "field 2D 4-vector byte". It works like the **read image** module, except that it has both a cacheing mechanism and a way of sharing data among **image manager** modules in separate subnetworks.

See the **read image** manual page for a description of the image format.

PARAMETERS

IMAGMGR Select

A choice that determines how newly-read images will be placed to the list of currently active images:

- □ If **Select** is chosen, a new image is added to the end of the list.
- □ If **Replace** is chosen, a new image replaces the currently selected member on this list.

In either case, the change is reflected in *all* the *image manager* modules in all active subnetworks.

Image Manager

A file browser that allows you to select an image file to read.

Image Choices

A set of choices, listing each of the currently active images.

OUTPUTS

Data Field (field 2D 4-vector byte)

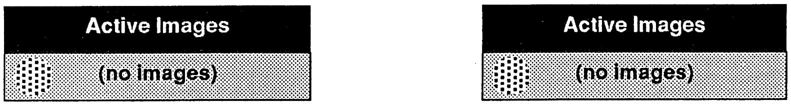
The output is an AVS *image*.

EXAMPLE

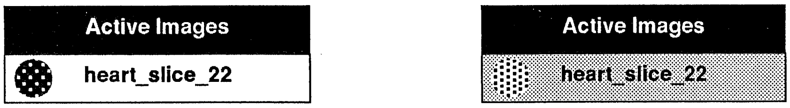
The following subnetworks might be used to display two images:

image manager
display image
image manager
display image

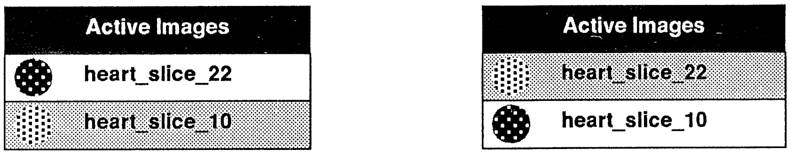
In this case, both **image manager** managers would contain "select/replace" choice buttons, a file browser, and an area below the browser:



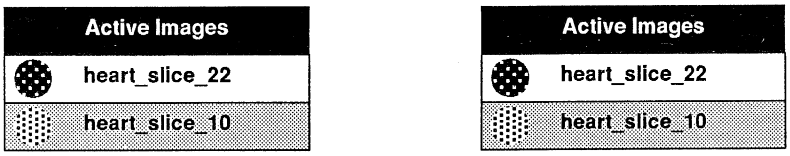
Once a file (e.g. *heart_slice_22*) is selected with the browser in the **image manager** on the left, these buttons would look like this:



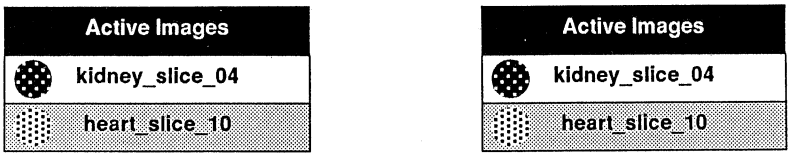
If a different file (e.g. *heart_slice_10*) is chosen from the browser in the **image manager** on the right, the buttons would look like this:



By selecting the same active image, you can have both networks display the same image:



Now, if you want to replace this image with a new one, click on the **Replace** buttons above the browser, then select a new file (e.g. *kidney_slice_04*) in just one of the **image manager** browsers. The result is that all **image manager** modules with the old image (*heart_slice_22*) selected as their active image will be automatically updated with the new image (*kidney_slice_04*):



RELATED MODULES

Same as for **read image**.

LIMITATIONS

The cached images are not freed until all *image manager* modules are destroyed. This is not the case with **read image** —r the old data is freed whenever a new file is read.

image to pixmap

convert image to pixmap

SUMMARY

Name	image to pixmap			
Type	mapper			
Inputs	field 2D 4-vector byte			
Outputs	pixmap			
Parameters	*Name*	*Type*	*Default*	*Choices*
	Approximation Technique (16-plane system only)	choice	none	none, dithering random, monochrome

DESCRIPTION

The **image to pixmap** module takes as input an *image* ("field 2D 4-vector byte") and outputs the same image as a *pixmap*. It is useful for converting the output of modules that produce images into modules that require pixmaps.

The *image* and *pixmap* data types differ in these major ways:

- ❑ Images are allow for efficient direct manipulation by a module, whereas pixmaps allow for efficient manipulation by the display device.
- ❑ Pixmaps are directly usable by a display device (under control of the X server). In X terminology, pixmaps contain "pixel values", images contain "colors". This difference is important only for 16-plane pseudo-color systems, in which pixmap values are interpreted as indices into the system's color lookup table. An image contains 24-bit color values, which cannot be used on such systems, which have only 12 color planes.
- ❑ A pixmap is represented by an X Window System *resource id* (an integer). This means that transferring a pixmap from one module to another is more efficient than transferring all the data that defines an image.

See the **read image** manual page for a description of the AVS image format.

INPUTS

Data Field (required; field 2D 4-vector byte)
The input field must be an AVS *image*.

PARAMETERS

This module has the following parameter only when running on a 16-plane system.

approximation technique (16-plane systems only)
Controls the conversion of color values to pixel values. There are four approximation techniques:

- ❑ **dithering**: uses a dither matrix to approximate each color
- ❑ **random**: uses a random number dither to approximate each color
- ❑ **monochrome**: uses the luminance of the color as an index into a greyscale ramp
- ❑ **none**: takes the closest approximation for each color

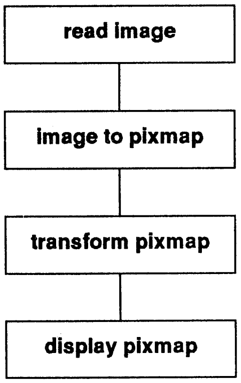
OUTPUTS

pixmap The output is an AVS *pixmap*.

EXAMPLE

This network allows an image to be displayed in an arbitrary-sized window:

image to pixmap



RELATED MODULES

pixmap to image, transform pixmap, display pixmap

interpolate

change the size of the data

SUMMARY

Name	interpolate					
Type	filter					
Inputs	field 2D/3D scalar byte					
Outputs	field *same-dimension* scalar byte					
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*	*Choices*
	interp_sx	float	1.0	0.0	4.0	
	interp_sy	float	1.0	0.0	4.0	
	interp_sz	float	1.0	0.0	4.0	
	sampling	choice	Point			Point, Bi/Trilinear

DESCRIPTION

The **interpolate** module arbitrarily changes the size of its input data, either by subsampling or interpolating it. This module is useful for smoothly scaling the data arbitrarily up and down.

The interpolation algorithm first selects, for each output point, its real (floating-point) position in the input data set:

```
New X = Old X * interp_sx
New Y = Old Y * interp_sy
New Z = Old Z * interp_sz
```

With the *point sampling* method, it then selects the closest pixel (voxel) to the computed one. With *bilinear* (in 2D) or *trilinear* (in 3D) sampling, it finds the four pixels (2D) or eight voxels (3D) around the computed point and does a linear sampling for in-between pixels.

The point sampling mode is much quicker than the linear sampling and should be used when interactivity is more important than image quality.

INPUTS

Data Field The input field may be 2D or 3D. The data for each element must be a single byte.

PARAMETERS

interp_sx
interp_sy
interp_sz The interpolation factors for the coordinate dimensions.

sampling This choice determines the sampling method, **Point** or **Bi/Trilinear**, as described above.

OUTPUTS

Data Field The output field has the same form as the input field.

RELATED MODULES

This module is similar to **downsize** (which does uniform, stride-based point sampling) and **crop** (which selects a subset of the data but doesn't change the resolution). Some advantages to using this module are: it can scale non-uniformly in each dimension; it can do high-quality linear sampling; and it can scale data up instead of only down.

LIMITATIONS

This module does the wrong thing when down-sampling (going from a large image to a small one) in the Bi/Trilinear mode. What it should do is "average" appropriately chosen regions down to each pixel. What is does is to choose the four pixels around the center of that region and interpolate between them. This is not a huge error, but it is not strictly correct.

isosurface

generate an isosurface for a volume of data

SUMMARY

Name	isosurface	
Type	mapper	
Inputs	field 3D scalar real field 3D scalar real (optional) colormap (optional)	
Outputs	geometry	
Parameters	*Name*	*Type*
	Isosurface Level	float
	Optimize Surface Description	toggle
	Optimize Wireframe Description	toggle
	Flip Normals	toggle

DESCRIPTION

The **isosurface** module inputs a volume data set (3D field of floating-point values, either curvilinear, rectilinear, or uniform). It produces a geometric object that represents an isosurface of this object. An *isosurface* is a 3D generalization of a 2D contour line — it connects all field elements that have the same parameter-controlled data value.

INPUTS

Data Field (required; field 3D scalar real)

The input data must represent a volume, with a single floating-point value for each field element.

Auxiliary Data Field (optional; field 3D scalar real)

This port can be used to generate a colored isosurface; the color at each point on the surface indicates the value of another attribute of the volume. For instance, you could generate a pressure isosurface with colors indicating the temperature at each point on the surface.

In this case, the **Data Field** would be used to input the pressure data, and the **Auxiliary Data Field** would be used to input the temperature data. In all cases, both volume data sets must have the same dimensions.

Colormap (optional; colormap)

If you use an **Auxiliary Data Field,** you must also specify a colormap. Since the auxiliary volume data is floating-point, you must adjust the **lo value** and **hi value** parameters of the **generate colormap** module to correspond to the minimum and maximum data values of the auxiliary field.

For the pressure-temperature example described above, the temperature data set might have data values in the range 0.0–100.0 degrees. In this case, set the **lo value** to 0 and **hi value** to 100 in **generate colormap**.

PARAMETERS

Isosurface Level

A floating-point value that specifies the common data value on the isosurface: for each point on the isosurface, the field element's data value equals the **Isosurface Level** value.

Optimize Surface Description
Optimize Wireframe Description

These two toggle parameters allow you to control a tradeoff between how efficiently the isosurface is computed and how efficiently it can be rendered. If you turn on **Optimize Surface**, extra time will be spent generating a more optimal surface description, containing fewer triangles.

Turn on **Optimize Wire** to generate a wireframe representation for the isosurface along with the shaded surface representation. If you want to view your surface as a wireframe (using the **Objects** selection in the **render geometry** control panel), you must toggle this on.

Flip Normals

Reverses the direction of each surface normal in the generated isosurface. If the normals point in the wrong direction, the outside of the isosurface will appear at the ambient light intensity. In this case, click this button or specify bi-directional lighting in the **render geometry** control panel (**Lights** selection).

OUTPUTS

Isosurface (geometry)

A shaded surface, optionally with an associated wireframe representation.

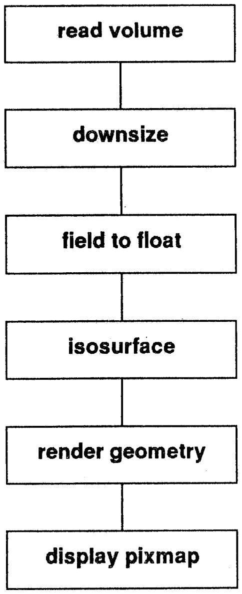
NOTES

The most important parameter is the **Isosurface Level** (threshold), which is defined in the unbounded floating-point data space of the volume. It is not always easy to know in what range the data is defined. Often, the data is defined some well-known real-world domain (e.g. temperature in degrees). In some cases, the data has been converted from *byte* data, and therefore must lie within the range 0.0–255.0.

Because **isosurface** can take a long time to compute, it is often advisable to include a **downsize** module in the network. This allows you to quickly select a proper isosurface level before generating one at full resolution.

Another technique is to use the **Action** capability of the Geometry Viewer (**render geometry** module) to save and play back a sequence of isosurfaces at different value levels.

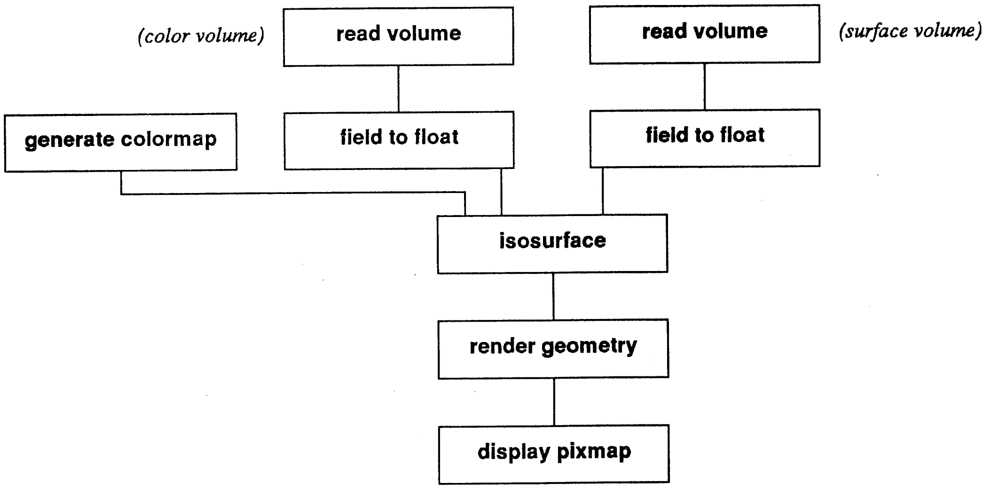
EXAMPLE 1



EXAMPLE 2

This example uses an auxiliary data set.

isosurface



RELATED MODULES

render geometry, downsize, generate colormap, field to float, read field, read volume

LIMITATIONS

In some circumstances, the generated isosurface may have some of its normals pointing inward and some outward. There is no way to correct this situation, but usage of bi-directional lighting (**Lights** selection of the Geometry Viewer/**render geometry**) may be helpful.

mirror

reverse array indices in a 2D or 3D data set

SUMMARY

Name	mirror
Type	filter
Inputs	field 2D/3D *any-data*
Outputs	field of same type as input

Parameters

Name	*Type*	*Default*	*Choices*
axis	choice	Original	Original, X, Y, Z

DESCRIPTION

The **mirror** module reverses the array indexes along one dimension of a 2D or 3D field. This has the effect of creating a mirror image of the data set. In a 50 x 100 field, applying **mirror** to the X dimension does the following (in FORTRAN array notation):

```
INPUT(1,i)  ---> OUTPUT(50,i)    (for all 100 values of i)
INPUT(2,i)  ---> OUTPUT(49,i)
INPUT(3,i)  ---> OUTPUT(48,i)
INPUT(4,i)  ---> OUTPUT(47,i)
  ...
INPUT(50,i)  ---> OUTPUT(1,i)
```

mirror can be used to change the orientation of the data for display and/or processing purposes.

To perform a reversal in two or more dimensions, use two or more **mirror** modules in succession.

INPUTS

Data Field The input may be any AVS *field.*

PARAMETERS

axis The choices for exchanging the data are:

Original Copies the input to the output; no transformation is performed.

X Reverses the array indices in the X dimension (first dimension).

Y Reverses the array indices in the Y dimension (second dimension).

Z Reverses the array indices in the Z dimension (third dimension). (Equivalent to **Original** for a 2D field.)

OUTPUTS

Data Field The output field as the same form as the input field.

RELATED MODULES

This module combined with **transpose** can re-orient the data in any desired way.

offset

make an object smaller

SUMMARY

Name	offset				
Type	filter				
Inputs	geometry				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	offset	float	0.0	none	none

DESCRIPTION

The **offset** module transforms an AVS *geometry*, so that each vertex of each polygon is translated along its vertex normal. It is useful for emphasizing surface discontinuities (e.g. cusps) and for producing "blow ups" of objects.

INPUTS

Geometry (required; geometry) An AVS geometry, created with the *libgeom* library or by another AVS module.

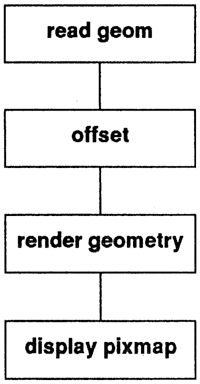
PARAMETERS

offset The amount by which each vertex is translated along its normal. Positive values create a "blow-up" of the geometry. Negative values collapse it.

OUTPUTS

Geometry A geometry that represents that same object(s) as the input data.

EXAMPLE



RELATED MODULES

read geom, flip normal, tube, render geometry

LIMITATIONS

This module works only for polytriangle strips and meshes, not for polyhedra. It has no effect on objects that do not have surface normals.

orthogonal slicer

slice through volume data with plane perpendicular to coordinate axis

SUMMARY

Name	orthogonal slicer					
Type	mapper					
Inputs	field 3D *any-values*					
Outputs	field 2D *same-values*					
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*	*Choices*
	slice plane	int	0	0	255	
	axis	choice	K			I, J, K

DESCRIPTION

The **orthogonal slicer** module takes a 2D slice from a 3D array. It does so by holding the array index in one dimension constant, and letting the other indices vary. For instance, a data set might include a volume of 5000 points, arranged as follows (using FORTRAN notation):

```
DATA(I,J,K)          I = 1,10
                     J = 1,20
                     K = 1,25
```

You can take a 2D "I-slice" from this data set by setting *I*=4 and letting the other indices vary:

```
DATA(4,J,K)          J = 1,20
                     K = 1,25
```

The notation used in the example above assumes that the field's data values are scalars (in FORTRAN, DATA(4,5,6) must be a scalar). If fact, however, the **orthogonal slicer** module can takes slices of vector-valued fields, also. It passes through whatever data type is presented to it; e.g. if the input is a "field 3D 3-vector float", the output is a "field 2D 3-vector float".

INPUTS

Data Field The input may be any 3D *field*.

PARAMETERS

slice plane Determines the value of the array index to be held constant.

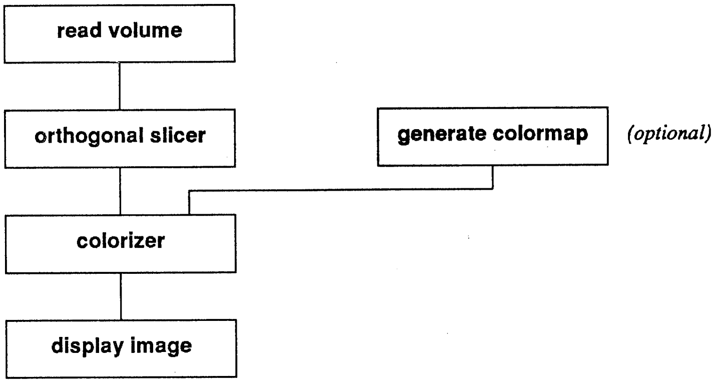
axis Selects the dimension (X, Y, or Z) in which the array index is to be held constant.

OUTPUTS

Data Field The output field is 2D instead of 3D, and has the same type of data as the input field.

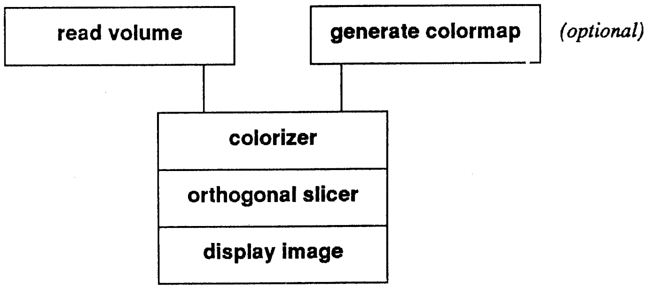
EXAMPLE

The following network takes a slice from a scalar volume and displays it:



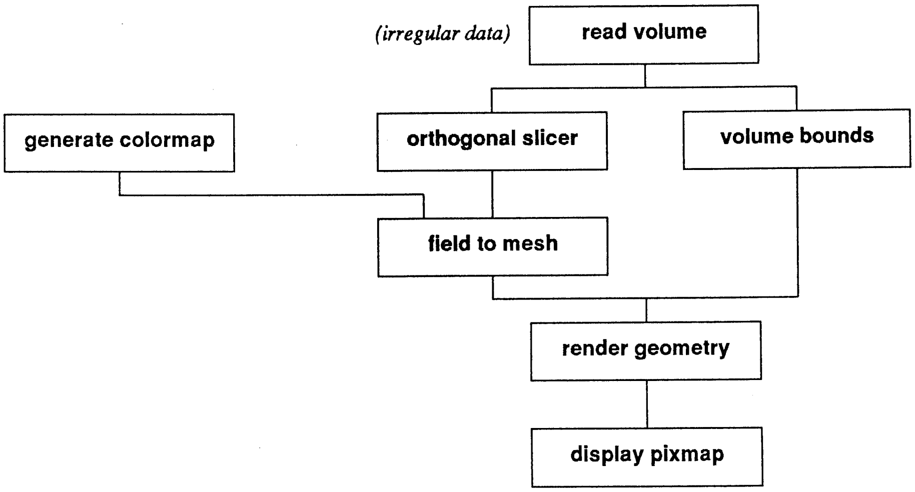
The **colorizer** module is necessary because the output of **orthogonal slicer** is a "field 2D scalar byte", which must be cast into an AVS *image* field for display.

For reasonably small volumes, a better way to construct this network is:



This network has the effect of colorizing the entire volume once, which make the slicing operation more efficient. It does this at the expense of allocating more memory up front.

Irregular Data: orthogonal slicer supports the passing of "points" data for *rectilinear* and *irregular* data. This is an important module for visualizing curved data sets. For example:



(This is the reason for labeling the axis control with "I, J, and K": frequently, the data is *not* aligned to the X, Y, and Z axes. **orthogonal slicer** takes slices through the logical data set, not the physical one.)

output postscript

convert pixmap to PostScript™ and store in file

SUMMARY

Name	output postscript	
Type	renderer	
Inputs	pixmap (required; pixmap)	
Outputs	*none*	
Parameters	*Name*	*Type*
	filename	browser
	mode	choice
	monochrome	toggle
	8 bit	toggle
	compress	toggle
	dither	toggle

DESCRIPTION

The **output postscript** module converts its input pixmap to the PostScript™ page description language and stores it in a file.

After the file is written, the filename is reset to NULL. This prevents subsequent changes upstream in the network from automatically triggering the rewriting of the file. A new file is written only when you enter a filename.

Two types of PostScript output are supported:

❑ Suitable for sending to a PostScript-compatible laser printer.

❑ Mathematica™ compatible.

INPUTS

pixmap (pixmap)

Any AVS pixmap.

PARAMETERS

filename A file browser that allows you to specify the name of the PostScript file to be created. After the file is written, the filename is reset to NULL. This prevents subsequent changes upstream in the network from automatically triggering the rewriting of the file. A new file is written only when you enter a filename.

Mode Selects the type of PostScript output: **laserwriter** or **mathematica**.

The following toggle parameters control the creation of Mathematica PostScript files:

monochrome

If ON, produces monochrome output. If OFF, produces color output.

8 bit If ON, produces 8-bit output. If OFF, produces 4-bit output.

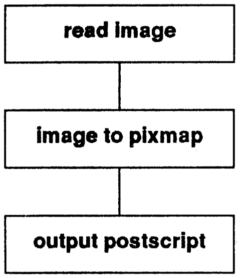
compressed If ON, produces compressed output. If OFF, produces uncompressed output.

dither If ON, produces dithered output. If OFF, produces undithered output.

EXAMPLE

This example converts an image to a PostScript file:

output postscript



RELATED MODULES

image to pixmap, render geometry, alpha blend

LIMITATIONS

This module does not work on a 16-plane system.

The Mathematica compress option is not supported in any released version of Mathematica.

The dither option produces visual artifacts on some images.

COPYRIGHT

Mathematica is a copyright of Wolfram Research.

particle advector

release grid of particles into velocity field

SUMMARY

Name	particle advector				
Type	mapper				
Inputs	field 3D 3-vector float				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Mesh Res	integer	16	2	128
	Time Step	float	0.0	0.0	1.0
	Size	float	0.0	0.0	1.0
	X Rotation	float	0.0	0.0	360.0
	Y Rotation	float	0.0	0.0	360.0
	Distance	float	0.0	–2.0	2.0
	Advect Batch	oneshot	(-- NOT APPLICABLE --)		
	Reset Particles	oneshot	(-- NOT APPLICABLE --)		
	Color	toggle	off		
	Show Bounds	toggle	on		
	Surface	toggle	off		
	Stop Advection	toggle	off		

DESCRIPTION

The **particle advector** module takes as input a 3D 3-vector field of *floats* (e.g. fluid flow simulation data), and treats it as a velocity field. A 2D mesh of zero mass particles is "advected" (placed into the field at various initial positions with no initial direction or speed). The particles move through the velocity field according to the magnitude and direction of the vectors at the nodes in the volume. A forward differencing method is used to estimate the next position of each particle as a function of the current position and velocity.

This module is an AVS *coroutine* — it generates new data continuously, rather than waiting for a module upstream to pass it new data.

INPUTS

Data Field (required; field 3D 3-vector float)

The input data must be a 3D field, representing a volume of points. The data value for each point must be a 3D vector of *floats*.

PARAMETERS

Various aspects of the particle advection process can be adjusted interactively.

Advect batch

Triggers the release of a batch of particles. The number of particles is controlled by the **mesh res** parameter. the total number in each batch is **mesh_res * mesh_res.**

Reset Particles

Sets the total number of particles to zero.

Time Step Adjusts a scalar that multiplies the magnitude of the vector along which each particle is travelling. This causes successive positions of particles to be more widely spaced. (See also the **Color** parameter.)

Stop Advection

Temporarily halts this module.

size Controls the radius of the particles, which are rendered as spheres.

X Rotation Rotates the mesh along the X-axis (first dimension) of the volume.

Y Rotation Rotates the mesh along the Y-axis (second dimension) of the volume.

Distance Performs a translation perpendicular to the mesh plane.

Show Bounds
(toggle) Controls the visibility of the mesh of particles.

Color (toggle) If **ON**, colors the line segments to indicate how fast the particles are travelling through the velocity field:

red	fastest backwards
yellow	backwards
green	slow backwards
cyan	not moving
blue	slow forwards
magenta	forwards
red	fastest forwards

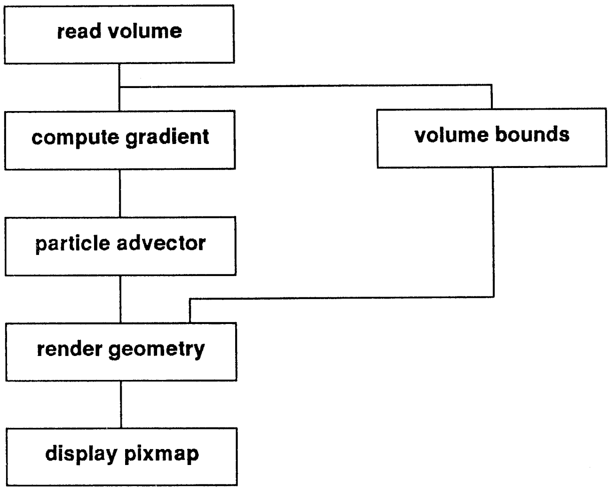
Surface Creates a solid shaded mesh. The coloring scheme is the same as that used with the **Color** parameter.

OUTPUTS

Geometry The output is an AVS *geometry* that represents the mesh of particles.

EXAMPLE

In the following network, the **read volume** and **compute gradient** modules should be replaced by a module that generates wind velocities or reads them from a data file. (No such module is currently supported in AVS.)



RELATED MODULES

Vector operations:

vector curl, vector div, vector grad, vector mag, vector norm

Additional geometries:

volume bounds
arbitrary slice
isosurface

Geometric rendering:

- render manager
- render geometry
- display pixmap

pdb to geom

create molecule geometry from PDB file

SUMMARY

Name	pdb to geom		
Type	data		
Inputs	none		
Outputs	geometry		
Parameters	*Name*	*Type*	*Choices*
	Render Mode	choice	ball and stick, ball, stick, colored stick, colored residue

DESCRIPTION

The **pdb to geom** module reads the description of a molecule from a file in the Brookhaven Protein Data Bank (PDB) data format. Typically, such files have a *.pdb* filename suffix. The output is an AVS *geometry* description of the molecule.

PARAMETERS

Data File The name of the *.pdb* file containing the module description.

Mode The type of geometry produced:

ball and stick
Small spheres represent the atoms, and white lines represent the bonds.

ball
Large spheres represent the atoms.

stick
White lines represent the bonds.

colored stick
Colored lines represent the atoms and their bonds.

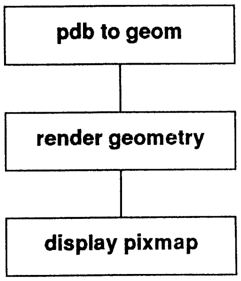
colored residue
Colored lines represent the atoms and their bonds. The color of the lines represents the type of amino acid that the molecule is in.

OUTPUTS

Molecule (geometry)
An AVS *geometry* description of the molecule.

EXAMPLE

This example shows a simple application of **pdb_to_geom**:



RELATED MODULES

render geometry

LIMITATIONS

If you read in the same *.pdb* file name twice, you will get only one instance of the geometry, not two.

Since the *.pdb* file does not contain any bond information, bonding is determined by the distances between atoms.

pixmap to image

transform AVS pixmap to AVS image

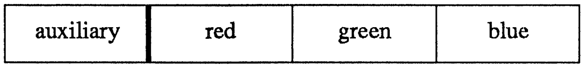
SUMMARY

Name	pixmap to image
Type	mapper
Inputs	pixmap
Outputs	image (field 2D 4-vector byte)
Parameters	none

DESCRIPTION

The **pixmap to image** module takes an AVS pixmap as input and outputs an AVS image ("field 2D 4-vector byte"). The pixmap is an X Window System resource used to store image data in the X server. This reduces the amount of data AVS must pass between modules: a pixmap id and window id.

The 4-vector byte representation for the image consists of pixels that look like this:



this field interpreted as
pixel's opacity value

these three fields make up
pixel's color value

The high-order byte field (auxiliary) is generally unused, but sometimes contains alpha (opacity) information on a per-pixel basis.

INPUTS

pixmap (required; pixmap)

The input is any AVS *pixmap.*

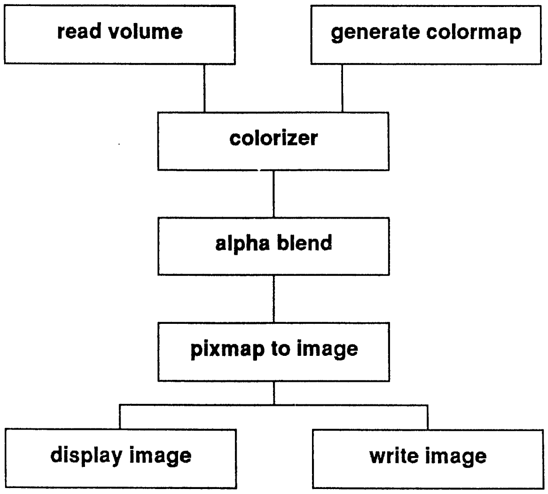
OUTPUTS

image (field 2D 4-vector byte)

The output data is a 2D block of pixels. The data set at each point of the 2D field will be a 4-vector of bytes in the AVS *image* format.

EXAMPLE

This module is useful for converting the output of renderers (e.g. **alpha blend**) into images for writing to a file.



RELATED MODULES

Image processing:

contrast, threshold, histogram stretch, clamp, interpolate, colorizer, generate colormap

Renderers which generate pixmaps:

render geometry, alpha blend, vbuffer

Display an image:

display image

Pixmap manipulation and display:

transform pixmap, display pixmap

LIMITATIONS

This module will not work for 16 plane systems. The "Refine" function in a **transform pixmap** module that is upstream of a **pixmap to image** module does not work.

read field

read AVS field from a disk file

SUMMARY

Name	read field	
Type	data	
Inputs	field *any-dimension any-type*	
Outputs	field	
Parameters	*Name*	*Type*
	Read Field	Browser

DESCRIPTION

The **read field** module reads an AVS *field* data structure from a disk file and sends the field to its output port. The on-disk field format includes two parts, an **ASCII header** and a **binary area**, with a special separator between them.

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. Comments also occur at the end of the 3rd and 6th lines. Any characters following (and including) # in a header line are ignored.

The 3rd through last lines of the header consist of TOKEN=VALUE pairs. This example shows all the token names, and all of them are required: an ASCII header that is missing one or more of these lines causes **read field** to generate an error. The tokens and their permitted values are described below. Note that ASCII header specifications are *not* case-sensitive. You can surround the = character with any amount of whitespace (including none at all).

ndim = *value*

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

dim1 = *value*
dim2 = *value*
dim3 = *value*
...

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 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.

nspace = *value*

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*

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
data = integer
data = float
data = double

The primitive data type of all the data values.

field = uniform
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.

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 size (in bytes) of the binary area depends on the field type:

❑ For **uniform** fields, the binary area contains data values only (no coordinates). Thus, the size is the product of the following numbers:

value of **dim1**	(product of sizes of computational dimensions
value of **dim2**	yields total number of field elements)
...	
value of **dim***x*	
value of **veclen**	*(number of data values per field element)*
size of **data**	*(byte size of primitive data type)*

In the stream of data values:

❑ All the data values for a field element are stored together.

❑ The first array index varies most quickly (FORTRAN-style).

❑ For **rectilinear** fields, the binary area contains both data values and coordinates. The data values occupy the same amount of space as for a **uniform** field. Each coordinate is a single-precision floating-point number (4 bytes), and there is one coordinate for each array index in each dimension of computational space. Thus, the size of the coordinates area is:

$(\mathit{dim1} + \mathit{dim2} \ldots + \mathit{dimx}) * 4$

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, the data area contains both data values and coordinates. The data values occupy the same amount of space as for a **uniform** field. Each coordinate is a single-precision floating-point number (4 bytes), and each field element is mapped to a point in *nspace*-dimensional physical space. Thus, the size of the coordinates area is:

$(\mathit{dim1} * \mathit{dim2} \ldots * \mathit{dimx}) * \mathit{nspace} * 4$

As with **rectilinear field**, all of the X-coordinates are stored together, at the beginning of the coordinates are. 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.

```
# 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 is null.

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 the wind velocity at each point in space.

```
# 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 is null.

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 fluid flow data.

```
# 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=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

RELATED MODULES

Many modules can process the output of **read field**.

The **write field** module writes to a disk file any field in the format described above.

The following *filter* modules accept a field as input:

clamp
colorizer
combine scalars
compute gradient
contrast
crop
dot surface
downsize
extract scalar
field to byte
field to double
field to float
field to int
gradient shade
histogram stretch
interpolate
mirror
threshold
transpose
vector curl
vector div
vector grad
vector mag
vector norm

The following *mapper* modules accept a field as input:

arbitrary slicer
bubbleviz
field to mesh
hedgehog
image to pixmap
isosurface
orthogonal slicer
particle advector
scatter dots
stream
lines
stream mesh
volume bounds

The following *render* modules accept a field as input:

alpha blend
display image
write image
write volume

ERROR CHECKING

read field performs a significant amount of error checking. If an error is detected while reading the field, an error dialog box appears on the screen, indicating the line in which the error occurred (if it was in the ASCII header), along with the type of error.

read geom

reads a data file containing an AVS ´geometry´

SUMMARY

Name	read geom	
Type	data	
Inputs	*none*	
Outputs	geometry	
Parameters	*Name*	*Type*
	Read Geometry	browser

DESCRIPTION

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. Since AVS replaces geometries based on the object name, if you read in the same filename twice, you will only get one representation of the object.

Since the Geometry Viewer subsystem (also accessible as the **render geometry** module) has a built-in **Read Object** function, you rarely need to use this module. It is most useful when used in conjunction with a filter module that processes geometric data (e.g. **shrink**).

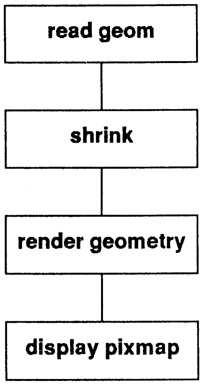
PARAMETERS

filename A file browser allows you to specify the name of the file that contains an AVS *geometry.*

OUTPUTS

geometry The output is the *geometry* that was read from the specified file.

EXAMPLE



RELATED MODULES

shrink, offset, render geometry, wireframe, tube

LIMITATIONS

This module reads GEOM-file files only. It cannot read *.obj* script files or *.scene* scene files that can be created with the Geometry Viewer Script Language (see Appendix B).

The object is always named after the file from which it is read. This makes it awkward to create animation loops, for which you might want to direct multiple files to the same name or to read in multiple instances of the same object.

read image

read image file from disk into a field

SUMMARY

Name read image

Type data

Inputs none

Outputs field 2D 4-vector byte

Parameters

Name	*Type*	*Default*	*Min*	*Max*
Read Image	Browser	not applicable		

DESCRIPTION

The **read image** module reads an image file from disk and outputs the image as a "field 2D 4-vector byte". Each field element represents a pixel. The data value for each element is a 4D vector of bytes, laid out as follows:

auxiliary	red	green	blue

this field interpreted as pixel's opacity value

these three fields make up pixel's color value

The auxiliary field ("alpha") is sometimes used to store opacity information on a per-pixel basis.

PARAMETERS

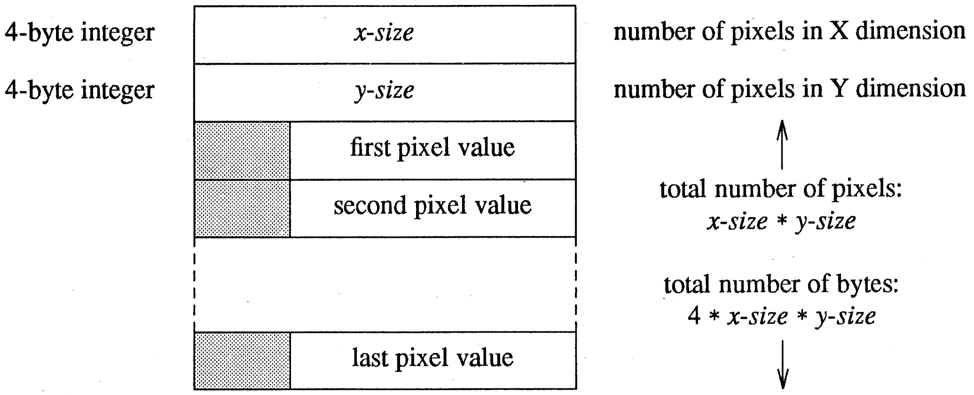
Read image A file browser window that allows you to specify the name of the image file to be read.

OUTPUTS

Data Field The output data is a 2D block of pixels. The data set at each point of the 2D field will be a 4-vector of bytes in the AVS *image* format.

IMAGE FILE FORMAT

read image expects its input file to be in the following format:



4-byte integer | *x-size* | number of pixels in X dimension

4-byte integer | *y-size* | number of pixels in Y dimension

first pixel value

second pixel value

last pixel value

total number of pixels: *x-size* * *y-size*

total number of bytes: 4 * *x-size* * *y-size*

RELATED MODULES

Image processing:

- contrast
- threshold
- histogram stretch
- clamp
- interpolate

Decompose/compose images from separate bands:

extract scalar
combine scalars

Display picture:

display image

Turn image data into a pixmap for more powerful viewing techniques:

image to pixmap
transform pixmap
display pixmap

read volume

read volume file from disk into a field

SUMMARY

Name	read volume	
Type	data	
Inputs	none	
Outputs	field 3D scalar byte	
Parameters	*Name*	*Type*
	Read Volume	Browser

DESCRIPTION

The **read volume** module reads a disk file in *volume data* format and outputs the data as a "field 3D scalar byte". It is used to read data files containing scalar-valued volume data (e.g. CAT scan data, NMR data).

PARAMETERS

read volume

A file browser allows you to specify the name of the file that contains the volume data set.

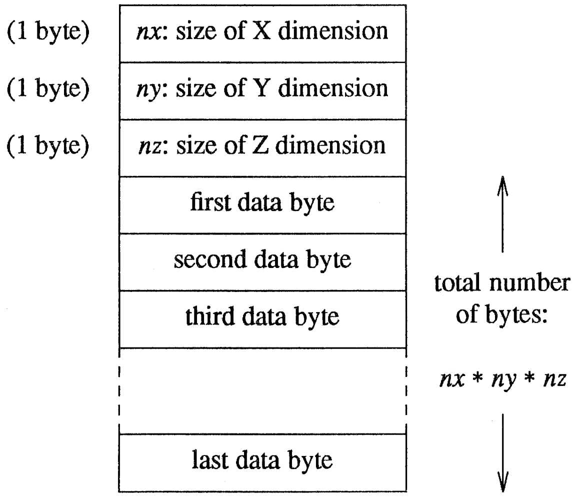
OUTPUTS

Data Field (field 3D scalar byte)

The output is the byte data cast as the scalar data in a 3D *field.*

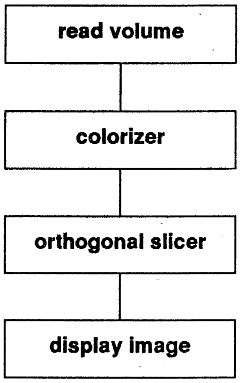
VOLUME DATA FILE FORMAT

read volume expects its input file to be in the following format:



EXAMPLE

This simple example displays a volume data set.



RELATED MODULES

Colormaps:

generate colormap, read colormap

Filters:

clamp, contrast, crop, downsize, field to byte, field to double,
field to float, field to int, histogram stretch, interpolate,
mirror, offset, transpose, colorizer, compute gradient, gradient shade

Mappers:

dot surface, arbitrary slicer, bubbleviz, orthogonal slicer,
field to mesh, isosurface, volume bounds

Renderers:

alpha blend, display image, render geometry, vbuffer

render geometry

convert geometric description to image (Geometry Viewer)

SUMMARY

Name	render geometry
Type	renderer
Inputs	geometry (optional, multiple) field 2D 4-vector byte (optional)
Outputs	pixmap
Parameters	*Name* add to object transform

DESCRIPTION

The **render geometry** 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 **render geometry**'s geometry input port. All the objects will be combined into a single scene. Each module providing input to **render geometry** 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 **render geometry** 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 **render geometry** 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**.

SPECIAL CONSIDERATIONS

This module is special: instead of having a few control widgets organized onto a single control panel page, its control panel is the entirely separate multi-level application menu of the Geometry Viewer subsystem. Thus, when you add the **geometry viewer** icon to a network, no page is added to the Network Control Panel. There are two ways to access the Geometry Viewer menu:

□ Click the small square in **render geometry** icon with the left mouse button.

□ Click the **Geometry Viewer** button located at the top of the Network Control Panel. This button is always visible, even when there is no active network.

In some circumstances, it is useful to be able to access both the Geometry Viewer control panel and the Network Control Panel simultaneously. They both occupy the same screen position, along the left edge of the screen. In these cases, use the X Window System window manager to move the one of these menu windows out of the way.

The **geometry viewer**'s control panel also differs from that of other modules in these ways:

□ The Network Editor's **Layout Editor** cannot be used to rearrange Geometry Viewer controls.

□ If a network includes more than one instance of **render geometry**, AVS does *not* create a separate control panel for each instance. Each **render geometry** sends its output to a different window, but the same Geometry Viewer application menu controls all the windows. The module whose output window is currently highlighted in red is the one being controlled. To switch the focus to another **render geometry** output window, just click in it with any mouse button.

INPUTS

Geometry (optional, multiple; geometry)

The input data can be any AVS *geometry*. More than one geometry can be input to this port. All the geometries will be combined into the same "scene".

Texture (optional; field 2D 4-vector byte)

The optional input provides a way to perform "dynamic texture mapping". The AVS *image* input to this port it is available as a dynamic texture. From within the "Edit Texture" submenu under **Objects**, you can bind this texture map to a particular object.

PARAMETERS

add to object transform

This parameter can be attached to the dialbox or the Spaceball, allowing these devices to control object transformations. In such cases, you can still control transformations using the mouse:

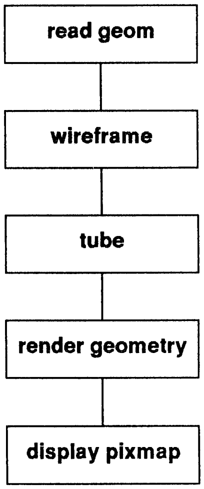
Mouse	**Transform**
middle	rotate
right	translate in plane of screen
middle with SHIFT key	scale
right with SHIFT key	translate perpendicular to plane of screen

OUTPUTS

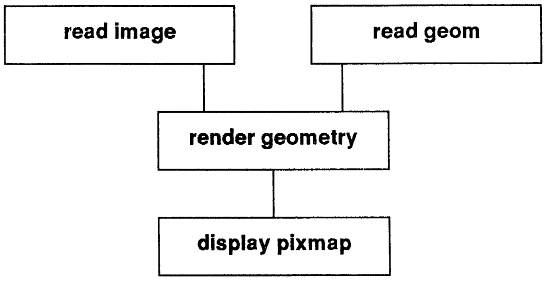
pixmap The output is a pixmap containing a scene that includes all the input objects.

EXAMPLE

This network creates a tube version of an object:



This network implements dynamic textures:



RELATED MODULES

display pixmap, read geom, pdb to geom, render manager

SEE ALSO The *Geometry Viewer* chapter.

render manager

share geometries among subnetworks

SUMMARY

Name	render manager	
Type	renderer	
Inputs	geometry	
Outputs	none	
Parameters	*Name*	*Type*
	Create New Window	one shot
	Active Windows	choice

DESCRIPTION

The **render manager** module takes geometries as input, uses the AVS Geometry Viewer to render them, and displays the results in one or more windows. This module is very similar to the **render geometry** module, with these differences:

- ❑ **render manager** creates its own pixmap and window on the screen, rather than relying on **display pixmap**. An initial window is created by default.
- ❑ **render manager** has a built-in mechanism for creating and selecting output windows. A set of windows is shared among **render manager** modules in separate subnetworks. At any moment, one of them — the *current output window* — is shared by all the **render manager** modules in all subnetworks. This window displays the combined results of all these modules.

It is possible to create a new output window, which automatically becomes the shared current output window. This provides a powerful capability for exploring differences between datasets, or different mappings of the same dataset. See the **Create New Window** parameter below.

This module is used by the AVS Image Viewer and Volume Viewer subsystems.

INPUTS

Geometry (geometry)

Any AVS *geometry.*

PARAMETERS

Create New Window

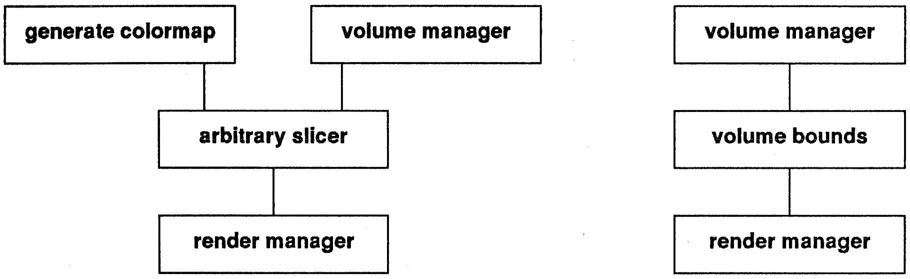
Click this button to create a new output window, which becomes the current output window. Subsequent geometric input is rendered into this window, until such time as you change the current output window again (perhaps by creating yet another window).

Active Windows

A choice menu that lists all the output windows, showing which one is current. You can also make an output window current by pressing any mouse button in the window itself.

EXAMPLE

Suppose you have built the following two networks:



When you select a volume dataset (e.g. *hydrogen.dat*) for the **arbitrary slicer** subnetwork, the slice is rendered by the Geometry Viewer, and a window is created to display the picture. If you select the same dataset in the **volume bounds** subnetwork, the bounds are rendered and displayed in the same window.

If you click **Create New Window**, and then select a new dataset was selected in the **arbitrary slicer** subnetwork, it (and it alone) is displayed in the new window. The geometries in the original window do not change.

RELATED MODULES

Same as for **render geometry**.

NOTES

The output window(s) are not destroyed until *all* **render manager** modules are destroyed.

scatter dots

generate spheres at points in 3D space

SUMMARY

Name	scatter dots
Type	mapper
Inputs	field 1D real 3-space irregular (a "scatter" field)
Outputs	geometry

Parameters

Name	*Type*	*Default*	*Min*	*Max*	*Choices*
Connect the dots	toggle	off			on, off
Radius	Real	0.01	0.0	1.0	

DESCRIPTION

The **scatter dots** module generates spheres of various radii at the coordinate locations in a specified field. For a scalar field, each sphere's radius is proportional to the scalar value, and the sphere is always colored white. If the field is a 4-vector float (such as that produced by the **bubbleviz** module), only the first element of the vector determines the sphere's radius. The other three elements are interpreted as red-green-blue color values (normalized to the range 0..1).

INPUTS

Point List (required; field 1D 3D float *irregular*) The input field must be a list of points in 3D space, with a *float* value specified at each point.

PARAMETERS

Connect the dots (toggle)

❑ **If OFF**, a sphere is drawn at each point in the field. The radius of the sphere is specified by the field element's scalar data value. (If the field has vector data, the value of the first vector element is used and the other values are discarded.)

❑ **If ON**, the points are represented as dots, connected with a single polyline (in the order specified by the 1D array). If the input field has 4-vector float data, the last three vector elements are interpreted as red-green-blue values, and the dots are assigned colors. No spheres are drawn.

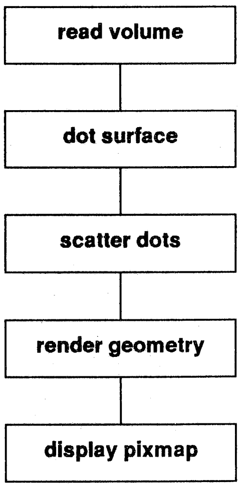
OUTPUTS

Geometry (geometry)

The output is an AVS *geometry*.

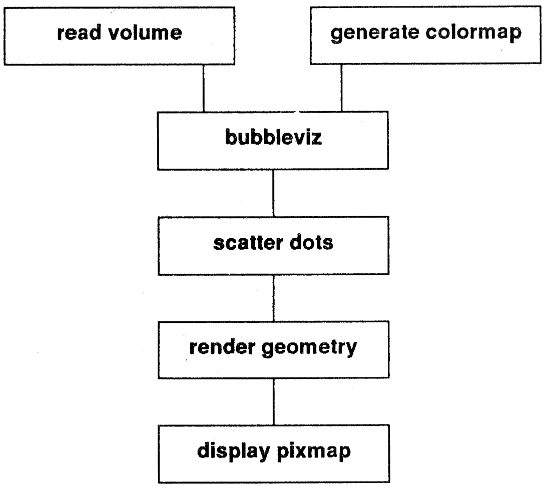
EXAMPLE 1

The **scatter dots** module can be used in combination with the **dot surface** module as follows:



EXAMPLE 2

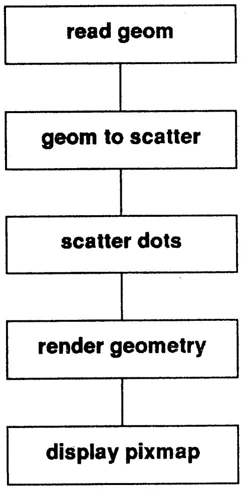
The **scatter dots** module is required to make **bubbleviz** work properly:



EXAMPLE 3

This example places "balls" at the vertices of a geometry:

scatter dots



RELATED MODULES

read geom, geom to scatter, tube, wireframe, render geometry

shrink

make an object smaller

SUMMARY

Name	shrink				
Type	filter				
Inputs	geometry				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	offset	float	0.0	none	none

DESCRIPTION

The **shrink** module transforms an AVS *geometry*, so that each vertex of each polygon is translated towards (or away from) the geometry's centroid (center of gravity). It is useful for visualizing the internal geometry of an object.

INPUTS

Geometry (required; geometry) An AVS geometry, created with the *libgeom* library or by another AVS module.

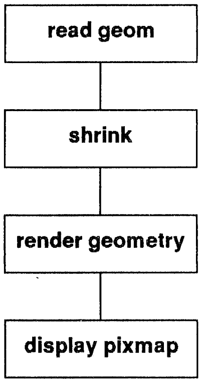
PARAMETERS

offset The amount by which each vertex is translated. Positive values collapse the geometry inward. Negative values create a "blow-up" of the geometry.

OUTPUTS

Geometry A geometry that represents the same object(s) as the input data.

EXAMPLE



RELATED MODULES

read geom, flip normal, tube, render geometry

LIMITATIONS

This module works only for polytriangle strips and meshes; it does not work for polyhedra.

This module doesn't copy UV data.

This module can increase the size of the data: it can generate up to five times the number of triangles for polytriangle objects, and up to three times the number of vertices for meshes.

stream lines

generate stream lines for a vector field

SUMMARY

Name	stream lines				
Type	mapper				
Inputs	field 3D 3-vector float				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	width	integer	12	4	32
	length	integer	12	4	128
	step	float	0.02	0.0	1.0
	position	trackball			

DESCRIPTION

The **stream lines** module generates streamlines based on a *field* that is a volume of 3D vectors. It places a straight line — a set of collinear points — at a starting location in the volume. (Both the location of the number of points are parameter-controlled.) Then, for every time step, it advances each point through space based on the interpolated value of the vector field at its present position. The result is a set stream lines showing the progress of massless particles through a vector field.

This module is similar to the **particle advector** module, except that (1) its source points define a line instead of a square region, and (2) the result is a static set of lines instead of a dynamically updated set of spheres.

This module is also similar to **stream mesh**, except that it produces lines instead of a polygonal mesh.

INPUTS

Data Field (required; field 3D 3-vector float)

The input field must be 3D, and the data for each field element must be a 3D vector of floats, representing the components of a velocity vector.

PARAMETERS

Width The number of points in the line placed within the field. This number of streamlines will result.

Length A scale factor, which multiplies the length of the streamline segments generated during each time step.

Step Determines the time step for the interactive computation. The larger the value, the greater the interval.

Position Determines the initial orientation of the line of points within the volume. The mouse controls the position using the same "virtual trackball" paradigm used by the Geometry Viewer (**render geometry** module):

Mouse	**Transform**
middle	rotate
right	translate in plane of screen
middle with SHIFT key	scale
right with SHIFT key	translate perpendicular to plane of screen

OUTPUTS

Streamlines (geometry)

A set of disjoint lines.

RELATED MODULES

hedgehog, particle advector, stream mesh

stream mesh

generate stream lines for a vector field as a polygonal mesh

SUMMARY

Name stream mesh

Type mapper

Inputs field 3D 3-vector float

Outputs geometry

Parameters

Name	*Type*	*Default*	*Min*	*Max*
width	integer	12	4	32
length	integer	12	4	128
step	float	0.02	0.0	1.0
position	trackball			

DESCRIPTION

The **stream mesh** module generates streamlines based on a *field* that is a volume of 3D vectors. It connects the streamlines into a geometric primitive called a *polygonal mesh*, which is optimized for fast rendering.

This module is essentially similar to **stream lines**, except that the output is a polygonal mesh rather than a set of disjoint lines.

threshold

restrict values in data field

SUMMARY

Name	threshold				
Type	filter				
Inputs	field *any-dimension any-data*				
Outputs	field of same type as input				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	thresh_min	float	0.0	none	none
	thresh_max	float	255.0	none	none

DESCRIPTION

The **threshold** module transforms the values of a field as follows:

- ❑ Any value less than the value of the **threshold_min** parameter is set to 0.
- ❑ Any value greater than the value of the **threshold_max** parameter is set to 0.
- ❑ All values within the **threshold_min**-to-**threshold_max** range are not changed.

After being **threshold**'ed, a data set's values are all in this range:

threshold_min ≤ *value* ≤ **clamp_max**

Note the difference between the **clamp** and **threshold** modules:

- ❑ **threshold** sets values outside the specified range to be zero.
- ❑ **clamp** sets values outside the specified range to be the range's minimum and maximum values.

INPUTS

Data Field (required; field *any-dimension any-data*)
The input data may be any AVS field.

PARAMETERS

thresh_min The minimum threshold value.

thresh_max The maximum threshold value.

OUTPUTS

Field Data The output field has the same dimensionality as the input field, but the number of elements in each dimension is reduced.

RELATED MODULES

Modules that could provide the **Data Field** input:

read volume
any other filter module

Modules that could be used in place of **threshold**:

clamp

Modules that can process **threshold** output:

colorizer
any other filter module

transform pixmap

perform 3D transformation on pixmap

SUMMARY

Name	transform pixmap	
Type	renderer	
Inputs	pixmap	
Outputs	pixmap	
Parameters	*Name*	*Type*
	image transform	4x4 matrix
	transform image	toggle
	reset	toggle
	refine	toggle

DESCRIPTION

The **transform pixmap** module maps its pixmap input onto a rectangle that has been arbitrarily transformed in three dimensions. The resulting pixmap is then output. The transformation allows for rotation, scaling, translation, or shearing of the image (or any combination thereof).

A benefit of using **transform pixmap** in a network is that it automatically scales its output pixmap size to fit the output window of a **display pixmap** module downstream. For example, if you read in a 512x512 pixmap, you can display the entire pixmap in any size window.

INPUTS

pixmap The input can be any AVS *pixmap*.

PARAMETERS

image transform

Controls the 3D transform to be applied to the pixmap. The control widget is a window containing a colored cube, annotated with coordinate axis information. Transforming this cube with the following mouse buttons causes the pixmap to be transformed accordingly:

Mouse Transform	
left	cycle among three views: along X-axis, along Y-axis, along Z-axis
middle	rotate
right	translate in plane of screen
middle with SHIFT key	scale
right with SHIFT key	translate perpendicular to plane of screen

The mouse button mapping is the same as in the Geometry Viewer (or the **render geometry** module).

transform image (toggle)

This toggle parameter controls whether you can transform the image directly (i.e. in its window), or must use the **transformation** widget described above.

- ❑ **If ON:** The **transform pixmap** module "grabs" button press events in the associated output window, allowing you to transform the image directly..eS NOTE: For pixmaps generated by a **render geometry** module, button clicks in the window will no longer transform the geometry, but will transform the image instead.
- ❑ **If OFF:** The mouse buttons have the same meanings, but you cannot "grab" the image in the output window directly. Instead, you must transform the cube in the transform control widget, which appears in the module's control panel.

refine (toggle)

Controls the use of point sampling to improve the quality of the output pixmap.

- ❑ **If ON**, A "successive refinement" algorithm is used to improve picture quality. When there is no other work left to do, **transform pixmap** applies nine refinement passes, each of which incrementally improves the picture. This is especially useful when small images are to be displayed in very large windows, or vice-versa.
- ❑ **If OFF**, the transformation applied to the image uses a "point sampling" algorithm.

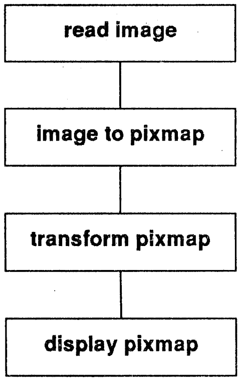
reset (one-shot)

Resets the transformation of the image to be the identity transformation.

OUTPUTS

pixmap The output is a pixmap containing a scene that includes all the input objects.

EXAMPLE



RELATED MODULES

image to pixmap, transform pixmap, display pixmap

LIMITATIONS

This module does not work properly on a 16-plane system. It simply passes pixmaps through unchanged.

When you transform an image directly (**transform image** toggle) or use the **Reset** function, the transform control widget is not updated.

transpose

exchange dimensions in a 2D or 3D data set

SUMMARY

Name	transpose			
Type	filter			
Inputs	field 2D/3D *any-data*			
Outputs	field of same type as input			
Parameters	*Name*	*Type*	*Default*	*Choices*
	axis	choice	Original	Original, YZ, XZ, XY

DESCRIPTION

The **transpose** module exchanges the data in two dimensions of a 2D or 3D field. It can be used to change the orientation of the data for display and/or processing purposes.

INPUTS

Data Field (required; field *any-dimension any-data*)
The input data may be any AVS field.

PARAMETERS

axis The choices for exchanging the data are:

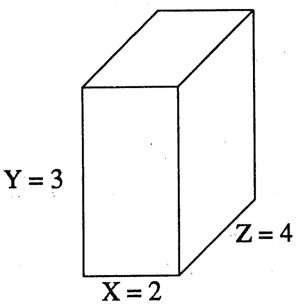
Original	Copies the input to the output; no transformation is performed.
YZ	Swaps the Y and Z dimensions. (Equivalent to "Original" for a 2D field.)
XZ	Swaps the X and Z dimensions. (Equivalent to "Original" for a 2D field.)
XY	Swaps the X and Y dimensions.

OUTPUTS

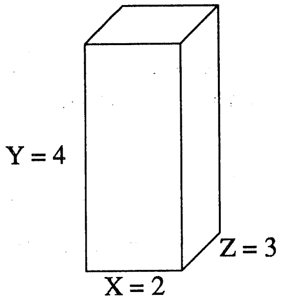
Data Field (field *any-dimension any-data*)
The output field has the same dimensionality and type as the input field.

EXAMPLES

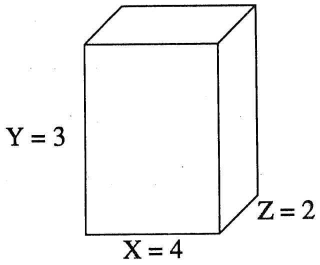
These drawings illustrate the transposition choices:



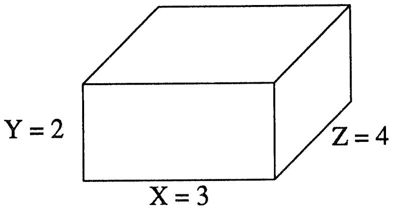
Original Data



After YZ Transpose



After XZ Transpose



After XY Transpose

RELATED MODULES

This module combined with **mirror** can re-orient the data in any desired way.

tube

convert lines to cylindrical tubes

SUMMARY

Name	tube				
Type	filter				
Inputs	geometry				
Outputs	geometry				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	radius	float	0.0	none	none

DESCRIPTION

The **tube** module transforms an AVS *geometry*, replacing a set of disjoint lines with "tubes" constructed out of eight polygons.

INPUTS

Geometry (required; geometry) An AVS geometry, created with the *libgeom* library or by another AVS module.

PARAMETERS

radius The radius to be used for the tube. Only values in the range 0.0 – 0.4 produce an acceptable result.

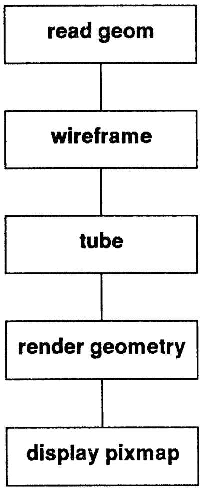
OUTPUTS

Geometry (geometry)

The output is an AVS *geometry*, representing each input line as a set of polygons.

EXAMPLE

In this example, the original geometry includes no disjoint lines. The **wireframe** module is used to add disjoint lines, which are then converted to tubes.



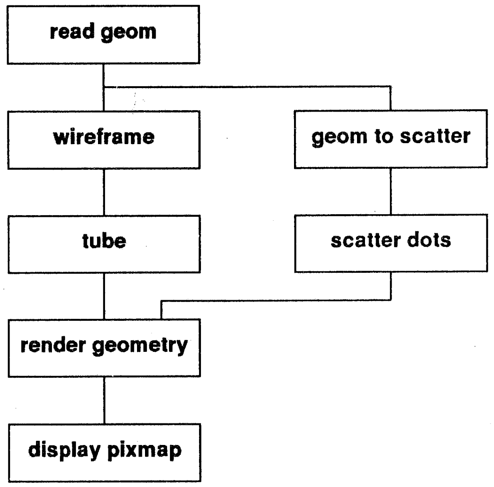
RELATED MODULES

read geom, offset, shrink, flip normal, wireframe, render geometry

LIMITATIONS

Only **radius** values in the range 0.0 – 0.4 produce acceptable results.

The cylinders are not capped and adjacent line segments are not joined. For thick cylinders, there may be quite a bit of surface intersections at the joins. This can be somewhat mitigated by modifying the example network above as follows:



Be sure to set the **scatter dots** radius to be the same as the **tube** radius. This network places a ball of the same diameter as the tube at each of the corners where the lines join.

vbuffer

perform volumetric rendering on volume data

SUMMARY

Name vbuffer

Type renderer

Inputs field 3D scalar (uniform)
colormap

Outputs pixmap

Parameters

Name	*Type*	*Default*	*Min*	*Max*
Xrotation	real	0.0	-180	180
Yrotation	real	0.0	-180	180
Zrotation	real	0.0	-180	180
ScaleX	real	0.5	0.0	5.0
ScaleY	real	0.5	0.0	5.0
ScaleZ	real	0.5	0.0	5.0
Ztrans	real	-2.0	-10	10
Imagesize	int	200	10	1024
Background	real	0.0	0.0	1.0
Cell-Based	logical	off	off	on
Re-use Scripts	logical	on	off	on
Script File	string	/tmp/vbuf.inp		
Scalar File	string	/tmp/vbuf.dat		
Image File	string	/tmp/vbuf.image		

DESCRIPTION

The **vbuffer** module creates a volumetric rendering of a 3D uniform scalar field, using RGB color and opacity transfer functions. The technique employed uses two levels of interpolation:

- ❑ A cell-averaged or voxel representation produces lower quality and lower apparent resolution images at moderate execution speeds.
- ❑ A trilinear or cell-based interpolation results in very high quality images with a slower execution time.

The **vbuffer** module is actually a wrapper for a standalone program, **vbuf**, which performs the image computation:

1. **vbuffer** places the parameters and transfer functions in a disk file for **vbuf** to read.
2. **vbuffer** executes **vbuf** and waits for it to exit.
3. **vbuf** reads the data from the disk file, computes an image, writes it to disk, and exits.
4. **vbuffer** reads the image that was written to disk and places it on the output port.

You can choose between two rendering algorithms:

- ❑ **Trilinear interpolation**: This algorithm uses an inverse-mapping scheme to generate images. The 3D field data is decomposed into 8-node *cells*, with one field value at each vertex of the cell. Each cell is processed by accumulating color and opacity along an integration volume which maps into one or more pixels.

  At several locations through this volume, both the value of the scalar field and its gradient are interpolated. The sampled scalar field value is used to map into transfer functions, which determine the color and opacity at the point. The color for the pixel is determined by a product of the diffuse illumination (due to the dot product of the light source and gradient vectors), the opacity, and the sampled color as accumulated along the volume. After all the pixels that the cell projects into are processed, the algorithm moves on to the next cell. Partial pixel contributions are saved into an in-memory frame buffer.

- ❑ **Voxel Approximation** (default): The 3D field is decomposed into cells, as described above. But no interpolation is performed within the cell. Each cell has a single opacity, color, texture color, surface gradient, and set of shading parameters. This method is much faster than the trilinear interpolation method. Use it to get a quick (albeit ragged) look at the data. It is most useful for selecting the opacity and color transfer functions.

INPUTS

Data Field (required; field 3D scalar uniform)
The input field must be 3-dimensional. The data for each field element must be a scalar.

Colormap (required; colormap)
Any AVS *colormap*.

PARAMETERS

Xrotation The x rotation of the data field in degrees.

Yrotation The y rotation of the data field in degrees.

Zrotation The z rotation of the data field in degrees.

ScaleX The x scale factor of the data field. By default the highest resolution axis of the data is scaled between -1.0 and 1.0, and other axes are scaled accordingly by this.

ScaleY The y scale factor of the data field.

ScaleZ The z scale factor of the data field.

Ztrans The z translation of the field. The coordinate system is right handed, so only that data which has a negative z coordinate will be visible.

Imagesize The resolution in pixels of the computed image. The image is always square. The algorithm complexity scales with the number of pixels and the number of cells which have a non-zero opacity.

Background
The background brightness. Zero corresponds to a black background, 1.0 to a white background.

Cell-Based A toggle switch between the voxel approximation algorithm (default) and the trilinear interpolation algorithm (cell-based and substantially slower).

Re-use Scripts
vbuffer generates a script to run **vbuf.** This toggle switch allows you either to reuse these scripts, data files and images, or to build new scripts and files with each new invocation.

Script File The name of the vbuf script file.

Scalar File The name of the vbuf data input file.

Image File The name of the vbuf output image file.

OUTPUTS

Pixmap (pixmap)

An AVS pixmap containing the 2D image rendered by **vbuffer**.

RUNNING VBUF AS A STANDALONE PROGRAM

vbuf itself can be run as a process separate from AVS, using the input files and datasets generated by **vbuffer**. **vbuf** recognizes these command-line options:

–v1 Send a message to *stdout* each time a new plane of the data is computed.

–v2 As each new cell is processed, lists the *ijk* value. (This is extremely verbose.)

–voxel *number*

When run standalone, the default rendering method uses a trilinear interpolation algorithm. This options invokes the voxel approximation algorithm instead.

The standard command line for running **vbuf** as a standalone program is:

vbuf –v1 < *input_file*

The *input_file* can be the script produced by the **vbuffer** module. Input files contain a small set of commands:

load *datafile* **bin res** *ni nj nk* **aspect** *dx dy dz* [**grid** *gridfile*]
load *datafile* **ascii res** *ni nj nk* **aspect** *dx dy dz* [**grid** *gridfile*]

Data description commands. The optional grid file contains the coordinate values for rectilinear fields. This ASCII file contains all the X coordinates (one per line), followed by all the Y coordinates, then all the Z coordinates.

image *imagefile* **res** *ires jres* **bg** *red green blue* **encode frame** *num*

Image description command.

rotate *rx ry rz* **trans** *tx ty tz* **fov** *angle*

Frame composition command.

color dcue *znear zfar exponent* **ambient** *redA greenA blueA*
scalar(1) red(scalar(1)) green(scalar(1)) blue(scalar(1))
...
scalar(n) red(scalar(n)) green(scalar(n)) blue(scalar(n))
<blank line>

Object color description and transfer function command. Note that this is a multi-line command whose last line is blank. The blank line terminates the command.

opacity
scalar(1) opacity(scalar(1))
...
scalar(n) opacity(scalar(n))
<blank line>

Opacity transfer function command. Note that this is a multi-line command whose last line is blank. The blank line terminates the command.

shade light *xl yl zl xd yd zd falloff*
scalar(1) shade(scalar(1))
...
scalar(n) shade(scalar(n))

Illumination command. The light points from *(xl, yl, zl)* to *(xd, yd, zd)*. Note that this is a multi-line command whose last line is blank. The blank line terminates the command.

map *mapfilename* **res** *ires jres kres*
3D texture mapping command.

view Image creation command.

LIMITATIONS

Only uniform fields can be rendered by **vbuffer.**

Some image anomalies can occur along cell boundaries. This is view-orientation dependent.

The execution time can be dramatically reduced by limiting the number of cells that contain a non-zero amount of opacity.

RELATED MODULES

Modules that could provide the Data Field input: read volume

Modules that could be used in place of **vbuffer**:

isosurface
dot surface
alpha blend

Modules that can process **vbuffer** output:

transform pixmap
display pixmap
write pixmap

SEE ALSO

"Vbuffer: Visible Volume Rendering," C. Upson, M. Keeler, *Computer Graphics*, V 22, N 4, August 1988, pp 59-65.

vector curl

compute the curl of a vector field

SUMMARY

Name	vector curl
Type	filter
Inputs	field 3D 3-vector float
Outputs	field 3D 3-vector float
Parameters	none

DESCRIPTION

The **vector curl** module accepts a vector field as input and computes the curl of that field as output. This is related to the divergence as follows:

$curl = (DEL \times F)$
$div = (DEL \bullet F)$

... where F is the vector input field.

The equation used to compute the curl is:

```
new_dx[X][Y][Z] = (dz[X][Y+1][Z] - dz[X][Y-1][Z]) -
                      (dy[X][Y][Z+1] - dy[X][Y][Z-1])

new_dy[X][Y][Z] = (dx[X][Y][Z+1] - dx[X][Y][Z-1]) -
                      (dz[X+1][Y][Z] - dz[X-1][Y][Z])

new_dz[X][Y][Z] = (dy[X+1][Y][Z] - dy[X-1][Y][Z]) -
                      (dx[X][Y+1][Z] - dx[X][Y-1][Z])
```

INPUTS

Data Field (required; field 3D 3-vector float)

The input field must represent a volume of elements, with a 3D vector of floating-point data values for each element.

OUTPUTS

Data Field (field 3D 3-vector float)

The output field is in the same format as the input field.

RELATED MODULES

vector grad, vector div, vector norm, vector mag, hedgehog, particle advector, gradient shade, stream lines, stream mesh

LIMITATIONS

This module works only with 3D 3-vector float fields. This data type is widely used in flow analysis, where each 3-vector of floats represents the components of a velocity or a gradient.

vector div

compute the divergence of a vector field

SUMMARY

Name	vector div
Type	filter
Inputs	field 3D 3-vector float
Outputs	field 3D scalar float
Parameters	none

DESCRIPTION

The **vector div** module accepts a vector field as input and computes the divergence of that field as output. This is related to the curl as follows:

curl = (*DEL* × *F*)
div = (*DEL* • *F*)

... where F is the vector input field.

The equation used to compute the divergence is:

```
divergence[X][Y][Z] = (dx[X+1][Y][Z] - dz[X-1][Y][Z]) +
                      (dy[X][Y+1][Z] - dy[X][Y-1][Z]) +
                      (dz[X][Y][Z+1] - dz[X][Y][Z-1])
```

INPUTS

Data Field (required; field 3D 3-vector float)

The input field must represent a volume of elements, with a 3D vector of floating-point data values for each element.

OUTPUTS

Data Field (field 3D scalar float)

The output field has a single floating-point value for each input field element.

RELATED MODULES

vector curl, vector div, vector norm, vector mag, hedgehog, particle advector, gradient shade, stream lines, stream mesh

LIMITATIONS

This module works only with 3D 3-vector float fields. This data type is widely used in flow analysis, where each 3-vector of floats represents the components of a velocity or a gradient.

vector grad

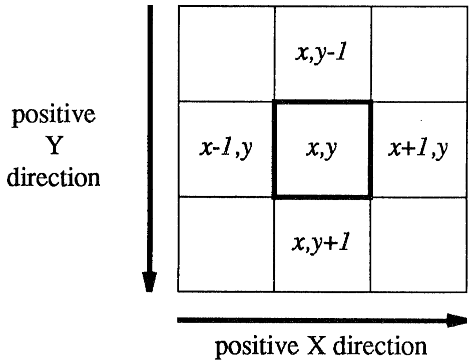
compute the vector gradient of a scalar field

SUMMARY

Name	vector grad
Type	filter
Inputs	field 3D scalar float
Outputs	field 3D 3-vector float
Parameters	none

DESCRIPTION

The **vector grad** module computes the gradient of a 2D or 3D field. The gradient is treated by some other modules as a "pseudo-normal" to the "surface" for each data element. A "nearest neighbor" algorithm is used to compute the gradient: the difference between the next data value (in each direction) and the previous data value. In two dimensions, this can be represented as follows:



```
Delta_x[X][Y][Z] = data[X+1][Y][Z] - data[X-1][Y][Z]
Delta_y[X][Y][Z] = data[X][Y+1][Z] - data[X][Y-1][Z]
Delta_z[X][Y][Z] = data[X][Y][Z+1] - data[X][Y][Z-1]
```

This module is identical to the **compute gradient** module, except that it does *not* normalize the output. **compute gradient** is designed for gradient shading fields, whereas this module is designed for input into the other vector field modules: **vector curl**, **vector div**, **vector mag**, and **vector norm**. Note that **vector grad** followed by **vector norm** produces the same results as **compute gradient.**

RELATED MODULES

vector curl, vector div, vector norm, vector mag, hedgehog, particle advector, stream lines, stream mesh

LIMITATIONS

There may be algorithms better than "nearest-neighbor" for computing the gradient.

This module produces 12 bytes per pixel (voxel). For example, a 128 x 128 x 128 byte volume is about 2.1 MB before the gradient is computed. The **compute gradient** module produces a 25.2 MB internal data set from this data. This will have an adverse performance effect on systems whose physical memory is 32 MB or less.

This module works only with 3D 3-vector float fields. This data type is widely used in flow analysis, where each 3-vector of floats represents the components of a velocity or a gradient.

vector mag

compute the magnitude of a vector field

SUMMARY

Name	vector mag
Type	filter
Inputs	field 3D 3-vector float
Outputs	field 3D scalar float
Parameters	none

DESCRIPTION

The **vector mag** module accepts a vector field as input and computes the magnitude of each vector data value. The output is a scalar field consisting of the magnitudes.

The magnitude equation is:

```
    Magnitude[X][Y][Z] = sqrt((dx[X][Y][Z]*dx[X][Y][Z]) +
                              (dy[X][Y][Z]*dy[X][Y][Z]) +
                              (dz[X][Y][Z]*dz[X][Y][Z]) )
```

INPUTS

Data Field (required; field 3D 3-vector float)

The input field must represent a volume of elements, with a 3D vector of floating-point data values for each element.

OUTPUTS

Data Field (field 3D scalar float)

The output field has a single floating-point value for each input field element.

RELATED MODULES

vector curl, vector div, vector norm, vector mag. hedgehog, particle advector, gradient shade, stream lines, stream mesh

LIMITATIONS

This module works only with 3D 3-vector float fields. This data type is widely used in flow analysis, where each 3-vector of floats represents the components of a velocity or a gradient.

vector norm

normalize a vector field

SUMMARY

Name	vector norm
Type	filter
Inputs	field 3D 3-vector float
Outputs	field 3D 3-vector float
Parameters	none

DESCRIPTION

The **vector norm** module accepts a vector field as input, and produces a normalized version of that vector field as output. The normalization equation looks like:

```
Magnitude = sqrt((dx*dx) +  (dy*dy) + (dz*dz))
New_dx    = dx / Magnitude
New_dy      = dy / Magnitude
New_dz      = dz / Magnitude
```

INPUTS

Data Field (required; field 3D 3-vector float)

The input field must represent a volume of elements, with a 3D vector of floating-point data values for each element.

OUTPUTS

Data Field (field 3D 3-vector float)

The output field is in the same format as the input field.

RELATED MODULES

vector curl, vector div, vector norm, vector mag, hedgehog, particle advector, gradient shade, stream lines, stream mesh

LIMITATIONS

This module works only with 3D 3-vector float fields. This data type is widely used in flow analysis, where each 3-vector of floats represents the components of a velocity or a gradient.

volume bounds

generate bounding box of 3D 3-vector field

SUMMARY

Name	volume bounds
Type	mapper
Inputs	field 3D *any-data*
Outputs	geometry

Parameters

Name	*Type*
Bounding Box	toggle
Hull	toggle
Min I	toggle
Max I	toggle
Min J	toggle
Max J	toggle
Min K	toggle
Max K	toggle

DESCRIPTION

The **volume bounds** module generates lines that indicate the "bounding box" of a 3D data set (field). It is frequently used in conjunction with other geometry-based volume-visualization modules (e.g. **bubbleviz**, **isosurface**, **hedgehog**, **arbitrary slicer**), since it provides some volumetric context for the data.

INPUTS

Data Field (required; field 3D *any-data*) The input data must be a 3D field, but may have any kind of data at each location in the field.

OUTPUTS

Bounds (geometry) The output *geometry* consists of the lines that form the bounding box. This box can take several forms, as specified by the **Type** parameter.

PARAMETERS

Bounding Box

If **ON**, the edges of a rectangular solid are drawn, showing the extent of the data in Cartesian space.

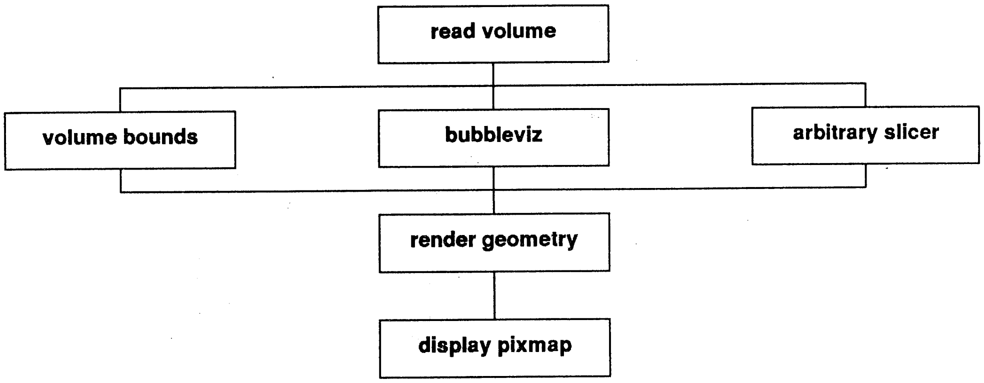
Hull (for rectilinear and irregular input fields only) If **ON**, draws the edges of the bounding box (see above), but deforms these edges in accordance with the coordinate data supplied in the input field. That is, **volume_bounds** draws the bounding box in the *physical space*, rather than the *computational space*.

Min I
Max I
Min J
Max J
Min K
Max K

These toggle switches provide further help is visualizing the way the computational space is mapped into physical space. Each one fills in one of the six faces of the hull. For example, turning on **Min I** draws a mesh showing the 2D slice of field elements with the minimum index value in the first dimension; turning on **Max K** draws a mesh showing the 2D slice of field elements with the maximum index value in the third dimension.

EXAMPLE

The following network showing a typical usage of **volume bounds**:



RELATED MODULES

read volume, volume manager

volume manager

share volumes among subnetworks

SUMMARY

Name	volume manager		
Type	data		
Inputs	none		
Outputs	field 3D scalar byte		
Parameters	*Name*	*Type*	*Choices*
	VOLUMGR select	choice	Select, Replace
	Volume Manager	browser	
	Volume Choices	choice	

DESCRIPTION

The **volume manager** module reads an volume file from disk and outputs the volume as a "field 3D scalar byte". It works like the **read volume** module, except that it has both a caching mechanism and a way of sharing data among **volume manager** modules in separate subnetworks.

See the **read volume** manual page for a description of the volume format.

PARAMETERS

VOLUMGR Select

A choice that determines how newly-read volumes will be placed to the list of currently active volumes:

- ❑ If **Select** is chosen, a new volume is added to the end of the list.
- ❑ If **Replace** is chosen, a new volume replaces the currently selected member on this list.

In either case, the change is reflected in *all* the *volume manager* modules in all active subnetworks.

Volume Manager

A file browser that allows you to select an volume file to read.

Volume Choices

A set of choices, listing each of the currently active volumes.

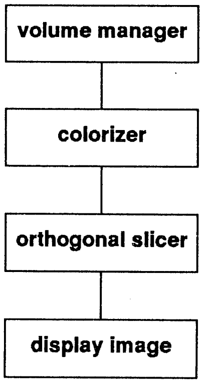
OUTPUT

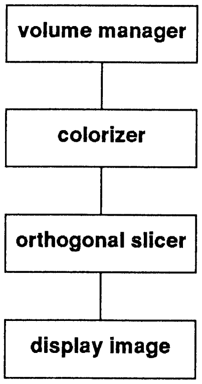
Data Field (field 3D scalar byte)

The output is the byte data cast as the scalar data in a 3D *field.*

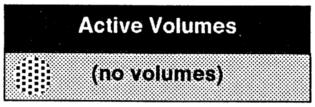
EXAMPLE

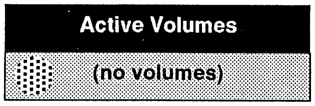
The following subnetworks might be used to display two volumes:



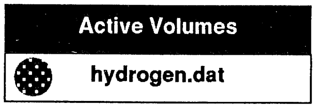


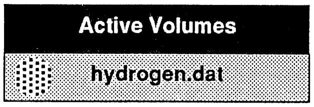
In this case, both **volume manager** modules would contain "select/replace" choice buttons, a file browser, and an area below the browser:



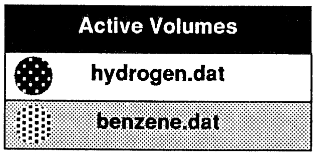


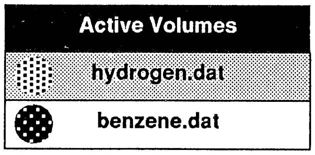
Once a volume (e.g. *hydrogen.dat*) was selected from the browser in the **volume manager** on the left, these buttons would look like this:



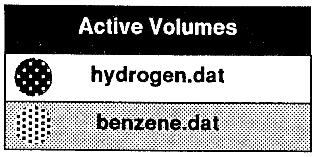


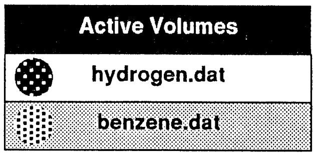
If a different file (e.g. *benzene.dat*) is chosen from the browser in the **volume manager** on the right, the buttons would look like this:





By selecting the same active volume, you can have both networks display the same volume:





Now, if you want to replace this volume with a new one, click on the **Replace** buttons above the browser, then select a new file (e.g. *methane.dat*) in just one of the **volume manager** browsers. The result is that all **volume manager** modules with the old volume (*hydrogen*) selected as their active volume will be automatically updated with the new volume (*methane.dat*).

RELATED MODULES

Same as for **read volume.**

LIMITATIONS

The cached volumes are not freed until all **volume manager** modules are destroyed. Because volume data can be large, caching multiple volume datasets can use up a lot of memory.

wireframe

convert object from surface to wireframe representation

SUMMARY

Name	wireframe
Type	filter
Inputs	geometry
Outputs	geometry
Parameters	none

DESCRIPTION

The **wireframe** module transforms an AVS *geometry*, replacing all surfaces defined as polytriangle strips with wireframe representations. This is useful for constructing a wireframe version of an object that has been defined as a shaded surface.

INPUTS

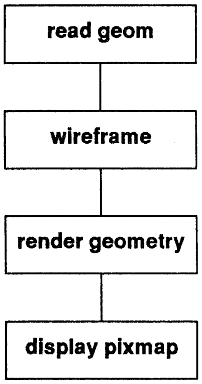
Geometry (required; geometry) Any AVS *geometry*, created with the *libgeom* library or produced by another AVS module.

OUTPUTS

Geometry A geometry that represents the same object as the input data.

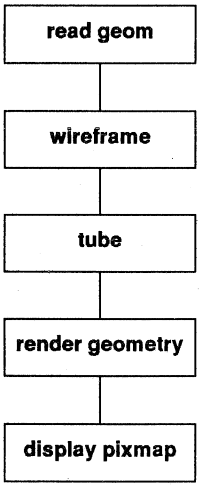
EXAMPLE

This example shows the use of the **wireframe** module to generate a wireframe version of a polygonal object:



EXAMPLE 2

This example uses the **wireframe** and **tube** modules to have a geometry involving spheres drawn with cylinders instead of lines:



RELATED MODULES

read geom, offset, shrink, flip normal, tube, render geometry

LIMITATIONS

The **wireframe** module generates lines based on the order of the vertices of a polytriangle strip. Sometimes, the resulting object is not exactly what you want. It may have "cobwebs" and other (usually invisible) data inconsistencies of the original polytriangle strip. You may need to regenerate the original data in order to produce the desired wireframe representation.

write field

write a field description to disk

SUMMARY

Name	write field				
Type	renderer				
Inputs	field				
Outputs	none				
Parameters	*Name*	*Type*	*Default*	*Min*	*Max*
	Write Field	browser			

DESCRIPTION

The **write field** module writes an AVS *field* description to disk. The field format on disk includes two parts, an *ASCII header* and a *binary area.* This format is described in detail in the manual page for **read field.**

INPUTS

Data Field The input can be an AVS *field.*

PARAMETERS

Write Field A file browser that allows you to specify the name of the field file to be created. The file suffix *.fld* is appended to the name automatically. If the file already exists, **write_field** issues a warning message and has you confirm the operation ("Overwrite") or cancel it ("Cancel").

After the field file is written, the filename is reset to NULL. This prevents subsequent changes upstream in the network from automatically triggering the rewriting of the file. A new file is written only when you enter a filename.

EXAMPLE

Following is an example of a file produced by **write field**:

```
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)

262,144 bytes of data
```

The field has three dimensions (it's a volume), 64x64x64. There is a single byte at each point, and the field is uniform.

RELATED MODULES

read field reads *any* AVS field file.

AVS data input modules that produce field output:

image manager
read 3-vector data
read image
read volume
volume manager

AVS filters that produce field output:

clamp
colorizer
combine scalars
compute gradient

geom to scatter
gradient shade
histogram stretch
interpolate

- contrast
- crop
- dot surface
- downsize
- extract scalar
- field to byte
- field to double
- field to float
- field to int
- mirror
- threshold
- transpose
- vector curl
- vector div
- vector grad
- vector mag
- vector norm

AVS mappers that produce field output:

- bubbleviz
- orthogonal slicer
- pixmap to image

ERRORS

write field will complain if it can't open the file, or if there isn't enough space to write the complete file.

write image

store image data in a file

SUMMARY

Name	write image
Type	renderer
Inputs	field 2D 4-vector byte
Outputs	none
Parameters	*Name* *Type* Write Image browser

DESCRIPTION

The **write image** module writes an AVS *image* data structure to a file. This structure takes the form of a "field 2D 4-vector byte". See the **read image** manual page for a detailed description of the image format.

INPUTS

Data Field (required; field 2D 4-vector byte)

The input can be any AVS *image*.

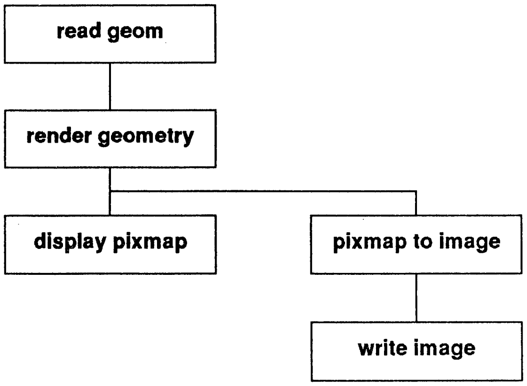
PARAMETERS

Write image

A file browser that allows you to specify the name of the image file to be created. The file suffix *.x* is appended to the name automatically. If the file already exists, **write_image** issues a warning message and has you confirm the operation ("Overwrite") or cancel it ("Cancel").

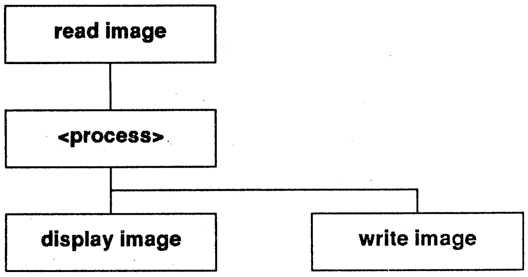
After the image file is written, the filename is reset to NULL. This prevents subsequent changes upstream in the network from automatically triggering the rewriting of the file. A new file is written only when you enter a filename.

EXAMPLE



AVS IMAGE VIEWER

All of the data pre-processing networks in the AVS Image Viewer have the following structure:



... where <process> is one of the following modules **crop**, **downsize**, **mirror**, **transpose**, or **interpolate**. These networks are in the *image_viewer* subdirectory of the */usr/avs/networks* directory.

RELATED MODULES

Image processing:

contrast, threshold, histogram stretch, clamp, interpolate

Decompose/compose images from separate bands:

extract scalar, combine scalars

Show image:

display image

Take output from renderer, and write the data out as an image:

render geometry, transform pixmap, pixmap to image

write volume

write volume data to a file

SUMMARY

Name	write volume	
Type	renderer	
Inputs	field 3D scalar byte	
Outputs	none	
Parameters	*Name*	*Type*
	Write Volume	browser

DESCRIPTION

The **write volume** module writes volume data to a file. The volume is in the AVS format "field 3D scalar byte". The data format on disk is:

1 byte: number of voxels in X
1 byte: number of voxels in Y
1 byte: number of voxels in Z
nx * ny * nz * 1 byte: voxel data

Each time the file is written, the filename is reset to NULL. This prevents successive changes upstream in the network to automatically trigger a volume data file to be written. A new filename must be entered each time the file is to be written out.

If the file to be written exists, the following warning appears:

```
File FILENAME
   already exists.  Do you want to overwrite it?
```

Two choices are presented. If you select **Cancel**, the write operation is aborted. If you select **Overwrite**, the existing file on disk is replaced with the new volume data.

This module is commonly used to pre-process a volume database for later use. For example, the input data might be very low-contrast. You could construct a network that includes the **contrast** module and the **write volume** module. Once you select appropriate settings for the contrast, the data could be written to a file, and used later for other types of processing.

INPUTS

Data Field (required; field 3D scalar byte)

The input data must be a 3D field, with a byte value at each location in the field.

PARAMETERS

Write Volume

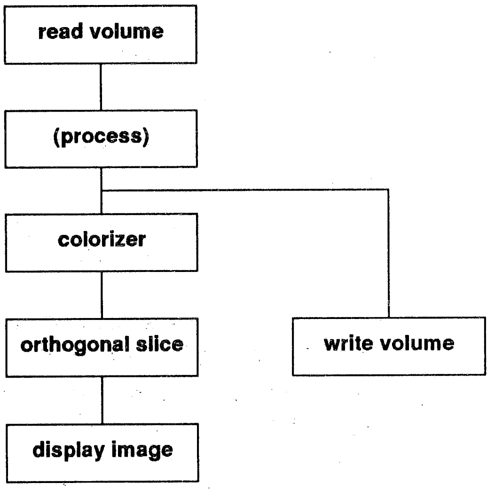
A file browser that allows you to specify the name of the volume data file to be created. The file suffix *.dat* is appended to the name automatically. If the file already exists, **write_volume** issues a warning message and has you confirm the operation ("Overwrite") or cancel it ("Cancel").

RELATED MODULES

read volume, clamp, contrast, crop, downsize, histogram stretch, interpolate, mirror, threshold, transpose

EXAMPLE

The data pre-processing networks in the AVS Volume Viewer all use the the following model:



where [process] is one of the following: **crop**, **downsize**, **mirror**, **transpose**, or **interpolate**. These networks are in the *volume_viewer* subdirectory of the *avs/networks* directory.

LIMITATIONS

The format of volume databases on disk is severely limiting. The dimensions are restricted to a maximum of 255 in x, y and z. The data also must be in the range 0 - 255.

Appendix A.
Tutorial: Image Viewer, Volume Viewer, and Network Editor

This appendix is a step-by-step tutorial for three of AVS's subsystems: the Image Viewer, Volume Viewer, and Network Editor. The Image and Volume Viewers allow you to perform visualization tasks without building any networks yourself. By making menu choices, you invoke preexisting networks. The Network Editor provides the tools for building and revising your own visualization networks.

Using the Image Viewer

The tutorial starts with an introduction to the Image Viewer subsystem. In this tutorial, you use AVS to view images, change their size, crop them, and adjust their color contrast.

Start an AVS session

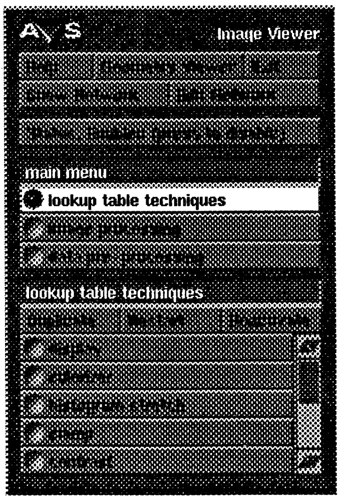
1. Type ***avs.***

An AVS session begins and the main menu appears at the left side of the screen.

Display an Image

2. Click ***Image Viewer.***

The Image Viewer subsystem begins by displaying its control panel at the left side of the screen.



3. Click ***display*** **under** ***lookup table techniques.***

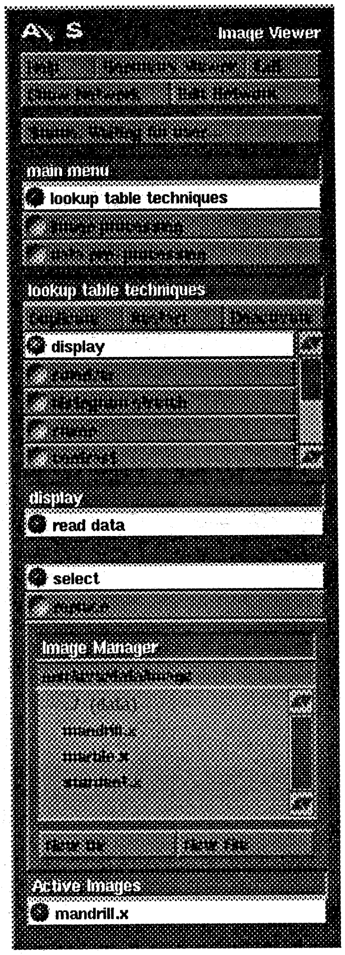
The *Image Viewer* control panel now includes widgets that are used by the *display* selection. One of these widgets is an *Image Manager* file browser. This "file browser" allows you to select a filename — in this case the name of a file that contains an image. Directory names appear in red; filenames appear in black. The default data directory (*/usr/avs/data*) does not contain any image files, so only the subdirectory names appear.

4. Click *image* in the *Image Manager* browser.

This selects the *image* subdirectory. The *Image Manager* browser now shows the names of the image files located in the subdirectory.

5. Click *mandrill.x* in the *Image Manager* browser.

This selects the mandrill image, which appears in the display window. (Note how the status indicator changes to red and the word **PROCESSING** is displayed. This tells you that AVS is busy processing your request). The *Image Manager* adds *mandrill.x* to the *Active Images* list, and a picture of a mandrill appears on the screen.



Zoom the Image

6. Magnify the image.

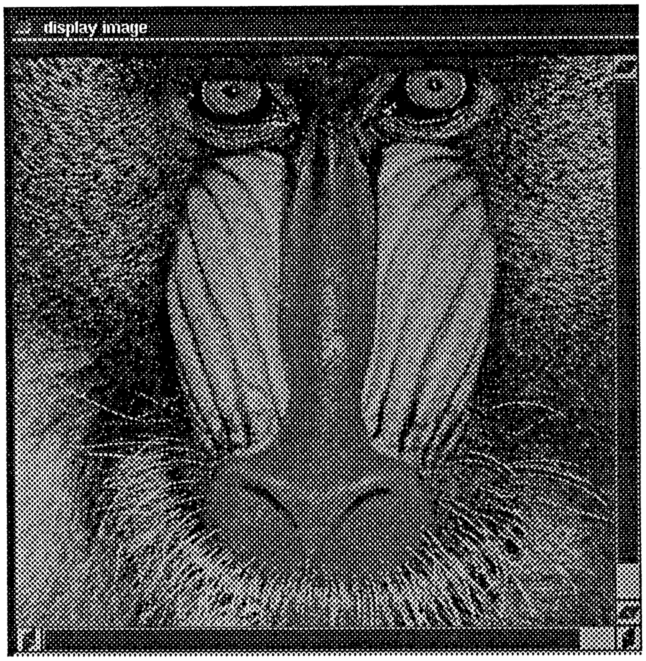
Move the cursor over the small square box (the "dimple") in the *Display Image* title bar. This is located above the picture of the mandrill. Press and hold down any mouse button. When the pull-down menu appears select *x2*. This magnifies the mandrill image by a factor of two.

Roam Through the Image

7. Make the *display image* window smaller by using the X window manager.

Notice that scroll bars appear on the right and bottom of the image. You can roam through the image by using these scroll bars.

The LEFT and RIGHT mouse buttons perform "jump scrolling" when you press the mouse button in the center of the scroll bar area. The CENTER mouse button performs "smooth scrolling" when you press the button and move the mouse in the center of the scroll bar area.



8. Press and hold any mouse button while the cursor is in the display image window.

A large dot appears indicating that you have grabbed the image in the display image window. Move the image by moving the mouse to roam the image directly.

9. Click *contrast* under *lookup table techniques.*

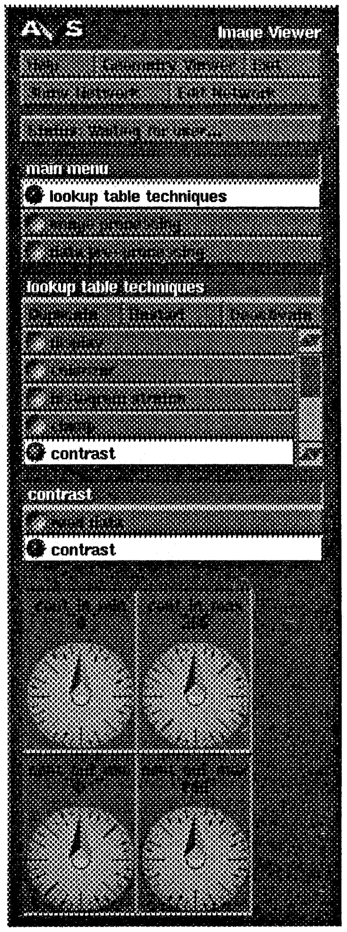
The control panel removes the selections for *display* and replaces them with menu selections for the *contrast* function. You must first select an image to be processed.

10. Click *mandrill.x* under *Active Images.*

The mandrill image appears in the new display window.

11. Click *contrast* under *contrast.*

The contrast technique's control widgets are displayed in the control panel. These dials are labelled *cont_in_min*, *cont_in_max*, *cont_out_min* and *cont_out_max*.



12. Move the *cont_in_min* and *cont_in_max* dials.

You can use any mouse button to move the dials. Either click the button at a new location on the dial, or drag the hand of the dial to a new position. The contrast does not change untill you release the mouse button.

13. Click *read data* under the *contrast* menu.

The *Image Manager* browser appears in the control panel. The current directory appears in the browser and contains directory names, (indicated in red).

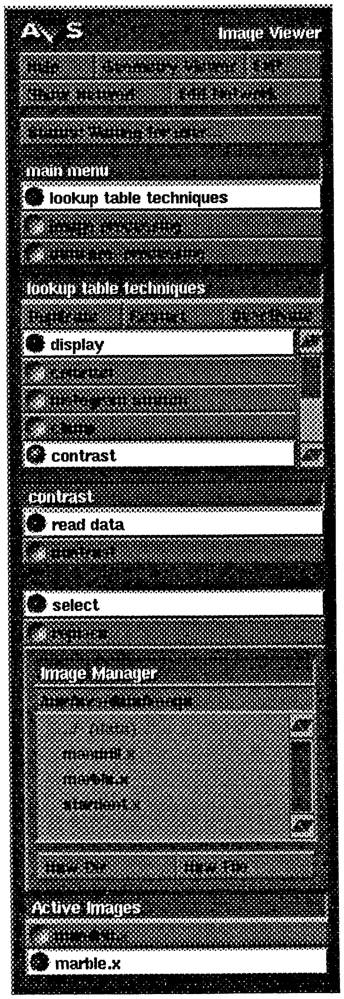
14. Click *image* in the *Image Manager* browser.

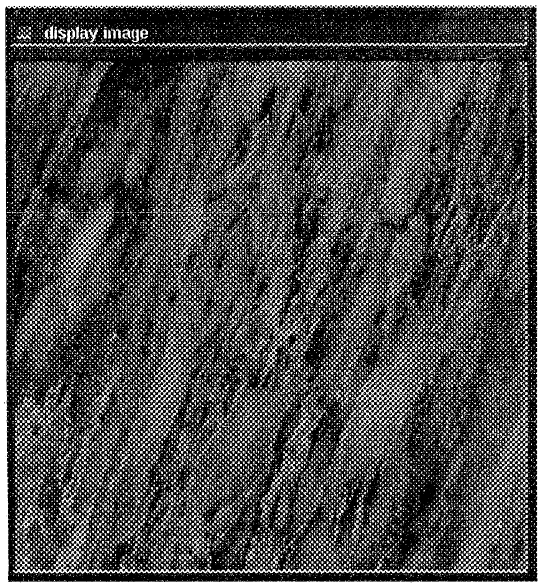
The contents of the *image* directory appears in the *Image Manager* browser.

15. Click *marble.x* in the *Image Manager* browser.

This selects the file *marble.x* in the image directory. The mandrill image is removed from the *display image* window and is replaced by the marble image.

The current control panel and the updated display window are shown in the figures below. Notice that the active image is *marble.x.*





16. Click *contrast* under *contrast*.

The contrast control widgets reappear. You can now modify the contrast of the marble image. Change the contrast bounds with the control widgets and watch the marble image change.

Crop An Image

17. Click *data pre-processing* under the *main menu*.

AVS provides preprocessing techniques to prepare data for further visualization. Cropping is one of these preprocessing techniques. It allows you to specify a subset of your data.

18. Click *crop* under *data pre-processing*.

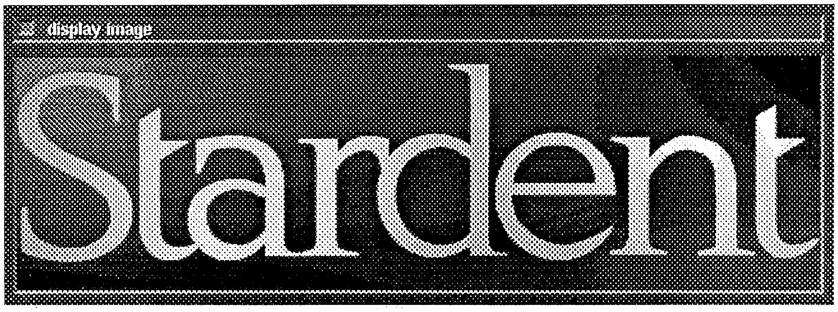
A new *display image* window appears. This window receives a new image that you select from the *Read Image* browser.

19. Click *image* in the *Read Image* browser.

The files residing in the *image* directory are listed in the browser.

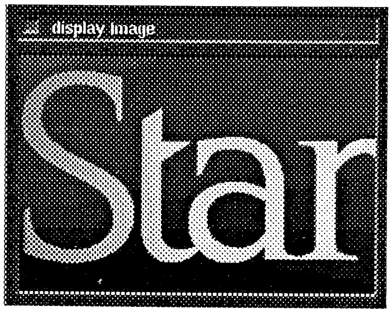
20. Click *stardent.x* in the *Read Image* browser.

The Stardent logo is selected and appears in the *display image* window.



21. Click *crop* under *crop.*

Manipulate the dials *min x*, *max x*, *min y*, and *max y* to eliminate data outside the specified range. The figure below shows the logo when the *max x* value is set to 280. This crops the right edge of the picture. The *min z* and *max z* dials have no effect because the image is two dimensional.



22. Click *read image* under *crop.*

The *Read Image* browser appears and you may use this file browser to select a new image. Select a new image as you have done before.

23. Click *crop* in the *crop* menu.

AVS is now prepared to crop the new image. Move the *crop control widgets* to crop the image data.

Use the Network Editor to Access Additional Functionality

24. Click *lookup table techniques* under *main menu.*

You will now examine the network that implements the contrast technique.

25. Click *contrast* under *lookup table techniques.*

The contrast menu appears in the control panel.

26. Click *Show Network* at top of screen.

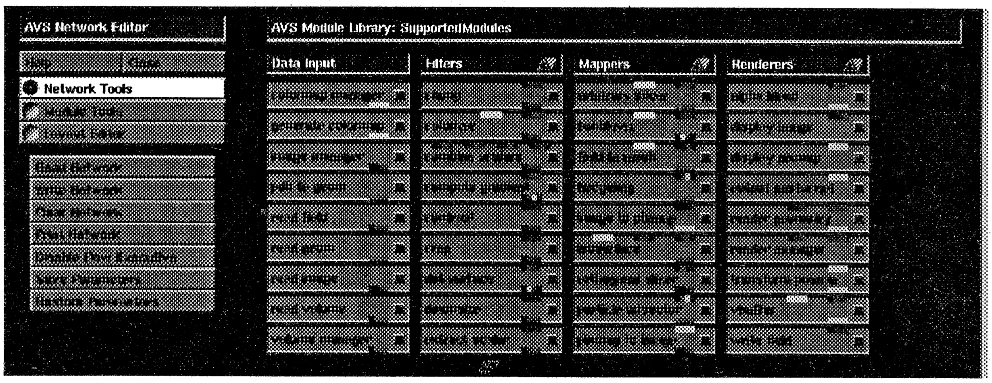
The current network is displayed on the screen. This network consists of three modules: *image manager*, *contrast*, and *display image*.

27. Click *contrast* under *contrast*.

Move the *contrast control widgets* with the mouse and watch the network diagram. The affected modules change color as the network executes.

28. Click *Edit Network* found at the top of the screen.

This starts the *Network Editor* subsystem. This process only re-instances the network. It does not save any of the information about dials, images, etc. You must repeat certain steps to re-create what you have already done in the Image Viewer. The *Network Editor* control panel is shown below.



29. Click *read data* under *Top Level Stack*.

The image data must be read into the network. The *Image Manager* module's file browser is displayed in the control panel so that you can make your selection.

30. Click *image* in the *Image Manager* browser.

The files in the *image* directory are displayed.

31. Click *mandrill.x* in the *Image Manager* browser.

The mandrill image is added to the *Active Images* list. You may want to move the mandrill image to a more suitable location by using the X window manager.

32. Click *contrast* under *Top Level Stack*.

Control widgets for the *contrast* module appear in the control panel. Move these widgets to perform contrast stretching. The network modules highlight as they are executed.

33. Click and hold on the contrast module with the LEFT mouse button.

The contrast module is located in the center of the network diagram. The module becomes a white wireframe box. Drag the module to the hammer icon in the lower right corner of the screen and release the mouse button. This deletes the module from the network.

34. Locate the *mirror* module.

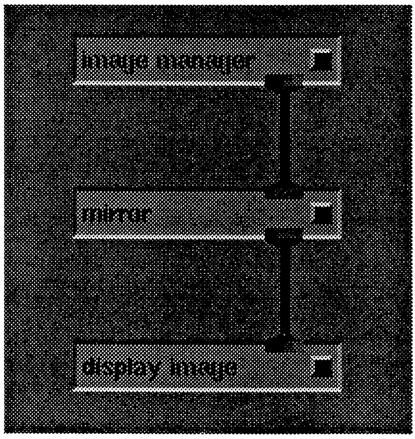
Move the cursor over the word *Filters* above the second column of modules. Scroll up and down using the LEFT and RIGHT mouse buttons to scan through the list of filters.

35. Connect the *mirror* module to the *image manager* module.

Grab the *mirror* module with the LEFT mouse button and drag it from the palette to the workspace. Place the cursor over the *mirror* input port, (the blue and grey rectangle on the top of the mirror box), and hold down the CENTER mouse button. A blue line is drawn connecting the *image manager* and *mirror* modules. Continue to hold down the CENTER button and move the cursor above the *mirror* module until the connecting line turns white. Release the CENTER mouse button. The thin white line is replaced with a thick blue line indicating that the connection has been successfully established.

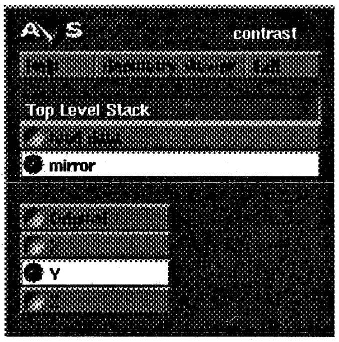
36. Connect the *mirror* module to the *display image* module.

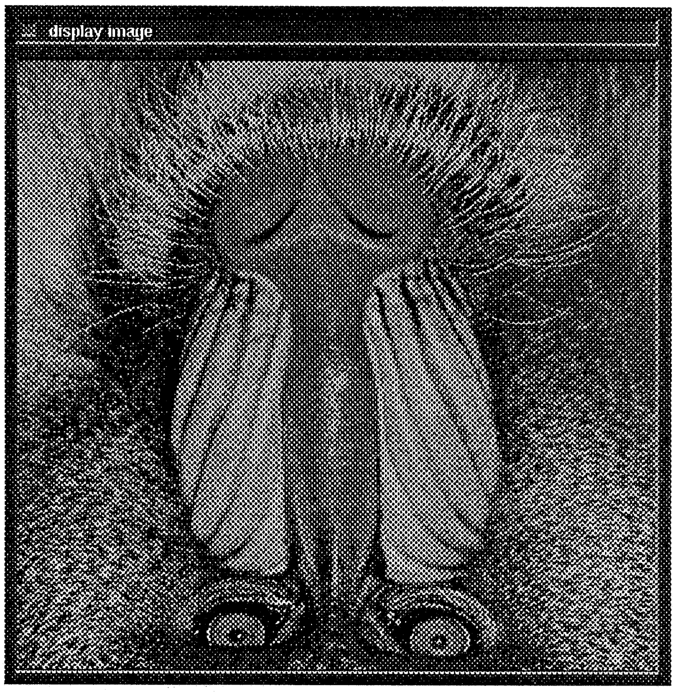
Using the same technique as described above, connect the *mirror* module and *display image* module together. At this point, the network should resemble the illustration below.



37. Click the *Y* button in the left hand control panel.

The image flips upside-down. Click the *Original* button to revert the image to its original state.





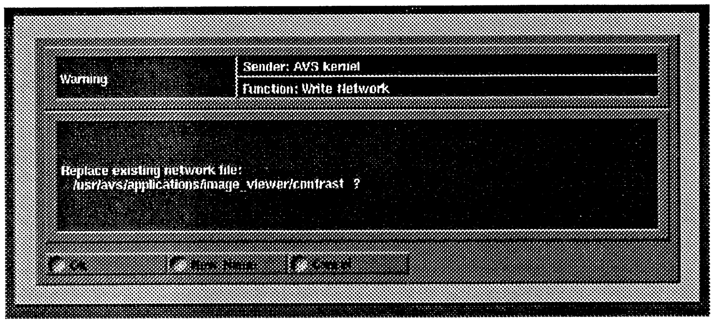
38. Replace the *mirror* module.

Any of the following modules can replace the *mirror* module: *clamp*, *contrast*, *crop*, *downsize*, *interpolate*, *threshold*, and *transpose*. Some of these modules take more time than others to execute. After adjusting a control widget, wait for the network to finish execution before requesting further processing.

Saving a Network

39. Click on the *Write Network* button.

If at any time you wish to save a network, you can click the *Write Network* button in the *Network Tools* submenu. A message window may appear, asking if you wish to replace an existing file.



Sender: AVS kernel

Function: Write Network

Replace existing network file:
/usr/avs/applications/image_viewer/contrast ?

40. Select the *New Name* option in the *Warning Window.*

A *Write Network* browser is displayed. Click *New File* or *New Dir* and type in the name of an appropriate file, (such as */tmp/mirror.net*).

Exit From AVS

41. Click *Exit* at the top right of the control panel.

This exits from the *Network Editor* subsystem. A message window appears to confirm that you want to exit.

42. Click *Ok* in the *Warning* window.

The main AVS menu is displayed.

43. Click *Exit AVS.*

A message is displayed to confirm that you want to exit from AVS.

44. Click *OK.*

The AVS session is complete and the shell prompt returns.

Volume Viewer Tutorial

The tutorial starts with an introduction to the Volume Viewer subsystem. You use it to view orthogonal slices of a volume, display isosurfaces of volumetric data and perform volume rendering using a voxel-blending technique.

Start an AVS session

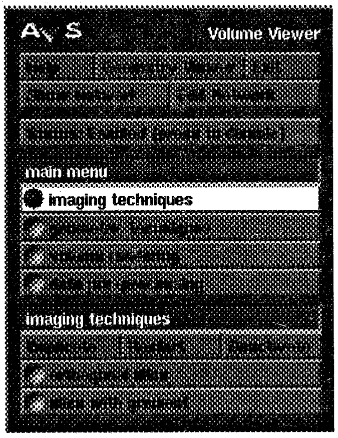
45. Type *avs.*

An AVS session begins and the main AVS menu is displayed.

Display Slices from a Volume

46. Click *Volume Viewer.*

The Volume Viewer subsystem is selected and its control panel is displayed on the left side of the screen.

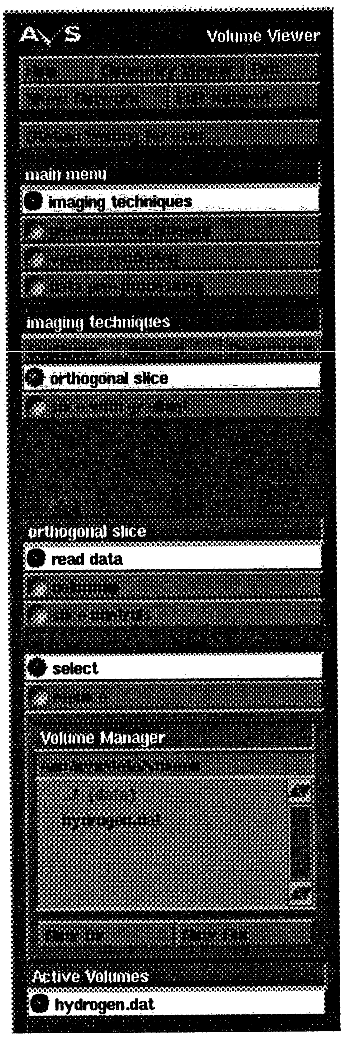


47. Click *orthogonal slice* under *imaging techniques.*

The control panel now contains widgets to visualize an orthogonal slice of volumetric data. A *Volume Manager* browser appears so that you may select a volume for viewing.

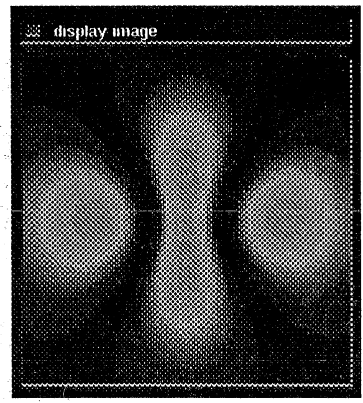
48. Click *volume* in *Volume Manager* browser.

The *volume* directory is located at the end of the directory listing. You need to use the browser's scroll bar to scan to the bottom of the list.



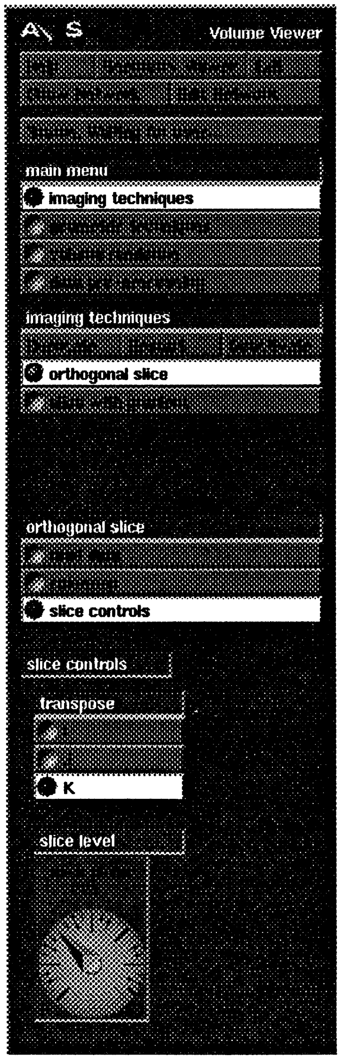
49. Click *hydrogen.dat* in the *Volume Manager* browser.

A disk file containing a simulated orbital of a hydrogen atom is selected. Notice the status indicator is red and the word **PROCESSING** is displayed. This tells you that AVS is busy processing your request. A slice through the middle of the volume appears in the *display image* window.



50. Click *slice controls* under *orthogonal slice*.

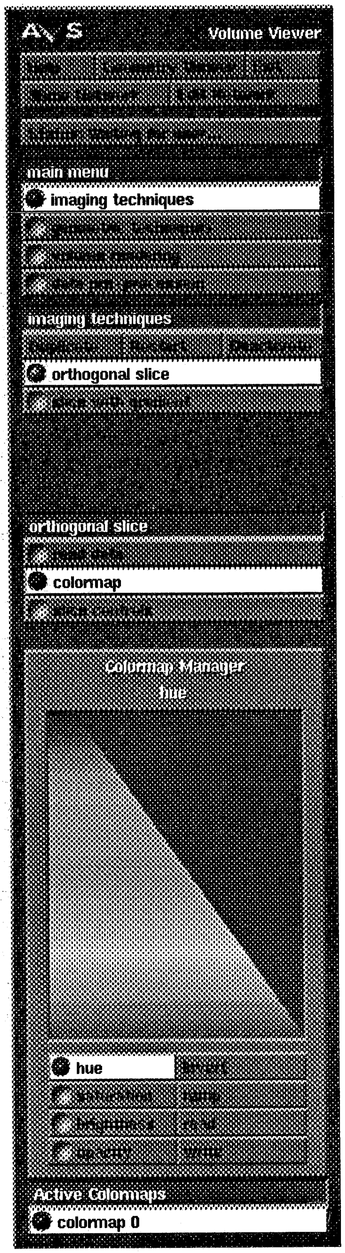
A set of *Slice control widgets* appear in the control panel. Use any mouse button to manipulate the dial labelled *slice plane*. This selects different orthogonal slices of the hydrogen volume.



Change the Pseudo-Color Mapping of the Data

51. Click *colormap* under *orthogonal slice.*

A *Colormap Manager* appears which allows you to modify the current colormap. The control panel containing the *Colormap Manager* is shown below.



52. Click the *read* button.

This button is located on the lower right side of the colormap editor. It enables AVS to read an existing colormap from disk. A *Colormap* browser appears in the middle of the screen.

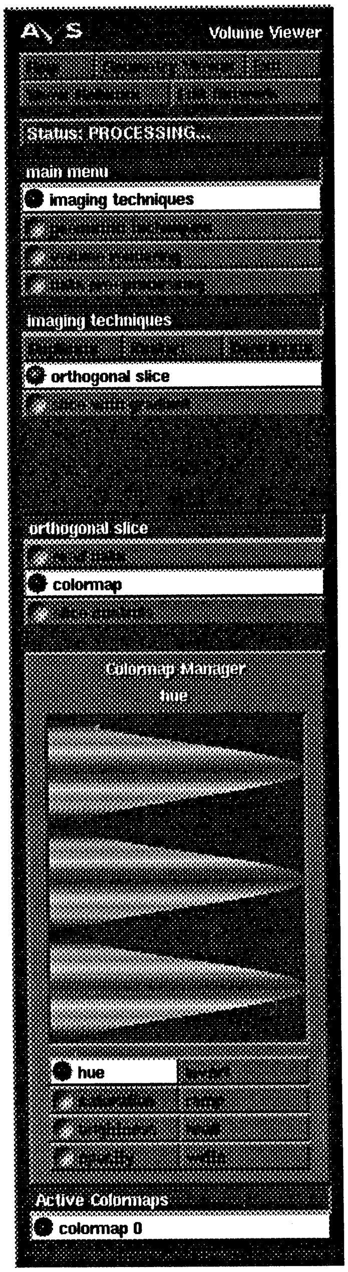
53. Click *colormap* in the *Colormap* browser.

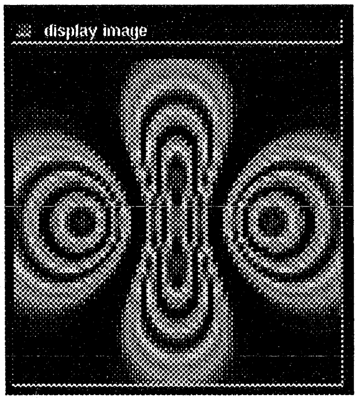
This selects the colormap directory. Any files which have a ".cmap" suffix are text files which specify colormaps.

54. Click *color_wave.cmap*.

The *Colormap Manager* displays the *color_wave.cmap* colormap. The orthogonal slice in the *display image* window is redisplayed using this colormap.

The following two illustrations show the orthogonal slice and the *Colormap Manager* after reading *color_wave.cmap* from disk.





55. Re-slice by repeating step 50.

Different orthogonal slices of the hydrogen volume are displayed using the new colormap.

Disable the Orthogonal Slicer

56. Click the *Deactivate* button under *imaging techniques*.

The widgets and windows belonging to the *Orthogonal Slicer* are removed.

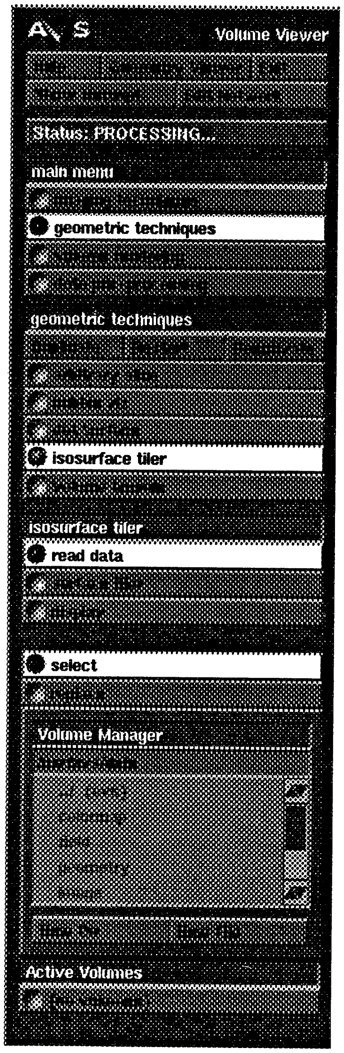
Display An Isosurface for this Data

57. Click *geometric techniques* under *main menu*.

Geometric techniques appear in the control panel. These techniques are used to visualize the volume in three dimensions.

58. Click *isosurface tiler* under *geometric techniques*.

A *Volume Manager* browser appears in the control panel. The current directory contains a list of directories, (indicated in red).



59. Click *volume* in the *Volume Manager* browser.

You need to use the browser's scroll bar to scan to the bottom of the directory listing.

60. Click *hydrogen.dat* in the *Volume Manager* browser.

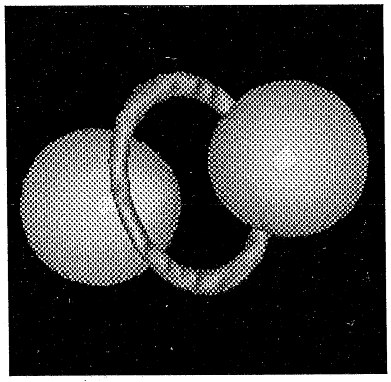
An isosurface for a layer of the hydrogen volume appears in the geometry window.

61. Rotate, translate and scale the object.

You may apply graphical transformations to the object using the mouse. By holding down the CENTER mouse button and moving the mouse in the geometry window, the object rotates. Holding the RIGHT mouse button and moving the mouse translates the object. Pressing the SHIFT key while holding the CENTER mouse button and moving the mouse scales the object.

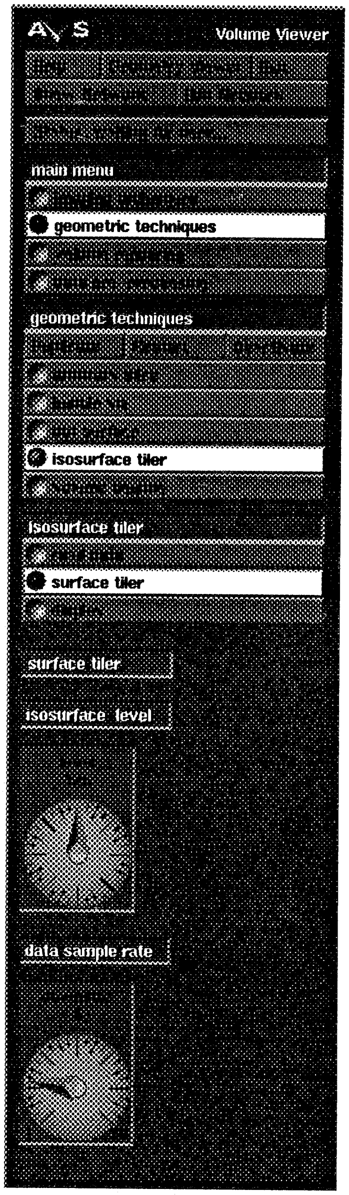
62. Click *surface tiler* under *isosurface tiler.*

Surface tiler control widgets appear in the control panel. The displayed volume is a sub-sampled version of the data base. To increase the resolution, change the *downsize* control widget from 4 to 2. The isosurface now looks smoother.



63. Move the control widget labelled *isosurface level.*

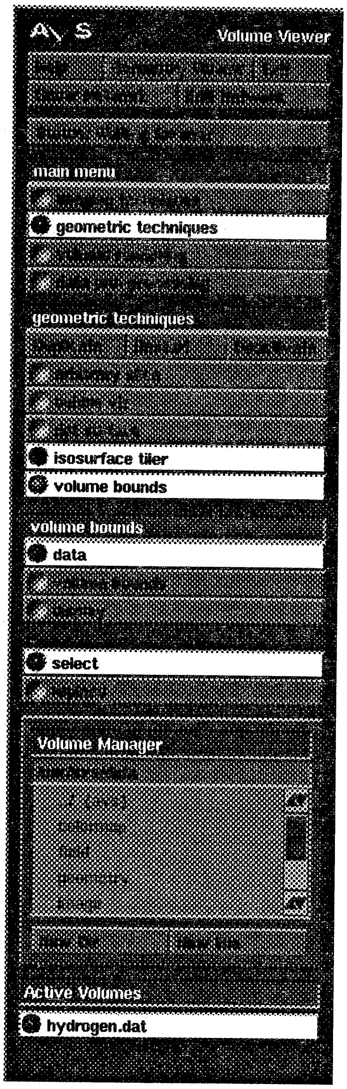
The level of the isosurface contour changes. Different depths of the volume are rendered as you move the *isosurface level* control widget.



Add a Bounding Box around the Isosurface

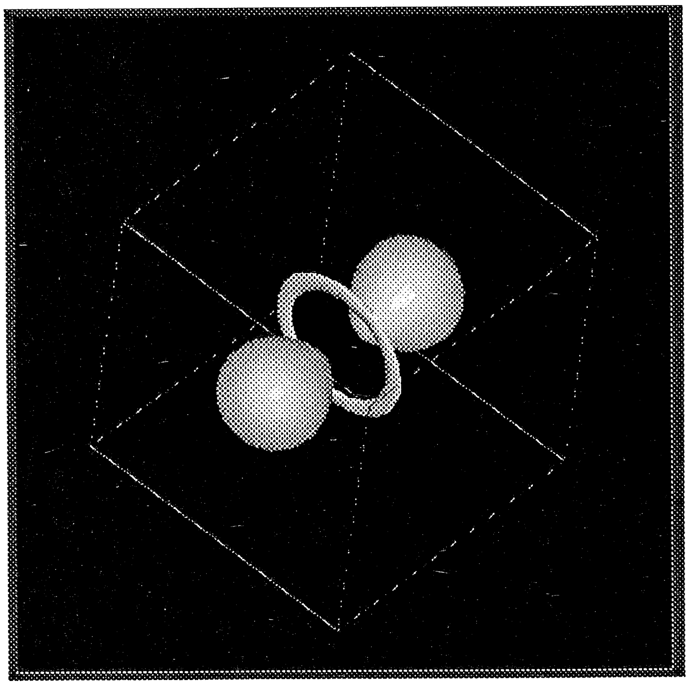
64. Click *volume bounds* under *geometric techniques*.

The *Volume Manager* browser appears in the control panel.



65. Click *hydrogen.dat* under *Active Volumes.*

A bounding box surrounding the isosurface is drawn around the volume. This helps you visualize the extent of the data.



Add An Arbitrary Slice Plane to the Isosurface and Bounding Box

66. Click *arbitrary slice* under *geometric techniques.*

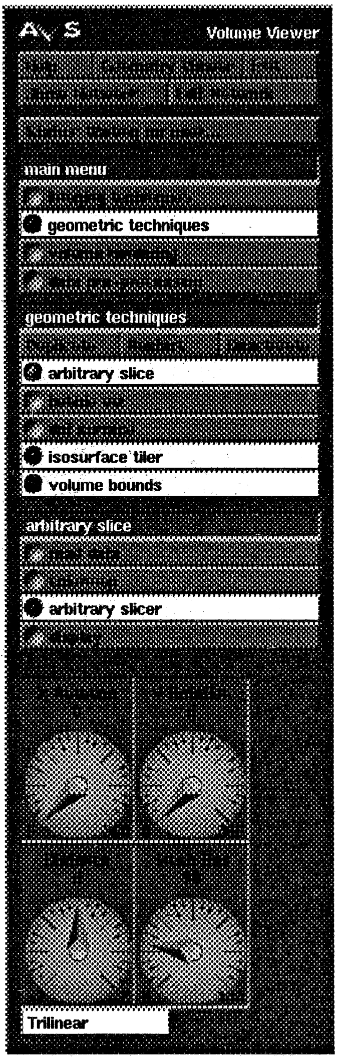
The *Volume Manager* file browser appears in the control panel. This allows you to select a data file.

67. Click *hydrogen.dat* under *Active Volumes.*

An arbitrary slice through the volume appears within the bounding box. Rotate the object at any time to get a better view of the object. (Object rotation is described in step 61.)

68. Click *arbitrary slicer* under *arbitrary slice.*

The control widgets for the arbitrary slice appear on the screen. You may move the slice by dialing the *X Rotation*, *Y Rotation* and *Distance* control widgets.



Make the Isosurface Smaller

69. Click *isosurface tiler* under *geometric techniques*.

If the isosurface is too big and obscuring the slice plane, you may want to make the isosurface smaller. You can make it smaller by interacting with the surface tiler.

70. Click *surface tiler* under *isosurface tiler*.

The *surface tiler* control widgets appear in the control panel.

71. Change the dial labelled *level*.

This re-tiles a surface on a different layer of the volume. Levels between 150 and 255 work well for this example.

72. Continue manipulating the slice plane.

Repeat steps 66 and 68.

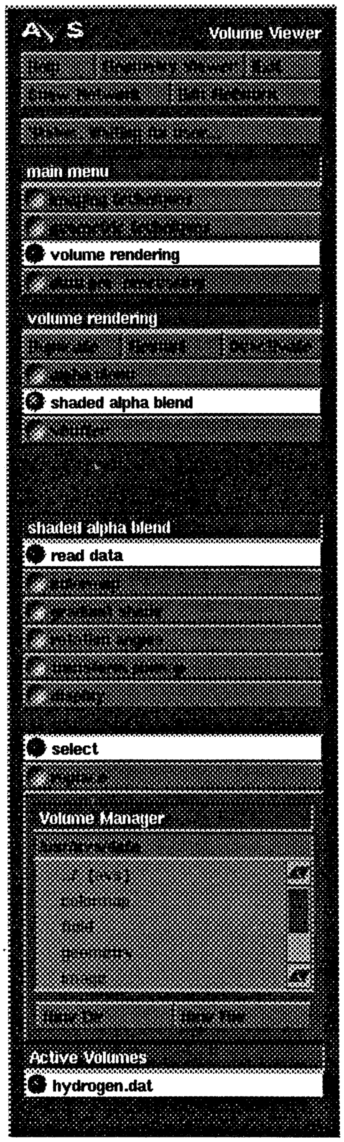
Create a Semi-Transparent Version of this Volume

73. Click *volume rendering* under *main menu.*

This enables the *volume rendering* functions and displays the *volume rendering* widgets in the control panel.

74. Click *shaded alpha blend* under *volume rendering.*

An empty *display pixmap* window appears on the screen. The current control panel is shown below.

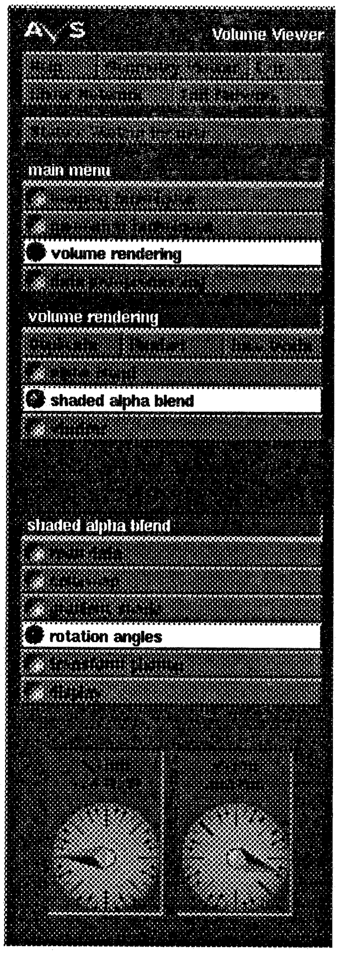


75. Click *hydrogen.dat* under *Active Volumes.*

A new representation of the hydrogen volumetric data is displayed in the *display pixmap* window. This representation contains surface properties, including transparency.

76. Click *rotation angles* under *shaded alpha blend.*

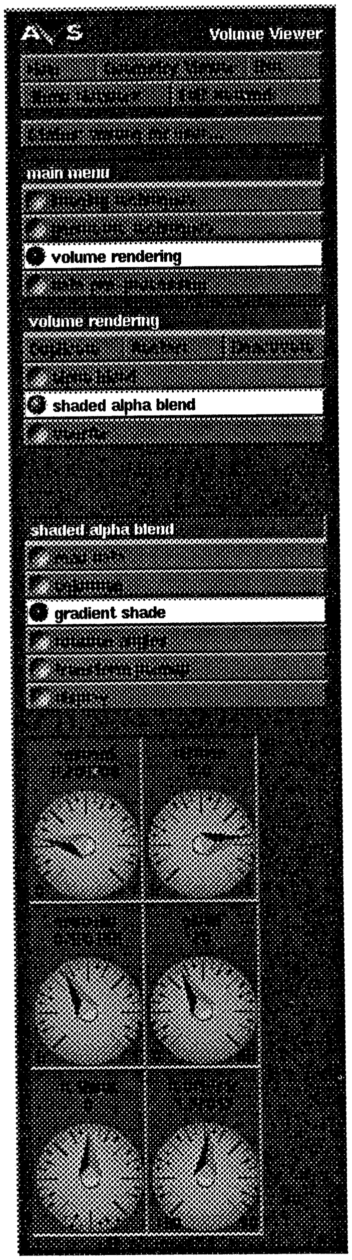
You may now rotate the object. Manipulate the *Y-Rot* dial to rotate the data set around the y-axis. The data is symmetrical around the X-axis, so the *X-Rot* dial has no effect.



77. Click *gradient shade* under *shaded alpha blend.*

The surface properties and light source are changed by manipulating any of the six dials which have appeared at the bottom of the control panel.

For instance, to swing the light source around the object, move the *lt off-ctr* dial. To make the object shiney, change the *specular* dial from 0 to between 0.5 and 1.0.



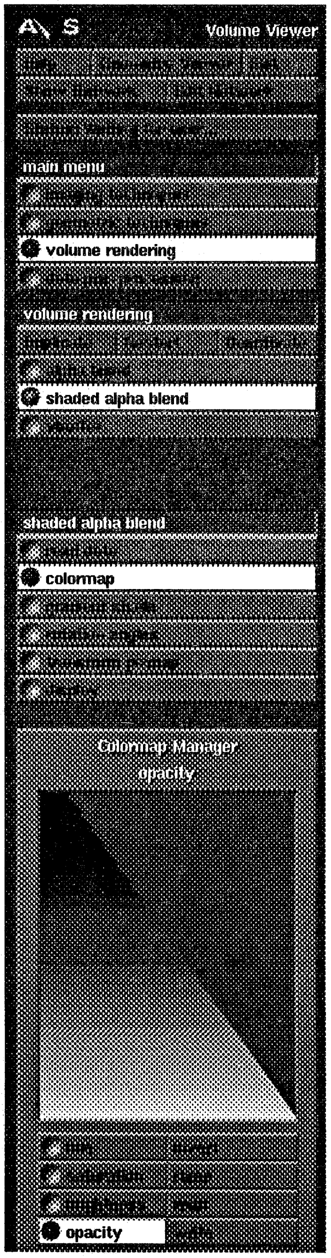
78. Click *colormap* under *shaded alpha blend.*

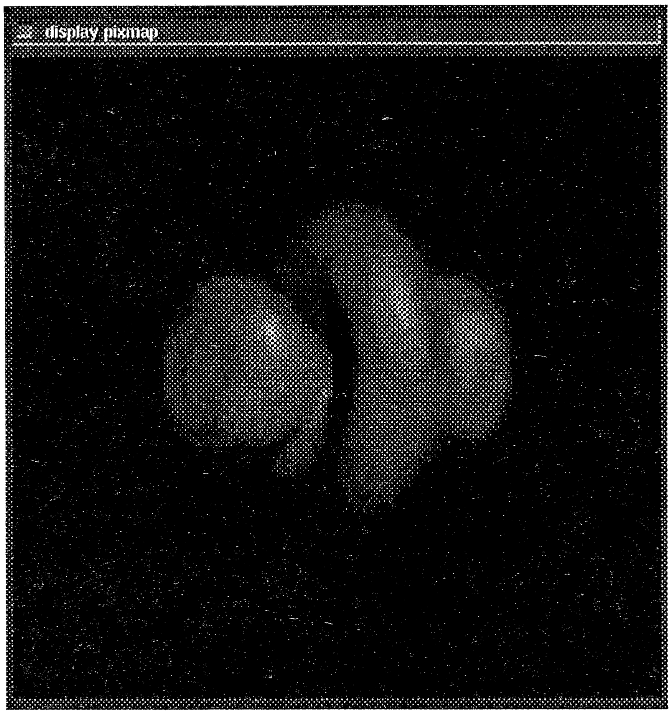
The *Colormap Manager* appears in the control panel.

79. Click *opacity*, found at the bottom of the *Colormap Manager.*

Now you may change the opacity of the voxels by drawing a new opacity curve. To do this, move the cursor into the window that has the grey-scale ramp. Hold down any mouse button and draw a new curve. When you release the button a new image is generated in the *display pixmap* window.

The top of the curve represents voxels with low values. Here they are colored blue. The bottom of the window represents voxels with high values. Black and dark grey areas in the colormap represent transparent values. Bright areas represent opaque values.





Use the Network Editor to View a Curvilinear Data Set

80. Click *geometric techniques* under *main menu.*

You will now look at the network that displays a bounding box around a volume.

81. Click *volume bounds* under *geometric techniques.*

This instances the *volume bounds* network.

82. Click *Show Network* at the top of the screen.

A network is displayed which consists of three modules. The *volume manager* reads data from disk. The *volume bounds* computes the boundary of the volume. The *render manager* module renders the object.

83. Click *Edit Network* at the top of the screen.

This takes you to the *Network Editor* subsystem. You may need to resize or move the black renderer output window in order to see and use the *Network Editor.* Do this using the X window manager.

84. Click *data* under *Top Level Stack.*

This selection is found in the control panel along the left side of the screen. A *Volume Manager* appears so that you can select a data file.

85. Click *volume* in the *Volume Manager* browser.

You need to scroll the browser using the scroll bar along the right side of the browser.

86. Click *hydrogen.dat.*

This activates the current network so that a bounding box is displayed. A wireframe cube appears in the black renderer output window. Rotate the cube by moving the cursor into this window and holding down the CENTER mouse button while moving the mouse.

87. Remove the *volume manager* module from the network.

Grab the *volume manager* module by placing the mouse over the box labelled *volume manager* and pressing the LEFT mouse button. The module becomes a white wireframe box. Drag the box to the hammer in the lower right hand corner of the screen. The module is removed.

88. Locate the *read field* module.

Move the cursor over the word *read field* in the *Data Input* section of the *Network Editor* menu. Drag this module from the palette to the workspace. Press and hold the LEFT mouse button on the module then position it above the module labelled *volume bounds*.

89. Connect the *read field* module to the *volume bounds* module.

Place the cursor over the input port of the *volume bounds* module, (the blue and grey box at the top of the icon). Hold down the CENTER mouse button. A thin blue line should appear going to the *read field* module. Continue to hold down the CENTER mouse button and move the cursor towards the *read field* module untill the blue line connecting the *read field* module and the *volume bounds* module turns white. Now, release the CENTER mouse button and the thin white line should get replaced with a thick blue line. This indicates that the connection is successful.

90. Click *field* in the *Read Field* browser.

The *field* directory is displayed in the *Read Field* browser.

91. Click *curvilinear.fld.*

The network reads the curvilinear data set which appears in the renderer output window. This field was computed with PLOT3D. It represents air density over a blunt fin.

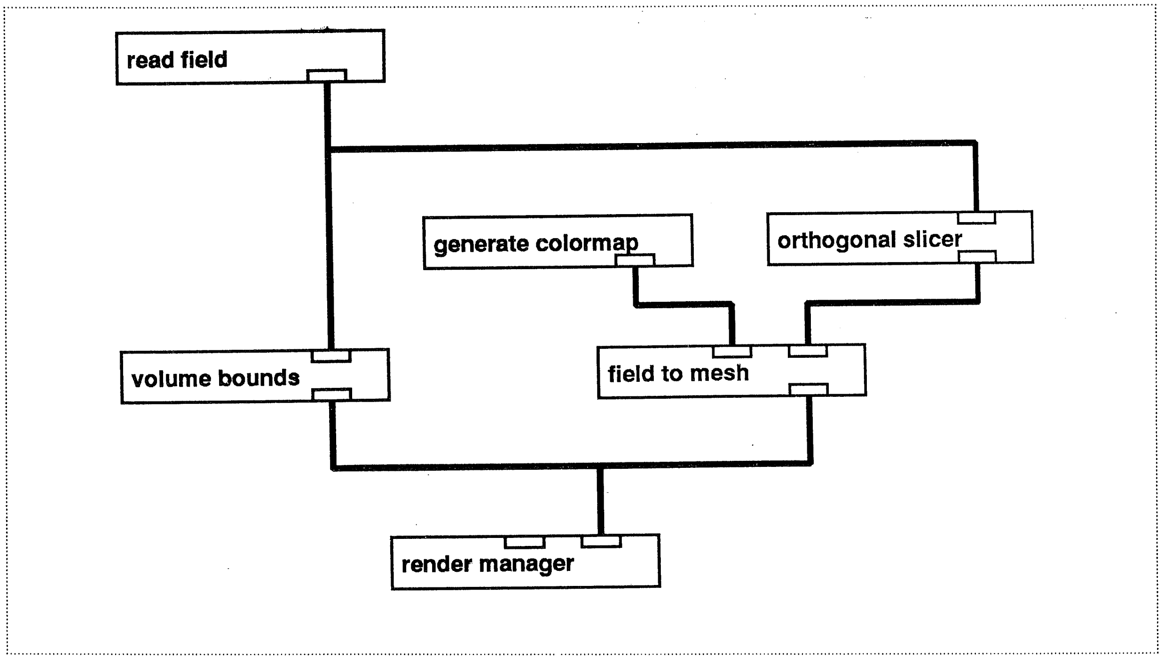
92. View the entire object.

To view the entire object, first move the cursor into the renderer window. Hold down the SHIFT key on the keyboard and press the CENTER mouse button. Sweep the cursor to the left. Scale the object untill the entire object is seen in the renderer window.

Take Slices of Data through the Object

93. Examine the data using a slice plane.

You need to insert three new modules into the network: *generate colormap* (found under the *Data Input* column), *field to mesh* (found under the column labelled *Mappers*), and *orthogonal slicer* (also found in the *Mappers* column). Drag them to the working area and connect them to the network as you did in steps 88 and 89. The network you will design is diagramed below.



A colored plane is displayed in the renderer output window. This shows air density inside the wireframe bounds.

94. Click *generate colormap* under *Top Level Stack.*

You need to modify the range of colormap values to lie between 0.0 to 3.0. A set of dials appear in the control panel which can adjust the colormap range.

95. Move the *hi value* dial to 3.0.

This allows you to visualize the complete range of air density data.

Change the Location of the Air Density Plane

96. Click the *orthogonal slicer* button under *Top Level Stack.*

Control widgets that manipulate the Air Density Slice appear in the control panel. You may turn the dial labelled *slice plane* to move the slice through the curvilinear field. To view slices in other directions, hit the buttons labelled *I*, *J*, or *K*.

Exit From AVS

97. Click *Exit* at the top right of the control panel.

This exits from the *Network Editor* subsystem. A warning message appears to confirm that you want to exit.

98. Click *Ok* in the *Warning* window.

The main AVS menu is displayed.

99. Click *Exit AVS.*

A window is displayed to confirm that you want to exit from AVS.

100. Click *OK.*

The AVS session is complete and you return to your shell prompt.

Appendix B. Geometry Viewer Tutorial

This tutorial walks you through several sessions with the AVS Geometry Viewer application. We start with the simplest case — working with a single, simple object. Then, we move on to creating a "scene" out of more than one object. We add lights to the scene, then add another "camera" (that is, another window to view the same scene from a different angle).

You probably have noted that we use a "movie studio" metaphor in describing the Geometry Viewer. The viewing application really does enable you to accomplish many of the same tasks as a movie director — you specify objects and compose them into scenes; you adjust the lighting; you shoot the scene from one or more camera angles.

The viewing application itself adopts the movie studio metaphor in its menu structure, which includes such choices as *Lights*, *Camera*, and *Action*.

NOTE *We assume that you (or someone at your site) have installed the Application Visualization System tape according to the instructions in the* ***AVS Installation and Release Note.*** *If not, you may have to deal with operating system issues, such as adjusting your* **PATH** *variable.*

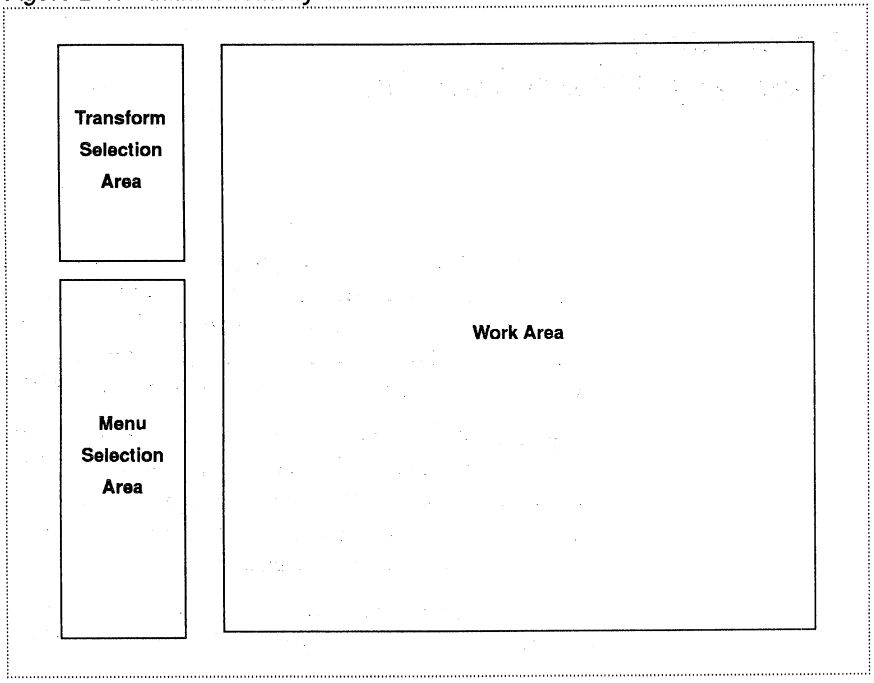
Session 1: Working with a Single Object

To get ready to run AVS, simply sign onto the Stardent Graphics Supercomputer, just as you always do. There is no need to change to a special directory first. To start the program, type this command:

```
    avs
```

Within a couple of seconds, the main AVS menu appears. Select the Geometry Viewer subsystem. An empty window appears on the screen, along with a control panel along the left edge:

Figure B-1. Initial Geometry Viewer Screen



The empty window is the viewing area. Think of it as a "scene" in which no objects have been placed yet. The control panel has two main areas:

- ❑ The *Transform Selection* area allows you to control the way in which the mouse works. At any point during a session, the mouse is in control of an object, a light, or a camera.
- ❑ The *Menu Selection* area is where you invoke most of AVS's functions: retrieving and storing data, specifying colors and lighting, etc. The main menu is always visible:

```
Objects
Lights
Cameras
Action
```

Note the red ball that indicates **Objects** is the current menu selection. Accordingly, the lower part of the Menu Selection area shows the various **Objects** menu choices.

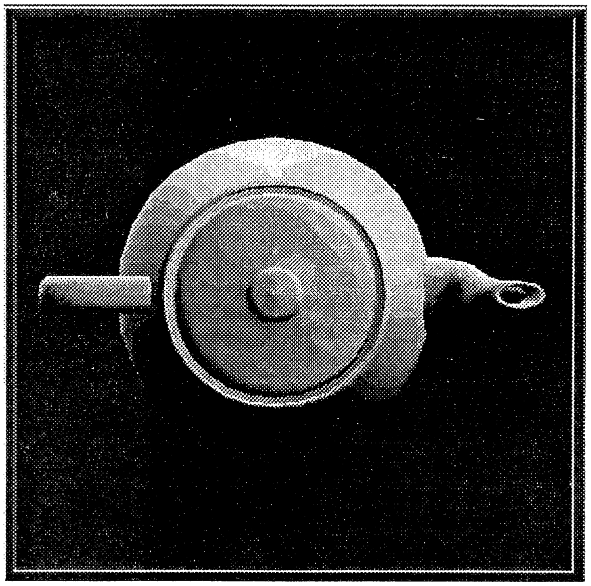
Reading in an Object

Let's bring an object into the work area. You'll note that the **Objects** menu selection is activated automatically at the beginning of the session. Thus, a variety of object-related options are listed in the menu selection area. Click on **Read Object.**

A new window, the **File Browser**, pops up, listing the files in the current directory that describe objects AVS can display. At the top of this window is the name of the current directory (initially, *data*). Click on *teapot.geom*.

You can add your own objects to this list, too — we'll do that later on. We'll also explain the File Browser's file naming conventions, and how to access data in other subdirectories.

Figure B-2. A Teapot

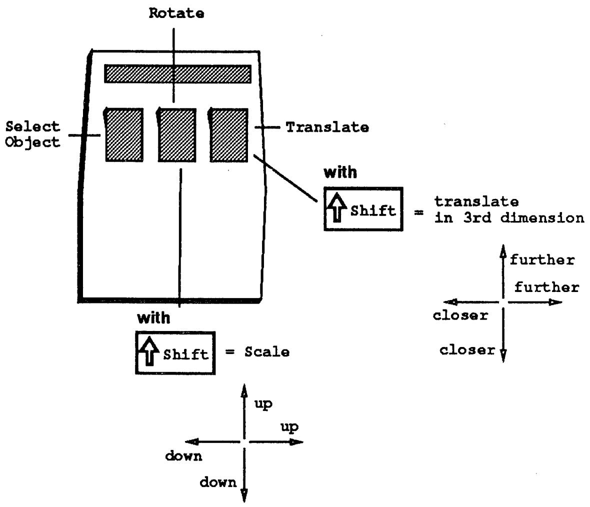


Note that the File Browser window remains onscreen. This makes it easy for you to bring several objects into a scene at once. For now, however, the teapot is enough, so click **Close** at the bottom of this window to remove the File Browser from the screen.

Manipulating an Object

Now, let's move the object around a bit. If you've done any graphics work at all, you know that the three basic ways to manipulate an object are *translation*, *rotation*, and *scaling*. Using the mouse, you can accomplish all of these:

Figure B-3. Functions of the Mouse Buttons



❑ *Translation:* When you place the mouse pointer on the teapot and hold down the right mouse button, you can drag the teapot all around the window. Give it a try! You can even move the teapot completely out of sight — if you do, click the **Reset** choice at the bottom of the Translation Selection area; this returns an object to its original position, in the middle of the window.

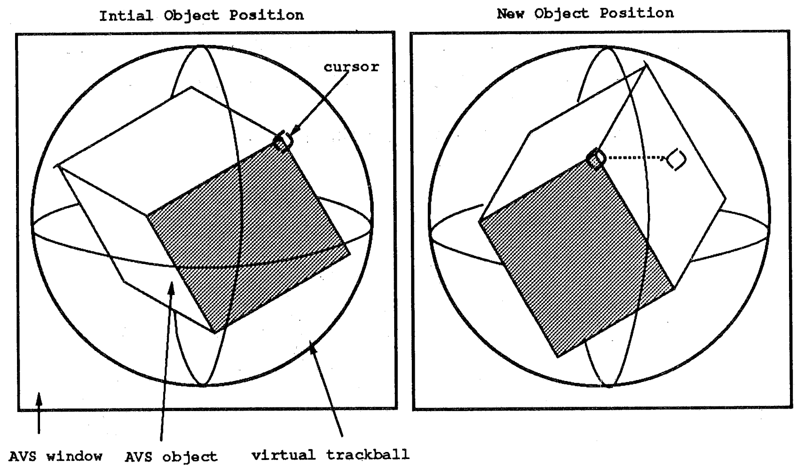
So far, you've translated the teapot only within a single plane. (It's the "X-Y plane" or the "Z=0" plane. By default, the positive Z axis comes directly out of the screen.) To translate the teapot in the third dimension, hold down the **Shift** key on the keyboard while dragging with the right mouse button. This may take a little while to get used to. Dragging downward or to the left moves the object closer to you; dragging upward or to the right moves it further away.

Note that you can make the teapot disappear by translating it too close or too far away. This is because AVS displays only a finite *view volume*. As an object passes though the near or far *clipping plane*, more and more of the object disappears. When it is entirely beyond the clipping plane, it vanishes completely. (If you lose an object, use the **Reset** button in the Transform Selection area to bring it back.)

❑ *Rotation:* When you place the mouse pointer on the teapot and hold down the middle mouse button, you can rotate the teapot in three dimensions. (It always rotates around its own center, rather than around some fixed point in the scene.) Try it, using the spout as the place where you "grab" the teapot.

To understand how this works, imagine that a trackball device is embedded in the display screen. The mouse cursor is, in effect, attached to the surface of this "virtual trackball":

Figure B-4. Rotating an Object with the Virtual Trackball



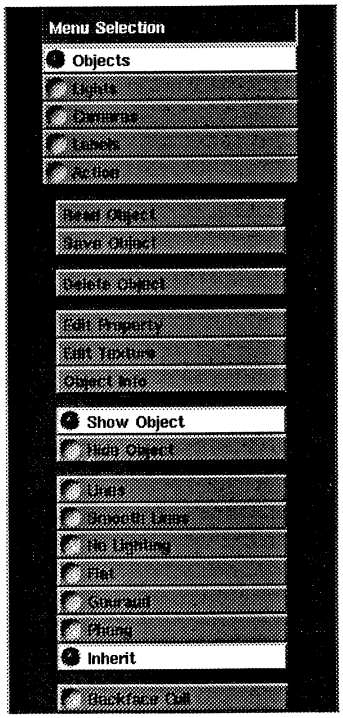
With a little practice, you can get the teapot "rolling", so that it continues to rotate by itself. To do this, release the middle mouse button while you are still moving the mouse. To stop a spinning object, just click the middle mouse button.

❑ *Scaling:* To make the teapot grow or shrink, hold down the **Shift** key on the keyboard while dragging with the middle mouse button. Dragging to the right makes the object larger; dragging to the left makes it smaller.

If you make an object too big, all its surfaces will fall outside the finite view volume, and the object will disappear. As before, remember the **Reset** choice.

NOTE *In general, you make a menu choice in the control panel by clicking once with any mouse button. You can click anywhere within a menu-choice rectangle — AVS confirms the click:*

Figure B-5. Clicking on a Menu Choice



Changing the Properties of an Object

While moving the teapot around the screen, you've no doubt noticed various aspects of its appearance. The entire teapot is a single color, but this color is subtly shaded rather than being uniform. The shading at a particular point depends on the shape, and on the angle between the surface and the light source. There are specular highlights resulting from a light source located directly in front of the screen.

AVS allows you to control all these aspects of an object's appearance:

❑ The apparent color of an object depends on its actual color, but also depends on the way the surface reflects light. You can set these "surface properties" (or "material properties") for the teapot by selecting **Edit Property** under the Object menu selection.

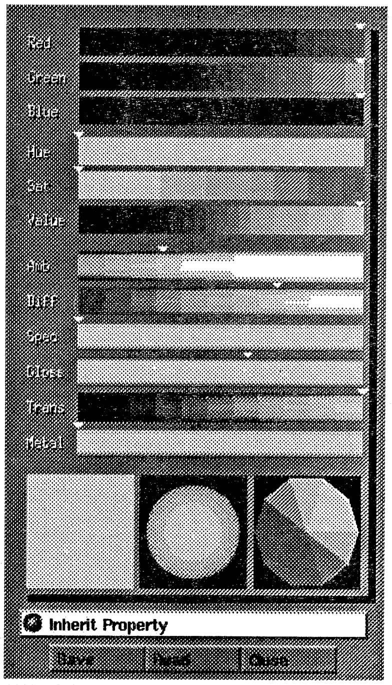
The apparent color also depends on the color of the light(s) which illuminate the object. We work with lights in the next session of the tutorial.

❑ The appearance of specular highlights on an object depends both on its surface properties and on the available light(s). In this session, we'll manipulate the surface properties only — we defer changing the lighting until the next session.

❑ The degree of realism in an object's image depends on the *rendering method* used to draw it. Under the Object menu selection, there are several rendering-method choices: **Lines**, **Smooth Lines**, **Flat**, and so on.

Let's start by changing the surface properties of the teapot. Click on **Edit Property** to open a window that contains slider controls for various surface properties:

Figure B-6. The Edit Property Window



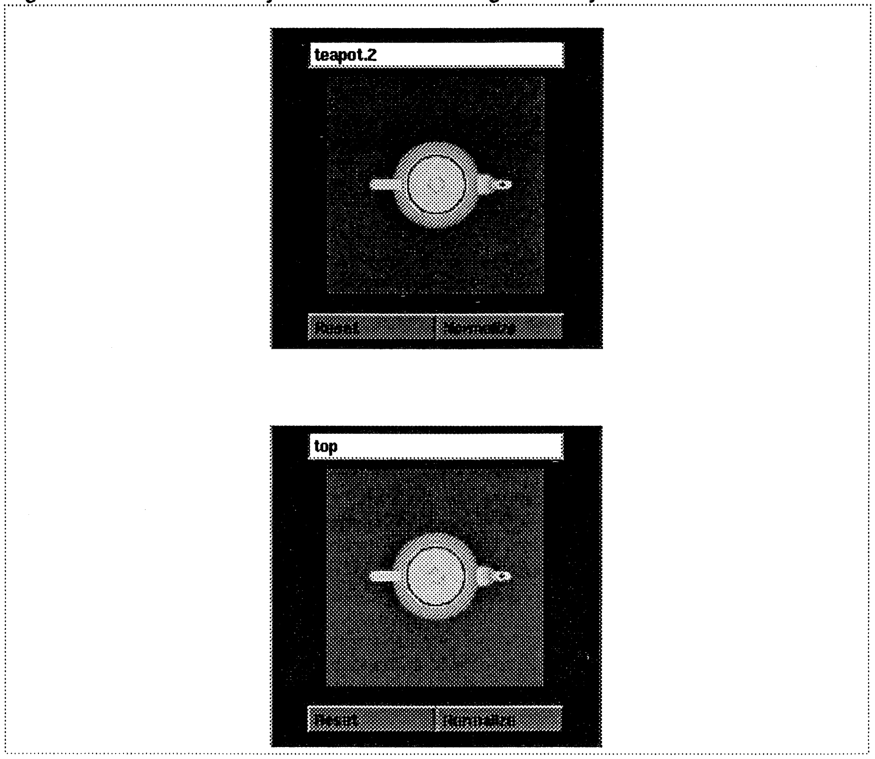
NOTE *Before we start working with the Edit Property window, remember that AVS can handle multiple objects. When you set about altering the properties of an object, be sure that you know exactly* **which** *object you're currently working with. In our situation, there are actually two slightly different objects:*

- ❑ *The "top-level" object, which hierarchically includes all the objects that you've read into the scene. At this point, "all the objects" is just the single teapot.*
- ❑ *The teapot itself, as an individual object.*

If you specify a change in the color of the top-level object, it may have no effect on the image. That's because the teapot itself can have a color, which overrides the color setting at the top level. Each object's color can be either set explicitly or inherited from its parent object. Explicit settings at lower (more specific) levels in the hierarchy override settings at higher levels.

To make sure that the teapot as an individual object is selected, click on the teapot repeatedly with the left mouse button, and watch the ***Current Object Indicator*** *in the Transform Selection area.*

Figure B-7. Current Object Indicator Showing Two Objects



Each time you click, you move to a different level in the scene's object hierarchy. The label above the Current Object Indicator also helps you to keep track:

`top` *Indicates that the top-level object is currently selected*

`teapot.1`
Indicates that the teapot as an individual object is currently selected (AVS adds a numeric suffix to help you keep track of objects. If you read in the same object twice, the instances are assigned two different numeric suffixes.)

Stop clicking when you are sure that the teapot itself is selected, not the top-level object. You may notice that each time you click the Edit Property window changes, so that it always shows the settings of the current object.

We'll examine object hierarchy and its implications for attribute setting more closely in the third session of this tutorial.

The first six sliders control the surface color of an object. You can specify the color in two ways:

❑ The first three sliders adjust the Red-Green-Blue (RGB) components of the color.

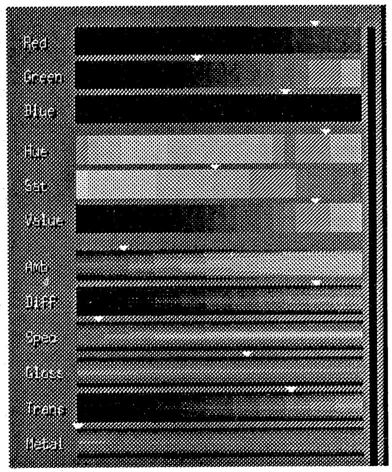
❑ The second three sliders also adjust the color, by specifying its Hue-Saturation-Value (HSV) components.

You can use any mouse button to adjust a slider. Just click where you want the slider to move to. (Alternatively, drag the slider from its current location to a new one.)

When you adjust a color-control slider, the teapot's color changes as soon as you release the mouse button. Also note that the HSV sliders automatically adjust if you change the RGB specification, and vice versa. There's only one current color, but you can specify it in two different ways.

Below the color-control sliders is a group of reflectance-control sliders.

Figure B-8. Material Properties Sliders



□ The fraction of **ambient** (uniform, directionless, background) light that the object reflects. Move the slider to the right to make the teapot appear brighter.

□ The fraction of **diffuse** light (non-ambient light — from directional, point, and spot light sources) that the object reflects diffusely (dependent on the angle between the light source and the surface). As above, moving the slider to the right makes the teapot appear brighter.

NOTE *You can also control the amount and type of light that actually exists in a scene. This is independent of each object's ability to reflect the various types of light. We'll adjust lighting in the next session.*

Three sliders control the appearance of specular highlights on an object. Such highlights depend on the angles determined by a light source, the surface, and the eye.

□ Move the **specular** slider to the right to make the specular highlight on the teapot brighter.

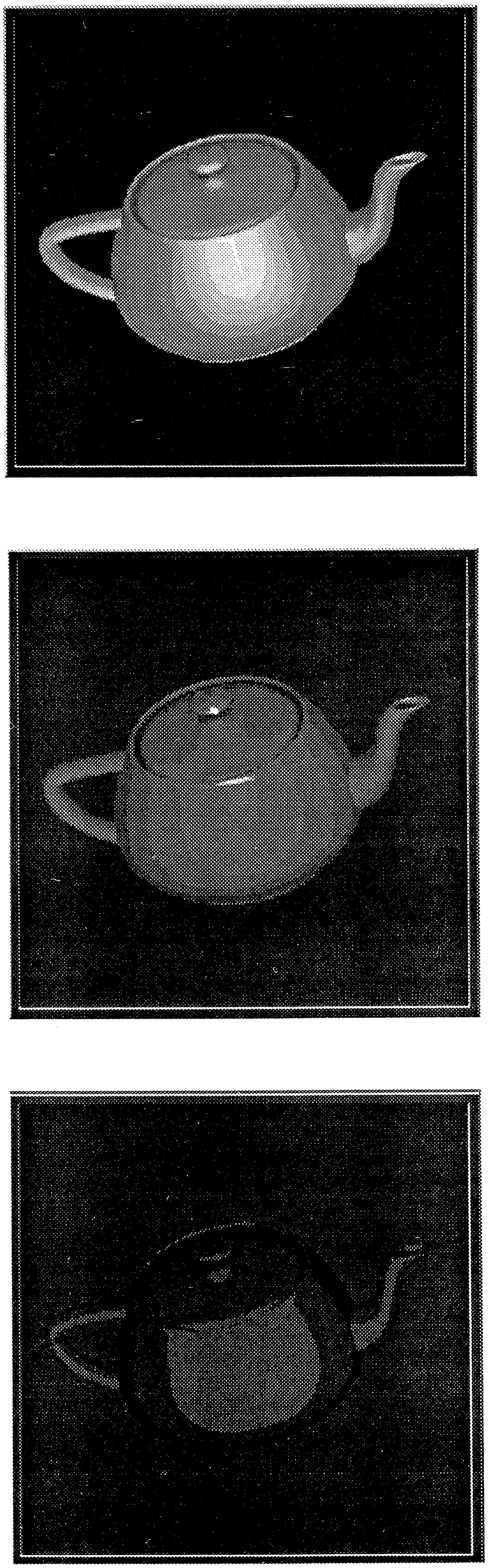
□ Move the **gloss** (specular sharpness) slider to the right to increase the glossiness of the highlight. This reduces its size.

□ AVS constrains the color of specular highlights — move the **metal** (specular color) slider to vary the color between that of the light source (to the left) and that of the object (to the right). In our default-lighting situation, the color of the light that produces the specular highlight is white.

A final slider, **trans**, controls the transparency of the object. Moving this slider to the left causes the teapot to fade away. Note that as it fades, you can see the back of the teapot through its increasingly transparent surface.

The best way to get the feeling for surface properties is to play with all these sliders. See how closely you can match the various teapot images shown on the next page.

Figure B-9. Teapots with Various Material Properties

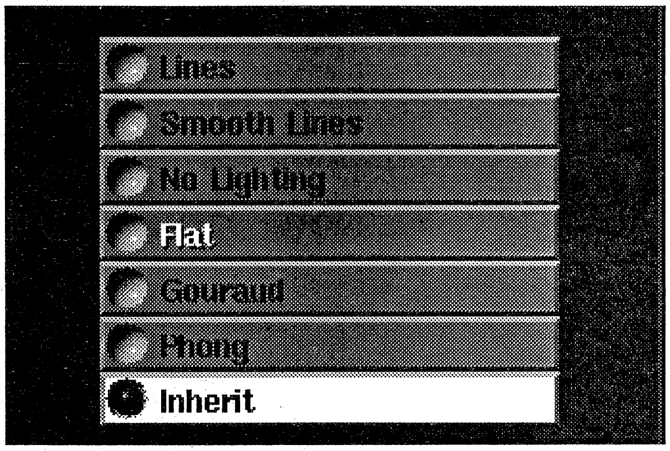


At any time, you can close the Edit Property window by clicking on **Close** at the bottom of the window.

Changing the Rendering Method

Besides being able to control the surface properties of objects, you can control the *rendering method* that AVS uses to create images. Different objects can be rendered using different methods.

Figure B-10. Rendering Methods



Click on each of the rendering-method choices, and note the effects on the teapot image. The most realistic method offered by AVS is *Phong shading*. In combination with specular highlighting, it provides excellent image fidelity.

The following paragraphs describe the differences among the rendering methods. Several of the methods address the way in which the facets of a solid object are colored. A typical object is defined as a collection of tens, hundreds, or even thousands of facets (usually triangles).

Lines

A "wireframe" image, consisting of vertices and lines connecting them.

Smooth Lines

A wireframe image in which the lines connecting the vertices are antialiases (smoothed). The anti-aliasing process requires extra computation; for complex scenes, there may be a slight performance cost with this method vis-a-vis **Lines**.

No Lighting

A solid image in which only the color you assign to the object is used. Any lights in the scene are ignored; no shading computations are performed.

If an object has colors assigned to each of its vertices, these colors are linearly interpolated across the facets of the object. For example, a finite element analysis model might represent stress values as colors for the various points in a support beam.

Flat

A solid image in which each facet of a polygon is drawn with a single, constant color. This depends on the object's color, along with the color and angle of the light(s) shining on the facet. Different facets can have different colors.

Gouraud

A solid image in which each facet of a polygon is shaded (different pixels are drawn with different colors) using the *Gouraud shading* algorithm. In this algorithm, a color is computed for each vertex that defines a facet. The computation takes into account the color assigned to the vertex (or to the entire facet), the "vertex normal" vector, and the available light(s). When color values have been computed for all vertices, the values are linearly interpolated to produce color values for the pixels in the interior of the facet.

Phong

A solid image in which each facet of a polygon is shaded (different pixels are drawn with different colors) using the *Phong shading* algorithm. This algorithm interpolates the vertex normals, rather than the computed vertex colors. A separate lighting calculation then takes place for each pixel in the facet, producing greater accuracy at the expense of more computation.

Inherit

Causes the rendering method of the object's parent to be used.

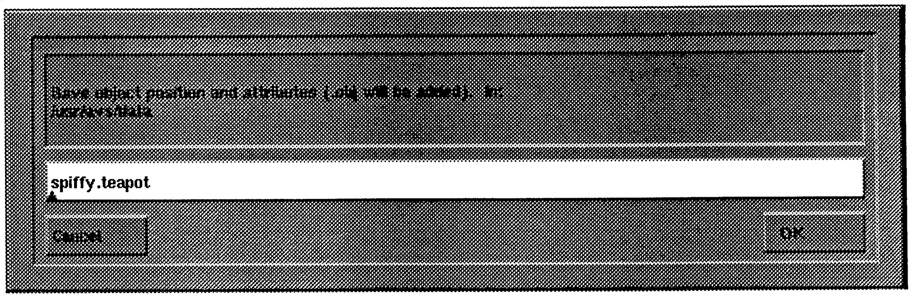
In some cases, you may notice performance differences among the surface-rendering methods. The following table suggests why these differences exist:

Rendering Method	**Lighting Calculations**
No lighting	none
Flat	one per facet
Gouraud	one per vertex
Phong	one per pixel

Saving Your Work

Continue experimenting with the Edit Property window and the rendering methods choices until you have a teapot you are proud of. Then, select **Save Object** under the Object menu selection. Type in a filename for your designer teapot in the popup window that appears. Be sure that the mouse cursor is in the filename entry area before you start typing. Finish by pressing **RETURN** or by clicking the **OK** box in the popup window. AVS automatically adds a *.obj* filename extension to the name you enter (unless you type the extension yourself).

Figure B-11. Saving an Object



Save object position and attributes (.obj will be added), in:
/usr/avs/data

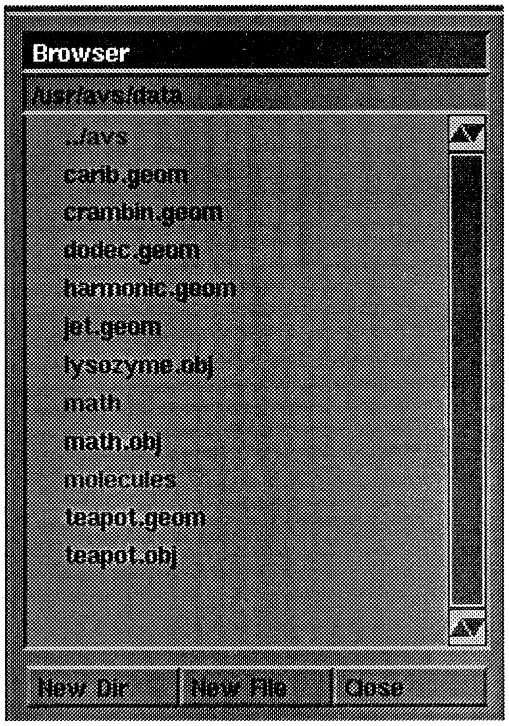
spiffy.teapot

To verify that your object was saved correctly, follow these steps:

❑ Select **Delete Object** to remove the teapot from the scene.

❑ Select **Read Object** to bring up the File Browser, and click on the filename you just entered. You may need to scroll the filenames. Scrolling works the same way in AVS as it does in the **xterm**(1) terminal emulator program: — the right mouse buttons scrolls toward the end of the list; the left mouse buttons scrolls toward the beginning of the list. Scrolling proceeds page-at-a-time unless the mouse cursor is in the little box at the top (or bottom) of the scroll bar. In that case, scrolling proceeds line-at-a-time.

Figure B-12. Scrolling through the File Browser



The teapot should reappear, exactly as you saved it.

Exiting the Geometry Viewer

That's enough for one session with the Geometry Viewer. To exit the program, select **Exit** at the very top of the control panel. To exit AVS, select **Exit AVS**, then click **OK** in the pop-up window that appears.

Session 2: Working with Lights

In the preceding session, we worked with a single object, changing its position, its surface properties, and the rendering method used to draw it. In this session, we'll leave the object alone and concentrate on another aspect of the scene — the lighting.

First, restart AVS:

```
avs
```

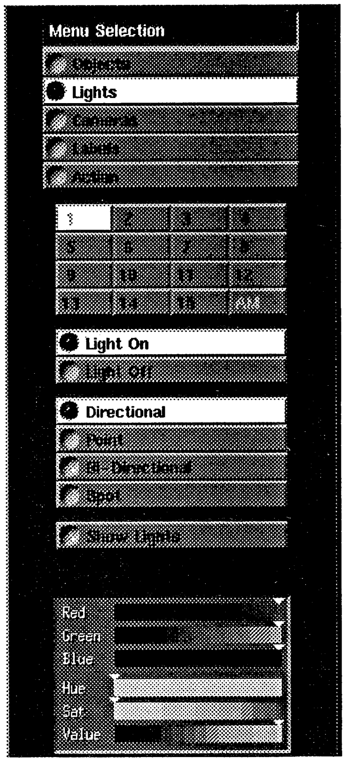
Once again, the **Object** menu selection is chosen automatically. Select **Read Object**, and click on the customized teapot object that you created in the preceding session. (In our example, it was called *spiffy.teapot.obj*.) After you've read in the object, close the File Browser window.

To get a feeling for the default lighting, rotate the teapot again. (Use the middle mouse button.) It's clear from the specular highlights that there is a light source located directly in front of the screen. Now, let's work with that light source.

The Lighting Panel

Select the **Lights** menu selection in the control panel. The various object-related menu choices disappear, and are replaced by lighting-related choices:

Figure B-13. The Lights Menu Selections



The bank of numbers is a *lighting panel*, which allows you to create and control up to 16 lights. Each of the first 15 lights can be any of the following types:

❑ Directional

❑ Point

❑ Bi-Directional

❑ Spot

The last one, **AM**, is the ambient (non-directional) light for the scene.

Note that the **1** and the **AM** are "lit up", indicating that light #1 and the ambient light are currently turned on. Below the lighting panel are control/indicator buttons, which you can use to turn particular lights on and off, and to change the type of each light. Click on the various buttons in the lighting panel to verify that all lights (except #16) are initially directional, and all but light #1 are initially turned off.

At this point, make sure you also select **Transform Light** in the Transform Selection part of the control panel. This causes the mouse to be "attached" to the lights.

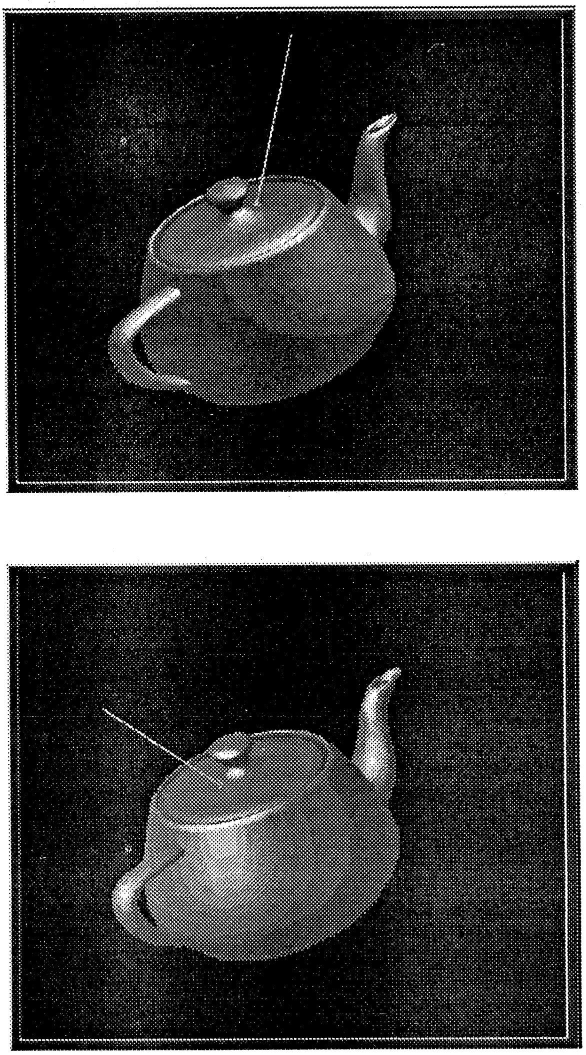
NOTE *As you become more familiar with the Geometry Viewer, you may find it convenient to "mix-n-match". For instance, you might choose the Object menu selection in order to read some new objects, but at the same time choose the* **Transform Light** *selection in order to move the lights around the new objects. At the* *beginning, however, we recommend that that you keep the menu selection and the transform selection synchronized.*

Manipulating Lights

Just as you have used the middle mouse button to rotate objects, you can now use this button to rotate the directional light. Give it a try. The specular highlights on the teapot give a clear indication that the light direction is, indeed, moving.

To help you work with lighting, there is a **Show Lights** menu selection. Make this selection, and an arrow appears in the scene, indicating the direction of the light. As you rotate the light, this arrow changes its angle accordingly.

Figure B-14. Using 'Show Lights'



The right mouse button controls translation of the light source parallel to the plane of the display screen, just as it did for objects. (And along with the **Shift** key, it can also control translation perpendicular to the screen.) If you try this, however, you won't see any change it the lighting. That's because translating a *directional* light source makes no difference.

To test the translation facility, change the light source from *directional* to *point* by clicking the **Point** menu selection. (A point light source, such as a light bulb, has a specific position.) The appearance of the teapot changes slightly, showing the difference between point lighting and directional lighting. And now, you can use the right mouse (with and without **Shift**) to move the location of the point light. As before, the middle mouse button rotates the light.

Note that for point lights, **Show Lights** causes a location marker to appear, rather than a directional arrow. This marker is depth-cued to show its location along the axis perpendicular to the screen.

Be sure to test the **Reset** button with the light. As with objects, it returns the light to its original position (and direction).

Creating Additional Lights

So far, we've been working with just one light. Let's create another one. Click 2 on the lighting panel, then turn the light on by clicking **Light On**. Assuming that you still have **Show Lights** selected, another arrow appears in the scene.

As you switch back and forth between Light #1 and Light #2 on the lighting panel, note that the label above the Current Object Indicator also changes, indicating which light is currently under mouse control. As long as **Transform Light** is the transform selection, the mouse will be "attached" to the currently-selected light on the lighting panel.

Try moving and translating both lights, making sure that you can see the effect of each light on the teapot. To facilitate differentiating the lights, try changing the color of one (or both) of them. At the bottom of the control panel is a miniature set of color controls, just like the ones in the Edit Property window for objects. Moving these sliders controls the color of the currently selected light.

Additional Light Types

AVS also supports *bi-directional* lights and *spot* lights. Try making light #2 bi-directional. Note that light #10 is turned on automatically. That's because this type of light is actually two directional lights, pointing in opposite directions. When **Show Lights** is selected, a bi-directional light is represented by two arrows.

A spot light is similar to a point light, but has a restricted "cone" of influence. (If a point light is like a naked light bulb, a spot light is like a well-focused flashlight.) When **Show Lights** is selected, the symbol for a spot light shows the cone. The angle of this cone is fixed in AVS — you cannot change it.

Be sure you also work with the ambient light (**AM** on the lighting panel). Using the mouse won't make any difference, since ambient light has no position or direction. But you can change the brightness and the color of this light using the sliders.

That's all for this session. Note that there is no "Save" menu selection under **Lights**. There is no way to save a particular lighting configuration independently from the scene in which they exist. In the next session, we'll save and retrieve an entire scene.

To end the Geometry Viewer session, select **Exit** at the top of the control panel. To exit AVS, select **Exit AVS**, then click **OK** to verify your choice.

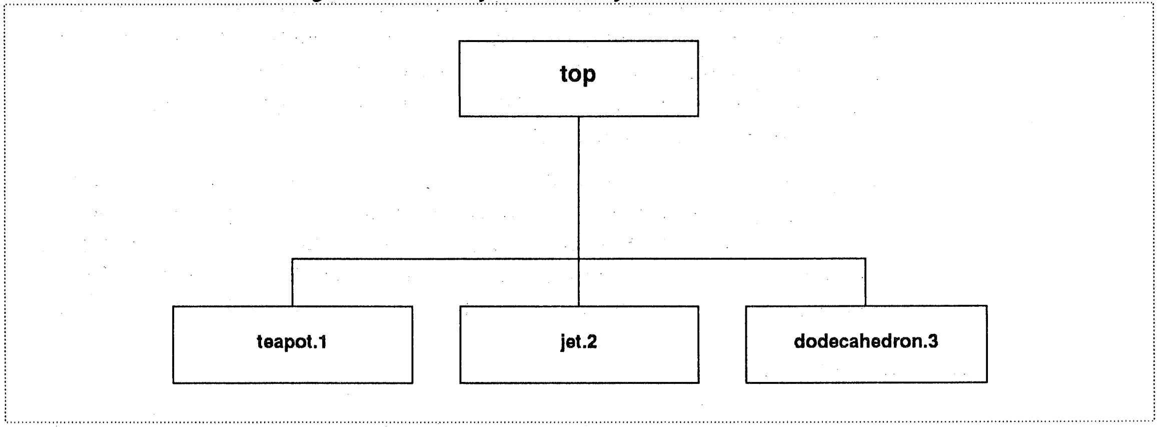
Session 3: Creating Composite Objects

So far, we have worked with a single object, a teapot, changing its properties and changing the way it is lighted. In this session, we explore one of AVS's more sophisticated capabilities — maintaining hierarchies of objects and attributes.

The Geometry Viewer allows you to define complex object hierarchies. For instance, a model of a building might feature an office unit, which contains rooms, which contain desks, which have built-in clocks, which have hands.

The viewing application itself does not create multi-level hierarchies in ordinary, interactive usage. Each *scene* you create consists of a top-level object, which is initially empty. Each time you bring an object into the scene using **Read Object**, the new object becomes a direct descendant of the top-level object:

Figure B-15. Object Hierarchy



NOTE *Actually, AVS is capable of handling deeper hierarchies. Some of the Stardent-supplied objects are themselves hierarchical. In addition, you can use the AVS Script Language to define multi-level object hierarchies. This language is described in a later chapter of this book.*

Using the Edit Property window and the rendering-method selections (as we did in Session 1), you can set the attributes of the top-level object, or of any of its children. If a child has its attributes set, they override the settings of its parent, the top-level object. Alternatively, you can specify that the child is to *inherit* the attributes of its parent. You can also save the current state of the Edit Property window in a *.prop* file, then retrieve the attributes later, applying them to the same object (or a different one).

Changing the Current Object

Let's see how it all works. Start the AVS program and then start the Geometry Viewer subsystem, select **Read Object**, and (once again) click on the customized teapot object that you created in the first session. This time, however, don't close the File Browser. Instead, select another object to read into the scene: *dodec.geom*. This is a dodecahedron (a twelve-faced platonic solid whose principal occurrence in nature is in the form of calendar paperweights).

The simplest way to explore the object hierarchy is to rotate and translate objects. First, make sure that **Transform Object** is the current Transform Selection. Now, click the left mouse button at various locations in the window. Watch the current object indicator in the Transform Selection area as you do so:

❑ Clicking on either the teapot or the dodecahedron makes it the current object.

❑ Clicking on the background makes the top-level object current — this includes both the teapot *and* the dodecahedron.

Note that the Current Object Indicator always shows the currently selected object. In addition, the label above this indicator shows you where you are in the hierarchy:

`top`	*top-level object is current*
`teapot.1`	*teapot is current*
`dodec.3`	*dodecahedron is current*

Verify that a particular object is selected by moving it with the right mouse button, or rotating it with the middle mouse button. When the top-level object is selected, the teapot-dodecahedron pair move as a unit.

There are a couple of additional ways to navigate the object hierarchy:

❑ Clicking on the same object multiple times cycles you through the hierarchy — the first click selects the object itself (the bottom level); the second moves up one level to the parent (in many cases, this is the top-level object); and so on. Continuing to click after you've reached the top level returns you to the individual object level.

❑ Clicking on the current object indicator window cycles through all the single and composite objects in the scene.

Try these methods, too, and verify them by moving and rotating the object you select.

Hierarchy and Property Editing

Now, let's explore the way in which you can use hierarchy in editing object attributes. Make the teapot the current object. Then select **Edit Property** under the Object menu selection. The sliders in the Edit Property window reflect the "customization" you performed on the teapot in the first session. Suppose you want to these same settings to apply to both the teapot and the dodecahedron. There are two ways to accomplish this:

❑ Apply the teapot properties directly to the dodecahedron.

❑ Apply the teapot properties to the top-level object, then have both the teapot and the dodecahedron *inherit* these properties from the top-level object.

Let's try both methods.

Saving and Retrieving Properties

Click **Save** at the bottom of the Edit Property window. Then enter a filename under which to save the properties. AVS automatically adds a *.prop* extension to the name you type (unless you type the extension yourself).

Now make the dodecahedron the current object by clicking on it with the left mouse button. The sliders in the Edit Property window change to indicate the current settings for the dodecahedron. Click on **Read** at the bottom of this window, bringing up the File Browser. Then click on the name of the *.prop* file you just created. This copies the saved properties onto the dodecahedron.

The Geometry Viewer subsystem doesn't try to keep the properties of the two objects synchronized. Both objects have the same color now, but if you change the color of the teapot, the dodecahedron *won't* change too.

Inheritance of Properties

Applying the properties to the top-level object *will* keep the teapot and dodecahedron synchronized. Make the teapot the current object, then click **Inherit** at the bottom of the Edit Property window. Do the same for the dodecahedron.

Now select the top-level object (e.g. by clicking with the left mouse button in the background of the window). Click on **Read** at the bottom of this window and, as before, click on the name of the *.prop* file you created. These saved properties now pertain to the top-level object and, through inheritance, to both the teapot and the dodecahedron. If you change the color of the top-level object, it will affect both objects at the lower level.

That's all for this session. Exit from the Geometry Viewer and AVS as usual.

Session 4: Creating Multiple Views

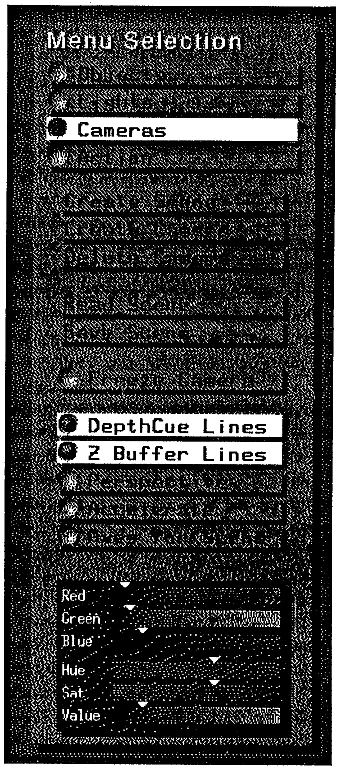
In this session, we explore AVS's ability to show the same scene from different points of view. Continuing the use of the movie studio metaphor, each view is associated with its own *camera*. Each camera's view appears in a separate window, which you can move and resize using any X Window System window manager. (The Stardent-standard window manager is **uwm**(1).) You can change the position of cameras in much the same way as you change the position of objects and lights. You can also adjust the "camera lens" (this is stretching the metaphor a bit): accessing such Stardent features as depth-cueing and Z-buffering.

Collectively, all the views on a particular collection of objects, along with a set of lights, is called a *scene*. (Thus far, we've been using the term *scene* more loosely, referring to a group of objects.) We'll also see in this session how the Geometry Viewer lets you work with multiple scenes concurrently.

First, recreate the situation of the previous lesson: start AVS and the Geometry Viewer and read in both the teapot and dodecahedron objects.

Now, bring up the **Camera** menu selections in the control panel:

Figure B-16. The Camera Menu Selections



Creating and Resizing a Window

Select **Create Camera** to create a new window that shows the same scene — that is, the same pair of objects. You may want to use the window manager to rearrange and/or resize the windows more to your liking.

The initial position of the new camera is the same as the initial position of the original camera, so both views may be the same. The two cameras are independent, however. To see how this works, click **Transform Camera** in the Transform Selection area. Now use the right mouse button (with and without **Shift**) to change the position of the newly created camera. Also try using the middle mouse button to rotate the new camera.

It's easy to move the camera to the other window. As soon as you click (e.g. with the middle mouse button) in the other window, the mouse becomes "attached" to that window's camera.

Relating Cameras to Lights and Objects

To gain a better understanding of how cameras work, try moving the light and the objects in either window. Remember to change the Transform Selection to **Transform Light** or **Transform Object** before you try to manipulate them with the mouse.

You'll notice that as you manipulate the light(s) in one window, the lighting also changes in the other window. That's because, conceptually, a single scene with a single set of lights is being viewed in both windows.

Similarly, when you move an object in one window, it also moves in the other — there is one set of objects, being viewed in two different ways.

Adjusting the 'Camera Lens'

Several of the menu selections under **Camera** allow you to "adjust the lens" of the current camera. These adjustments provide access to several features of the underlying graphics software:

- ❑ You can switch back and forth between an *orthogonal* view (the default) and a *perspective* view. Moving the camera or an object perpendicular to the screen produces quite different effects, depending on which type of view is currently selected.
- ❑ You can turn on and off the Z-buffering of lines and the depth-cueing of lines and spheres.
- ❑ You can turn on and off the display of the X-Y-Z coordinate axes. This can help you keep track of the orientation of the object(s) as you work with one or more camera views.

Try each of these adjustments by clicking the menu choices. Keep in mind that each selection is applied to the current camera (window). You may need to click in a particular window (with any mouse button) before making the **Camera** menu choice.

The Accelerate Feature

The Geometry Viewer includes a performance acceleration feature that is quite useful when your scene includes a large number of objects, or a few very complex objects. When you select **Accelerate**, only the current object is rendered repeatedly as you move it around the screen. All other objects and the window background are saved in a *virtual pixel map*. Parts of this pixel map are copied into display memory (the frame buffer) as needed while you perform object manipulations. Since copying pixels is faster than rendering, the result is a performance acceleration.

Freezing a View

There may be times when the view provided by one of the camera is "just right". You can temporarily freeze the image with the **Freeze Camera** menu selection. You can still manipulate the lights and/or objects in the scene, but the changes won't be shown in the frozen window until you unfreeze it by clicking (with any mouse button) in that window.

Saving an Entire Scene

The **Save Scene** menu selection causes the entire current scene to be saved in a *.scene* file. You supply a filename, just as you do when saving objects and Edit Property attributes. The saved scene includes all the objects and their attributes, all the lights, and all the cameras/windows that you've created for the scene.

To test this, first save the scene with **Save Scene**, then delete each of its windows with **Delete Camera**. Now, select **Read Scene** and click on your scene's name in the File Browser. All the windows defined for the scene return to the screen.

Creating a New Scene

The Geometry Viewer lets you create as many windows as you want on the current scene, using **Create Camera**. You can also create an entirely new scene with **Create Scene**. The new scene is an empty window, into which you can read one or more objects, adjust the lighting, and create additional cameras, just as you did with the original scene.

If you have two or more scenes onscreen at the same time, be sure you know which is the current window when you use **Create Camera.** This function creates a new camera for the scene to which the current window belongs. To get what you want, you may need to click in a particular window (using any mouse button) before selecting **Create Camera.**

Appendix C. Geometry Conversion Programs

The AVS Geometry Viewer reads and writes graphical objects in a data format called *geom*. This format is described in **geom**(3V). The Application Visualization System includes a filter facility, which enables conversion of data in other formats to the *geom* format. The filter facility has several usage levels:

- ❑ Stellar supplies a set of *filter utilities*. The AVS viewing application sometimes invokes one of these automatically when the user executes the **Read Object** function. Each filter converts data in a particular format to the *geom* format.

  During a **Read Object**, if AVS determines (using the filename extension) that the file you select is in a known data format, it invokes the appropriate filter automatically to create a corresponding *geom* file on disk. It then reads in the object from the *geom* file.

- ❑ You can also execute the filter utilities from the shell, outside the AVS viewing application.

- ❑ A set of *source code templates* for creating new filter utilities. Each template handles the conversion of a particular kind of object to the *geom* format. The templates are written in C and in FORTRAN.

  For simple data, you may be able to use one of these templates directly. For more complex data, use one of the templates as a starting point for creating a customized filter utility.

- ❑ The GEOM library, consisting of the Stellar-defined object types and C-language procedures to be used in writing new filter utilities. This library is described in **geom**(3V).

Automatic Data Filtering

When you execute the AVS function **Read Object**, you select a filename in the File Browser window. If the file has a *.geom* 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 filters directory for utility programs named ...*_to_geom*. Each such filter determines a particular filename extension. For instance, the filter utility *pdb_to_geom* determines the extension *.pdb*.

- ❑ It compares the extension of the **Read Object** filename with its list of filter extensions.

- ❑ If there is a match, AVS automatically invokes the appropriate filter utility, creating a *geom*-format file of the same name, with a *.geom* extension.

- ❑ The **Read Object** function is completed by reading in the newly created *.geom* file.

The table below lists the filter utilities that Stellar supplies in directory */usr/avs/bin*. (You can specify an alternative filters directory when you start AVS, by using the **–filter** *dirname* option.)

TABLE C-1. Stellar-Supplied Filter Utilities

Filter Name	Extension	Data Format
ts_to_geom	.ts	Mathematica ThreeScript
wfront_to_geom	.wfront	Wavefront
byu_to_geom	.byu	Movie BYU
pdb_to_geom	.pdb	Protein data bank
ppoly_to_geom	.ppoly	UNC
pobj_to_geom	.pobj	Stellar internal format
objs_to_geom	.objs	Stellar internal format

Shell-Level Usage of Filter Utilities

Each of the filters listed in the table above can be invoked from the shell. 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 filters don't accept any command line options. The exceptions are described in the following sections.

ts_to_geom Filter Options

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.

pdb_to_geom Filter Options

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

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

Postprocessor Filters

Stellar supplies an additional set of filters, which 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 filters supplied by Stellar. Except for **send_to** and **animate_to**, each filter reads a file from *stdin* and writes a file to *stdout*.

TABLE C-2. 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 object's normals to unit length (normalizes them).
geom_to_normals [–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.
geom_split [*name* **]**	Creates a series of *.geom* files, each of which contains one of the objects defined in the input file. This is appropriate only for input files that define complex objects (e.g. a polyhedron, which consists of several polygons). 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 causes a currently-executing AVS program to read the script file. The cycle includes the specified *geom-file* sequence.

Templates for New Filter Utilities

Several C-language and FORTRAN-language templates are provided for those who wish to write their own filter utilities. (If your data format is simple enough, you may be able to use one of the templates without modifying it. The mesh format, in particular, can often be used without modification.)

Each template handles a particular type of object defined in the GEOM library. The table below lists the Stellar-supplied filter templates. Each one reads a file form *stdin*, writes a file to *stdout*, and accepts no command-line options.

TABLE C-3. Templates for Filter Utilities

Source Filename(s)	Filename of Executable	Object Type
mesh.c, mesh.f	mesh_to_geom	Mesh
polygon.c, polygon.f	polygon_to_geom	Disjoint polygon
polyh.c	polyh_to_geom	Polyhedron
sphere.c	sphere_to_geom	Sphere (from **xmole** demo)

The filters are all located in directory */usr/avs/filter*. The file formats used by the filters are described in the *AVS Programmer's Guide*.

Writing a New Filter Utility

This section provides pointers for those who wish to create new filter utilities, using the template programs listed in the table above.

The basic procedure for creating a *geom*-format object is:

1. Decide which of the *geom*-format objects conforms most closely to the application data:

   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.

   Note that no tools exist for direct conversion of non-linear geometries, such as spline surfaces and quadrics.

2. Create an instance of that *geom*-format.
3. Perform any necessary processing on the object, such as generating normals.
4. If necessary, convert this object to an optimized-format object, such as a polytriangle.
5. Write the object to a file.

The GEOM library contains routines that help with these tasks. For documentation of these routines, see **geom**(3V). The sections below describe the steps for converting a variety of object types to *geom* format.

Converting a Polyhedron

Start with the template *polyh.c*, and

- ❑ Create a polyhedron object.
- ❑ Add vertices.
- ❑ Add a list of polygons (as a list of pointers).
- ❑ Generate normals (if necessary).
- ❑ Convert to polytriangle object — both wireframe and surface descriptions.

Converting a Polygon

Start with the template *polygon.c* or *polygon.f*, and

- ❑ Create a polyhedron object.
- ❑ Add disjoint polygons (either faceted or smooth).
- ❑ Generate normals (if necessary).
- ❑ Convert to polytriangle object — both wireframe and surface descriptions.

Converting a Scalar Mesh

Start with the template *mesh.c* or *mesh.f*, and:

- ❑ Create a mesh from a list of scalars.
- ❑ Generate normals (if necessary).
- ❑ Convert to polytriangle object — both wireframe and surface descriptions.

Converting a Mesh

Start with the template *mesh.c* or *mesh.f*, and:

- ❑ Create a mesh from the vertices.
- ❑ Generate normals (if necessary).
- ❑ Convert to polytriangle object — both wireframe and surface descriptions.

Converting a Sphere

Start with the template *sphere.c*, and:

- ❑ Create a sphere object from the sphere centers and radii.

Converting a Disjoint Line

There is no starting template for this case. You should:

- ❑ Create a polytriangle object.
- ❑ Add disjoint lines to this object.

Converting a Polyline

There is no starting template for this case. You should:

- ❑ Create polytriangle object.
- ❑ Add zero or more polylines to this object.

Appendix D.
The Geometry Viewer Script Language

The *AVS 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).

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 viewing application itself creates a file using the Script Language whenever you use the **Save Object** or **Save Scene** function. It is often useful to create a file in this way, then revise it later using a text editor.

NOTE *A programmer can access the script language's functionality through the OBJ object library (see* **obj***(3V)).*

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 D-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 transform 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
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 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:

Figure D-1. The group Command

```
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:

Figure D-2. The cycle Command

```
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 widthxheight+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** module. 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:

Figure D-3. 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 will start 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

Figure D-4. 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
   }
}
```

Appendix E. AVS File Formats

This appendix summarizes the various data file formats used by the Application Visualization System.

AVS Startup File

When it begins execution, AVS searches for a *startup file*, which specifies the locations of various directories. AVS looks for the following files, in the order listed:

./.avsrc	(current directory)
$HOME/.avsrc	(home directory)
/usr/avs/runtime/avsrc	(system directory)

At most *one* of these startup files is read. If AVS finds one of them, it ignores the others. A */usr/avs/runtime/avsrc* file is included on Stardent's AVS distribution tape.

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

```
NetworkWindow       867x567+407+2
NetworkDirectory    /usr/johnp/avs/nets
DataDirectory       /usr/johnp/avs/data
DialDevice          /dev/tty02
```

In most cases, 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.

The AVS startup file keywords are as follows:

DataDirectory
(command-line equivalent: **–data**) Specifies the directory in which the various "read data" modules (**read field, read geometry**, etc.) initially will look for data files.

DialDevice
(command-line equivalent: none) Indicates the serial communications port to which a DIGIT dialbox device is attached.

This entry corresponds to the environment variable DIALS; if DIALS is set, the startup file entry (if any) is ignored.

ImageScrollbars
(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.)

NetworkDirectory
(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

(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.

Path

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

SaveMessageLog

(command-line equivalent: none) It 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).

SpaceballDevice

(command-line equivalent: none) Indicates the serial communications port to which a Spaceball device is attached.

This entry corresponds to the environment variable SPACEBALL; if SPACEBALL is set, the startup file entry (if any) is ignored.

Module Library File Format

This section describes the format of a *module library file*, as you can create it with any text editor. AVS itself uses a somewhat more complex file format for the module library file it creates when you select the **Write Module Library** function. The more complex format allows the Network Editor to load the module library more quickly.

ID Line

The module library file begins with this line:

```
# AVS Module Library
```

Other lines that begin with # may precede this line.

Command Lines

Each subsequent line must be in one of these forms:

builtin	*module_name*
file	*filename*
directory	*dirname*

The **builtin** keyword specifies a module that is built into AVS itself, rather than being implemented as a separate executable file. The following modules are builtins:

- colormap manager
- generate colormap
- image manager
- volume manager
- orthogonal slicer
- display image
- display pixmap
- render geometry
- render manager
- transform pixmap
- write image
- write volume

The **file** keyword specifies an executable file that includes one or more module definitions. See the *AVS Developer's Guide* for more on creating modules.

The **directory** keyword specifies a directory that includes executable module definition files.

Image File Format — .x

An image 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:

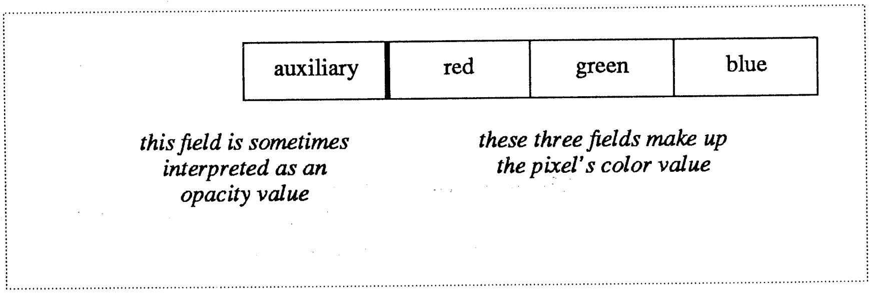
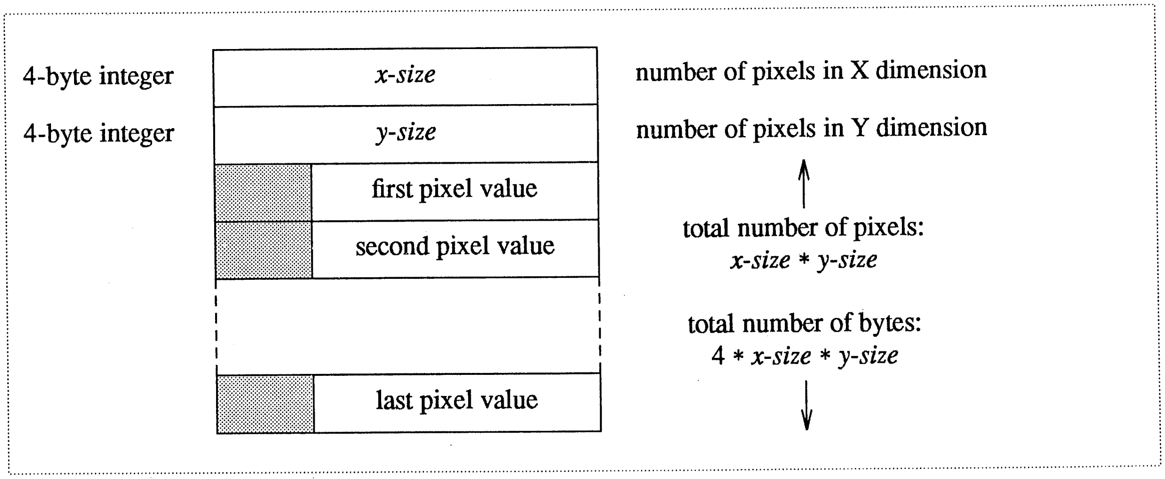

auxiliary
red
green
blue
this field is sometimes interpreted as an opacity value
these three fields make up the pixel's color value

Image data files should have names that end with a *.x* suffix. The image file format can be pictured as follows:

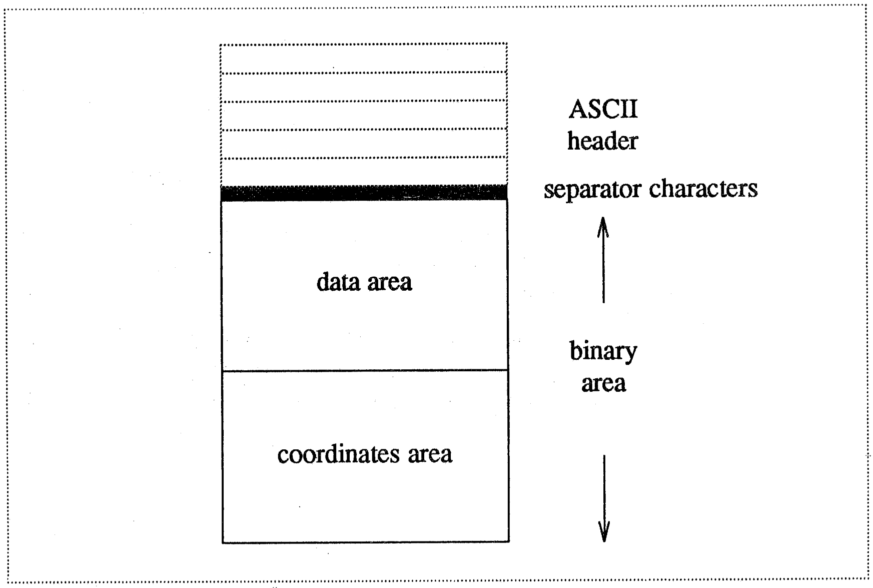


Field File Format — .fld

A *field* is a generalized array structure. Whereas each element of an ordinary 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.

Field data files should have names that end with a *.fld* suffix. The field file format can be pictured as follows:



ASCII Header. Every field file must begin with a header in ASCII text format. This header includes a number of "keyword=value'' pairs, one per line; it also may include comment lines. For example:

Figure E-1. ASCII Header for AVS Field

```
# 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 keywords are described below, with reference to the three examples in the preceding sections.

ndim

This value specifies the number of *computational* dimensions in the field. That is, it specifies the number of dimensions in the field element array. In all the examples above, **ndim** has the value 2.

dim1, dim2, ...

The values for these keywords specify the size of the computational space. For a 2D field, you would specify **dim1** and **dim2** values. In all the examples above, **dim1** = 2 and **dim2** = 5. For a 4D field, you would specify **dim1**, **dim2**, **dim3**, and **dim4** values.

nspace

This value specifies the dimensionality of the physical space that corresponds to the computational space.

veclen

This value specifies the number of data values for each field element.

data

This keyword takes one of the following values: **byte, integer, real, double.** It indicates the type of data that is supplied for each field element. AVS fields have the restriction that all of the **veclen** data values must be of the same type.

field

This keyword takes one of the following values: **uniform, rectilinear, irregular.** The main purpose of three examples above is to illuminate these three field types. 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.

Separator Characters. The ASCII header must be followed by two *formfeed* characters, in order to separate it from the binary area. A formfeed is expressed variously as **Ctrl-L**, octal 14, decimal 12, or hex 0C.

This scheme allows you to 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 consists of all the data that is associated with the field elements, along with all the coordinates. (For uniform fields, the coordinates area is null.)

The **data area** begins with the one or more data values for the first field element. All the data values for a field element are stored together. The first array index varies most quickly ("FORTRAN-style"). For example, suppose the ASCII header is as follows:

```
ndim = 3
dim1 = 10
dim2 = 5
dim3 = 8
nspace=3
veclen=5
data=byte
field=uniform
```

The data ordering can be pictured as follows:

DATA(1,1,1)	value 1
DATA(1,1,1)	value 2
DATA(1,1,1)	value 3
DATA(1,1,1)	value 4
DATA(1,1,1)	value 5
DATA(2,1,1)	value 1
DATA(2,1,1)	value 2
DATA(2,1,1)	value 3
DATA(2,1,1)	value 4
DATA(2,1,1)	value 5

<- each value is one byte

DATA(10,5,8)	value 1
DATA(10,5,8)	value 2
DATA(10,5,8)	value 3
DATA(10,5,8)	value 4
DATA(10,5,8)	value 5

Volume Data File Format — .dat

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*, but 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.

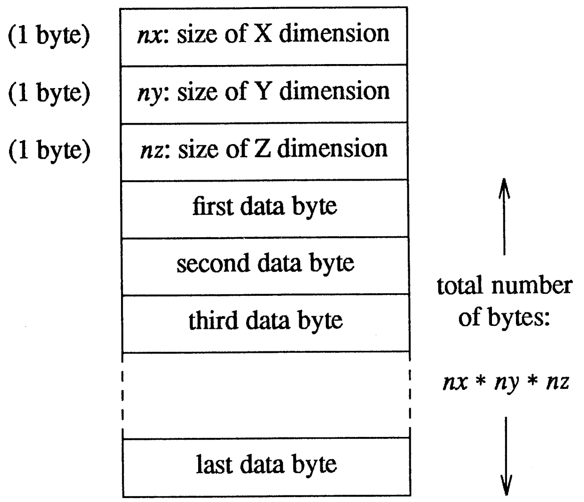
NOTE *The **volume data** and **field** file formats are not compatible.*

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 can be pictured as follows:



Geometry File Format — .geom

AVS geometric descriptions are created and written to disk files with calls to the *libgeom.a* programming library. The **geom**(3V) manual page presents a complete description of this library. It is reproduced in the following pages.

The **geom**(3V) manual page can be viewed from the shell with the **man** command:

```
man 3V geom
```

Within AVS, this manual page is available in the Geometry Viewer help browser under the name *geom_library.txt*.

geom(3V)

Application Visualization System geometry database

OVERVIEW

The *geom* package is a geometry-processing library. It reads and writes 3D geometric files, provides routines to manipulate geometry such as normal generation, and optimizes geometry for the Stellar architecture. An application can use this library to define new filters to convert data into *geom* format.

This manual page is organized as follows:

SYNOPSIS section:

- ❑ Compiling and linking information
- ❑ Alphabetical list of *geom* routines, showing their parameters

DESCRIPTION section:

- ❑ An overview of how *geom* routines should be used

ROUTINES section:

Descriptions of *geom* routines, grouped under the appropriate topics:

- ❑ Object Creation Routines
- ❑ Object Utility Routines
- ❑ Object Property Routines
- ❑ Object Texture-Mapping Routines
- ❑ Object File Utilities
- ❑ AVS Module Interface Routines

FORTRAN BINDING section:

- ❑ A discussion of the FORTRAN calling sequences for *geom* routines

SYNOPSIS

A C language application that uses *geom* routines must use the following header file:

/usr/avs/include/geom.h

A FORTRAN application that uses *geom* routines must use the following header file:

/usr/avs/include/geom.inc

An application that uses *geom* routines must be linked with the following libraries, in this order:

/usr/avs/lib/libgeom.a
/usr/avs/lib/libutil.a
/usr/lib/libPW.a
/lib/libm.a

geom ROUTINE LISTING

GEOMadd_disjoint_line(*obj, verts, colors, n, alloc*)
GEOMadd_disjoint_polygon(*obj, verts, normals, colors, nverts, flag, alloc*)
GEOMadd_float_colors(*obj, colors, n, alloc*)
GEOMadd_int_colors(*obj, colors, n, alloc*)
GEOMadd_int_value(*obj, type, value*)
GEOMadd_label(*obj, text, ref_point, offset, height, color, label_flags*)
GEOMadd_normals(*obj, normals, n, alloc*)
GEOMadd_polygon(*obj, nverts, indices, flags, alloc*)
GEOMadd_polygons(*obj, plist, flags, alloc*)
GEOMadd_polyline(*obj, verts, colors, n, alloc*)
GEOMadd_polytriangle(*obj, verts, normals, colors, n, alloc*)

GEOMadd_polytriangle_uvs(*obj, uvs, i, n, alloc*)

GEOMadd_radii(*obj, radii, n, alloc*)

GEOMadd_uvs(*obj, uvs, n, alloc*)

GEOMadd_vertices(*obj, verts, n, alloc*)

GEOMadd_vertices_with_data(*obj, verts, normals, colors, n, alloc*)

GEOMauto_transform(*obj*)

GEOMauto_transform_list(*objs, n*)

GEOMauto_transform_non_uniform(*objs, n*)

GEOMauto_transform_non_uniform_list(*objs, n*)

GEOMcreate_label(*extent, label_flags*)

GEOMcreate_label_flags(*font_number, title, background, drop, align, stroke*)

GEOMcreate_mesh(*extent, verts, m, n, alloc*)

GEOMcreate_mesh_uvs(*obj, umin, vmin, umax, vmax*)

GEOMcreate_mesh_with_data(*extent, verts, normals, colors, m, n, alloc*)

GEOMcreate_normal_object(*obj,scale*)

GEOMcreate_obj(*type, extent*)

GEOMcreate_polyh(*extent, verts, n, plist, flags, alloc*)

GEOMcreate_polyh_with_data(*extent, verts, normals, colors, n, plist, flags, alloc*)

GEOMcreate_scalar_mesh(*xmin, xmax, ymin, ymax, mesh, colors, n, m, alloc*)

GEOMcreate_sphere(*extent, verts, radii, normals, colors, n, alloc*)

GEOMcvt_mesh_to_polytri(*tobj, flags*)

GEOMcvt_polyh_to_polytri(*tobj, flags*)

GEOMdestroy_edit_list(*list*)

GEOMdestroy_obj(*obj*)

GEOMdestroy_uvs(*obj*)

GEOMedit_color(*list, name, color*)

GEOMedit_concat_matrix(*list, name, matrix*)

GEOMedit_geometry(*list, name, obj*)

GEOMedit_group(*list, name, parent*)

GEOMedit_light(*list, name, type, status*)

GEOMedit_properties(*list, name, ambient, diffuse, specular, spec_exp, transparency,*
spec_col)

GEOMedit_render_mode(*list, name, mode*)

GEOMedit_set_matrix(*list, name, matrix*)

GEOMedit_visibility(*list, name, visibility*)

GEOMflip_normals(*obj*)

GEOMgen_normals(*obj, flags*)

GEOMinit_edit_list(*list*)

GEOMnormalize_normals(*obj*)

GEOMquery_int_value(*obj, type, value*)

GEOMread_obj(*fd, flags*)

GEOMset_color(*obj, color*)

GEOMset_computed_extent(*obj, extent*)

GEOMset_extent(*obj*)

GEOMset_object_group(*obj, name*)

GEOMset_pickable(*obj, pickable*)

GEOMunion_extents(*obj1, obj2*)

GEOMwrite_obj(*obj, fd, flags*)

GEOMwrite_text(*obj, fp, flags*)

DESCRIPTION

The *geom* library provides functions for reading and writing objects, creating objects from a variety of different data formats, processing objects (such as generating normals), and "compiling" databases into a format that is most efficient for the hardware to render.

The *geom* library contains both a C and a FORTRAN version of each routine. The main routine descriptions in this manual page discuss the C versions. For a discussion of the FORTRAN calling sequences, see the FORTRAN BINDING section of this manual page.

Many routines allow a NULL value for some arguments. In all such cases, the constant **GEOM_NULL** should be used to represent a NULL value.

The basic entity in the *geom* package is the *geom* object. Several types of objects are supported:

GEOM_LABEL
GEOM_MESH
GEOM_POLYHEDRON
GEOM_POLYTRI
GEOM_SPHERE

Each object can have colors or normals associated with each vertex in the object, but it need not have either.

Label A label contains one or more text strings normally used to label an AVS object or view. The label is presented as annotation text: its position can be transformed, but the text always appears upright in a plane parallel to the display surface.

Mesh A mesh object contains one two-dimensional array of vertices.

Polyhedron A polyhedron contains a list of vertices and a list of polygons, which are defined by indirectly referencing the vertices that the polygon contains.

Polytriangle A polytriangle object contains a list of polytriangle strips, a list of polylines, or a list of disjoint lines. When a single object has both line and surface data, the line data is assumed to be an alternate wireframe description for the same geometry (only one should be displayed at a given time).

Sphere A sphere object contains a list of points and a corresponding list of radii.

OBJECT CREATION ROUTINES

Many routines can be used to create an object. The goal is to provide entry points that allow many different data formats to be simply converted to the internal data base format. Different routines are used by different filters.

The creation routines for the *geom* library define some simple data formats:

- ☐ A list of vertices is a 2D array of floats of X, Y, and Z.
- ☐ A list of normals is a 2D array of floats of NX, NY, and NZ.
- ☐ A list of float colors is a 2D array of floats of R, G, B (in the range of 0.0 to 1.0).
- ☐ A list of integer colors is also defined where R, G, and B are bytes packed into a 32-bit word. R occupies bits 23–16, G bits 15–8, B bits 7–0. All colors are stored internally as arrays of floats.

Many routines have an *alloc* flag as a parameter. If this flag is set to **GEOM_COPY_DATA**, the *geom* routine allocates its own space and copies the data of the object. If this flag is set to **GEOM_DONT_COPY_DATA**, the *geom* routine does not copy the data. In this case, the data must be allocated using the C entry library routine **malloc**(3C). When in doubt, use **GEOM_COPY_DATA**.

A *geom* object is initially created without any data at all. Data is then added to the object incrementally. A typical sequence would be to create an object of type polyhedron, add a polygon list, add vertices, then add normals. Notice that an object can be in an intermediate state where it doesn't make sense — it can have a polygon list without vertices, for example.

To reduce the number of procedure calls required to create an object, *geom* also provides macro functions that create and add various pieces of data. When one of these calls has a parameter for an optional piece of data (*normals* and *colors*, for example), **GEOM_NULL** can be used to indicate that this object does not have this type of data.

GEOMadd_disjoint_line

```
GEOMadd_disjoint_line(obj, verts, colors, n, alloc)
GEOMobj  *obj;
float    *verts;
float    *colors;
int      n;
int      alloc;
```

Adds an array of lines to an object of type **GEOM_POLYTRI**. It adds the vertices of this disjoint line to any existing disjoint lines in the object.

GEOMadd_disjoint_polygon

```
GEOMadd_disjoint_polygon(obj, verts, normals, colors, nverts, flag, alloc)
GEOMobj  *obj;
float    *verts, *normals, *colors;
int      nverts;
int      alloc;
int      flag;
```

Adds a disjoint polygon to a polyhedron object. The polygon has *nverts* vertices which are specified in the array *verts*. The *normals* and *colors* arguments can contain the *normals* and *colors* for the object, or they can be **GEOM_NULL**. The *flag* argument contains two pieces of information: the nature of the polygon (whether it is convex, concave, or complex), and whether the vertices of the polygon should be shared with the other vertices in the object. Shared vertices create an object whose vertices approximate a smooth object (such as a sphere); unshared vertices create an object that is faceted. The *flags* **GEOM_SHARED** and **GEOM_NOT_SHARED** are used to determine whether the vertices are shared or not. The values **GEOM_CONCAVE**, **GEOM_CONVEX**, and **GEOM_COMPLEX** can be OR'd with the other value to produce the *flag* argument.

Specifying shared vertices causes this routine to try to determine whether any vertices in the object are already represented. Instead of adding a new vertex when an old identical vertex is found, it uses a reference to this vertex. This process can take considerable time when the number of vertices in the object is large, but it produces an object that is significantly more efficient to render, because the resulting object contains fewer vertices to transform, light, and shade.

GEOMadd_float_colors

```
GEOMadd_float_colors(obj, colors, n, alloc)
GEOMobj  *obj;
float    *colors;
int      n;
int      alloc;
```

Adds a list of float colors to an object. This routine cannot be used with objects of type **GEOM_POLYTRI**.

GEOMadd_int_colors

```
GEOMadd_int_colors(obj, colors, n, alloc)
GEOMobj        *obj;
unsigned long  *colors;
int            n;
int             alloc;
```

Adds a list of integer colors to an object. This routine cannot be used with an object of type **GEOM_POLYTRI.**

GEOMadd_label

```
GEOMadd_label(obj, text, ref_point, offset, height, color, label_flags)
GEOMobj         *obj;
char            *text;
float           ref_point[3];
float           offset[3];
float           height;
float           color[3];
int             label_flags;
```

This routine adds a text string and related characteristics to an existing label object. Each label object can have more than one text string, along with related characteristics for each string. Each such string is added by a separate call to **GEOMadd_label** for the same label object. All data is copied (**GEOM_COPY_DATA** is assumed). The arguments are as follows:

text — The text string for the label.

ref_point
offset — These arguments determine the positioning of the label. The reference point is an array of X, Y, and Z coordinates. For a label used as a window title, these are in screen space, with (–1, –1, –1) at the lower left rear corner and (1, 1, 1) at the upper right front corner, and are not transformed. For other labels the coordinates of the reference point are transformed. The offset is an array of X, Y, and Z values in screen space. After the reference point is transformed (if necessary), the offset is applied to determine the final position of the reference point in screen space. The label always appears upright and in a plane parallel to the display surface.

height — The height of the label in screen space.

color — An RGB triple specifying the text color, or **GEOM_NULL** to indicate that the foreground color (usually white) should be used. (When the text background rectangle is drawn, the window background color is used for the rectangle.)

label_flags — An integer returned by a call to **GEOMcreate_label_flags**, or a value of –1 to indicate that the default label flags in the label object, added by the call to **GEOMcreate_label**, should be used. For a given label object, either all text strings must use the default label flags, or no text strings can use the default label flags. That is, either all calls to **GEOMadd_label** must pass –1 for the *label_flags* argument, or no calls to **GEOMadd_label** can pass –1 for the *label_flags* argument.

GEOMadd_normals

```
GEOMadd_normals(obj, normals, n, alloc)
GEOMobj   *obj;
float           *normals;
int             n;
int             alloc;
```

Adds a list of *normals* to an object. This routine cannot be used with objects of type **GEOM_POLYTRI** or **GEOM_SPHERE.**

GEOMadd_polygon

GEOMadd_polygon(*obj, nverts, indices, flags, alloc*)
GEOMobj **obj;*
int *nverts;*
int **indices;*
int *flags;*
int *alloc;*

Adds a polygon to a polyhedron object. The *indices* argument specifies an array of *nverts* integers, where each integer is an index into a vertex array that is added with the **GEOMadd_vertices** call either before or after this call is made. If multiple calls to **GEOMadd_vertices** are made, the first vertex added always remains the first vertex in the list. The indices in this array are "1 based". The first vertex in the list is 1, not 0. The *flags* argument can be either **GEOM_CONCAVE** or **GEOM_CONVEX.**

GEOMadd_polygons

GEOMadd_polygons(*obj, plist, flags, alloc*)
register GEOMobj **obj;*
register int **plist;*
int *flags;*
int *alloc;*

Adds a polygon list to a polyhedron object. The polygon list is an array of **ints** where the first **int** (*plist*[0]) indicates the number of vertices in the first polygon. The number of vertices (*plist*[0]) is followed by that number of indices into the vertex list. The second polygon's vertex list immediately follows the first. The list is terminated with a 0 number of vertices after the last polygon's vertex list. As with the **GEOMadd_polygon** routine, the the vertex indices are "1 based". The first vertex in the list is 1, not 0. The flags argument can be either **GEOM_CONCAVE** or **GEOM_CONVEX.**

GEOMadd_polyline

GEOMadd_polyline(*obj, verts, colors, n, alloc*)
GEOMobj **obj;*
float **verts;*
float **colors;*
int *n;*
int *alloc;*

Adds a polyline to an object of type **GEOM_POLYTRI.** The colors argument can be **GEOM_NULL.**

GEOMadd_polytriangle

GEOMadd_polytriangle(*obj, verts, normals, colors, n, alloc*)
GEOMobj **obj;*
float **verts;*
float **normals;*
float **colors;*
int *n;*
int *alloc;*

Adds a polytriangle to the object. Note that *colors* is an array of **float** *colors*, not **int** *colors*v. An object can contain more than one polytriangle strip.

GEOMadd_radii

GEOMadd_radii(*obj, radii, n, alloc*)
GEOMobj **obj;*
float **radii;*
int *n;*

int *alloc;*

This routine adds the radii supplied to an object of type **GEOM_SPHERE.** The number *n* contains the number of spheres in the object. The *alloc* parameter is **GEOM_DONT_COPY_DATA** if the data has been allocated using the **malloc**(3C) routine, and has not been freed by the application. It should be **GEOM_COPY_DATA** otherwise.

GEOMadd_vertices

GEOMadd_vertices(*obj, verts, n, alloc*)
GEOMobj **obj;*
float **verts;*
int *n;*
int *alloc;*

Adds a list of vertices to an object. This routine should not be used for objects of type polytriangle or type sphere (use **GEOMadd_polytriangle** or **GEOMcreate_sphere** instead).

GEOMadd_vertices_with_data

GEOMadd_vertices_with_data(*obj, verts, normals, colors, n, alloc*)
GEOMobj **obj;*
float **verts;*
float **normals;*
unsigned int *colors;*
int *n;*
int *alloc;*

Adds *vertices*, *colors*, and *normals* to the object. It assumes **integer** *colors*. Both the *normals* and *colors* parameters can be **GEOM_NULL.** This is a macro function combining the **GEOMadd_vertices, GEOMadd_normals,** and **GEOMadd_int_colors** routines.

GEOMcreate_label

GEOMobj *
GEOMcreate_label(*extent, label_flags*)
float **extent;*
int *label_flags;*

This routine creates a label object. Each label object can have more than one text string, along with related characteristics for each string. Each such string is added by a separate call to **GEOMadd_label** for the same label object. The *label_flags* argument to **GEOMcreate_label** is normally an integer returned by a call to **GEOMcreate_label_flags.** It specifies default characteristics for all text strings in the label object. Either all text strings must use the default label flags, or no text strings can use them; see **GEOMadd_label** for more information. The *extent* argument can be **GEOM_NULL** if no extent is known.

GEOMcreate_label_flags

int
GEOMcreate_label_flags(*font_number, title, background, drop, align, stroke*)
int *font_number, title, background, drop, align, stroke;*

This routine creates and returns a bit mask that is used to represent some characteristics of a label. The label flags are added by a call to **GEOMcreate_label** or to **GEOMadd_label.** To add a text string and related characteristics to the label, use the **GEOMadd_label** routine. The arguments are as follows:

font_number — An integer from 0 through 21 that specifies the font for the label's text string.

title — A value of 1 means that the label is to be used as a title for the window. The label is drawn in an absolute position with respect to screen space, which is defined so that (–1, –1, –1) is the lower left rear corner and (1, 1, 1) is the upper right front corner. A value of 0 means that the reference point of the label is transformed before the label is drawn. See the documentation for the **GEOMadd_label** routine for more information.

background — A value of 1 means that both the foreground text and the background rectangle that encloses the text are drawn. A value of 0 means that only the foreground text is drawn.

drop — A value of 1 means that a one-pixel drop-shadow highlight is added to the text. This makes the text stand out against a background of similar color. A value of 0 means that no highlight is added.

align — Specifies the position of the reference point within the label and therefore the alignment of the label. A value of **GEOM_LABEL_LEFT** places the reference point at the lower left corner of the label. A value of **GEOM_LABEL_CENTER** places the reference point at the bottom center of the label. A value of **GEOM_LABEL_RIGHT** places the reference point at the lower right corner of the label.

stroke — Not implemented; the value should be 0.

GEOMcreate_mesh

```
GEOMobj  *
GEOMcreate_mesh(extent, verts, m, n, alloc)
float       *extent;
float       *verts;
int         m, n;
int         alloc;
```

Creates a mesh from a 2D array of vertices. The dimensions of the array are specified by the *m* and *n* parameters. The first *n* vertices constitute the first row of the mesh. There are *m* rows of vertices. The extent parameter can be **GEOM_NULL** if no extent is known.

GEOMcreate_mesh_with_data

```
GEOMobj       *
GEOMcreate_mesh_with_data(extent, verts, normals, colors, m, n, alloc)
float          *extent;
float          *verts;
float          *normals;
unsigned long  *colors;
int            m, n;
int            alloc;
```

This routine is a macro function combining the **GEOMcreate_mesh**, **GEOMadd_int_colors**, and **GEOMadd_normals** routines.

GEOMcreate_obj

```
GEOMobj  *
GEOMcreate_obj(type, extent)
int         type;
float       *extent;
```

Type should be one of **GEOM_LABEL, GEOM_MESH, GEOM_POLYHEDRON, GEOM_POLYTRI** or **GEOM_SPHERE.** Extent can be either the extent of the object or **GEOM_NULL** if no extent is known. This routine creates an object of the specified type. Initially the object has no data.

GEOMcreate_polyh

```
GEOMobj  *
GEOMcreate_polyh(extent, verts, n, plist, flags, alloc)
float        *extent;
float        *verts;
int          n;
int          *plist;
int          flags;
int          alloc;
```

This routine is a macro function combining the **GEOMcreate_obj, GEOMadd_vertices,** and **GEOMadd_polygons** routines. The *flags* argument can be either **GEOM_CONCAVE** or **GEOM_CONVEX.**

GEOMcreate_polyh_with_data

```
GEOMobj      *
GEOMcreate_polyh_with_data(extent, verts, normals, colors, n, plist, flags, alloc)
float          *extent;
float          *verts;
float          *normals;
unsigned long  *colors;
int            n;
int            *plist;
int            flags;
int            alloc;
```

This routine is a macro function combining the **GEOMcreate_polyh, GEOMadd_int_colors,** and **GEOMadd_normals** routines. The *flags* argument can be either **GEOM_CONCAVE** or **GEOM_CONVEX**

GEOMcreate_scalar_mesh

```
GEOMobj  *
GEOMcreate_scalar_mesh(xmin, xmax, ymin, ymax, mesh, colors, n, m, alloc)
float        xmin, xmax, ymin, ymax
float        *mesh, *colors;        /* Colors is R,G,B */
int          n, m;
```

Creates a mesh from a single array of scalar values (a height field). The scalars are taken to be the Z component of the object. X will be evenly spaced between *xmin* and *xmax*, and Y will be evenly spaced between *ymin* and *ymax*. The colors parameter can be **GEOM_NULL.**

GEOMcreate_sphere

```
GEOMobj      *
GEOMcreate_sphere(extent, verts, radii, normals, colors, n, alloc)
float          *extent;
float          *verts;
float          *radii;
float          *normals;
unsigned long  *colors;
int            n, alloc;
```

Creates a sphere object. The vertices (*vert*) argument specifies the sphere centers. The *normals* and *colors* arguments can be **GEOM_NULL**. The *normals* are generally not used. To create a sphere with **float** *colors*, use the **GEOMadd_float_colors** routine after the sphere is created.

GEOMdestroy_obj

GEOMdestroy_obj(*obj*)
GEOMobj **obj;*

Frees all memory associated with the object, including memory given to a "create" call with the flag **GEOM_DONT_COPY_DATA.**

OBJECT UTILITY ROUTINES

The generation of proper normals is critical to an accurate and illustrative description of geometry. The following routines are provided to generate normals.

GEOMauto_transform

GEOMauto_transform(*obj*)
register GEOMobj **obj;*

GEOMauto_transform_non_uniform

GEOMauto_transform_non_uniform(*obj*)
register GEOMobj **obj;*

Transforms the object specified to lie within the cube from –1 to 1 in X, Y, and Z. The scaling and translation factors are uniform for **GEOMauto_transform** and nonuniform for **GEOMauto_transform_non_uniform.**

Once a *geom* object has been created, the utility routines can be used. In the Stellar architecture, the most efficient object format is the polytriangle for nonsphere surface descriptions and the polyline for wireframe descriptions. The following two routines convert data to the proper type. The conversion routines convert to either polytriangles, polylines, or both, depending on the setting of the *flags*. Sphere primitives should not be converted.

This conversion should be performed after normals have been generated for the object (if normals are desired) as the conversion results in a loss of vertex coherence information.

GEOMauto_transform_list

GEOMauto_transform_list(*objs, n*)
register GEOMobj ***objs;*
register int *n;*

GEOMauto_transform_non_uniform_list

GEOMauto_transform_non_uniform_list(*objs, n*)
register GEOMobj ***objs;*
register int *n;*

Transforms the list of objects specified to lie within the cube from –1 to 1 in X, Y, and Z. First the bounding box of all objects in the list is generated, then scaling and translation factors are computed to transform this box to lie inside the cube from –1 to 1 in X, Y, and Z. The scaling and translation factors are uniform for **GEOMauto_transform_list** and nonuniform for **GEOMauto_transform_non_uniform_list**. The relative sizes of objects in the list are not affected.

GEOMcreate_normal_object

GEOMobj *
GEOMcreate_normal_object(*obj,scale*)
GEOMobj **obj;*
float *scale;*

This routine takes an object that has normal data and returns an object that consists of disjoint lines that represent the normals of that object. The normals will be of length *scale* times their current length.

GEOMcvt_mesh_to_polytri

GEOMcvt_mesh_to_polytri(*tobj, flags*)
GEOMobj **tobj;*
int *flags;*

Creates a polytriangle or polyline description of the mesh object given. The resulting object contains one large polytriangle strip (if the *flags* argument is **GEOM_SURFACE**) and *n* * *m* polylines (if the *flags* argument is **GEOM_WIREFRAME**).

GEOMcvt_polyh_to_polytri

GEOMcvt_polyh_to_polytri(*tobj, flags*)
GEOMobj **tobj;*
int *flags;*

Uses a graph traversing algorithm to generate either a surface or a wireframe description of a polyhedron object, depending on the *flags* argument. It attempts to share as many vertices as possible. The surface algorithm can take a reasonably long time to complete for very large objects. The wireframe algorithm creates polylines for large connected strips and disjoint lines for smaller ones. The *flags* argument can be **GEOM_SURFACE, GEOM_WIREFRAME**, or **GEOM_EXHAUSTIVE**. The **GEOM_EXHAUSTIVE** flag should be used only in dire circumstances.

GEOMflip_normals

GEOMflip_normals(*obj*)
GEOMobj **obj;*

This routine inverts the direction of the normals in the object given.

GEOMgen_normals

GEOMgen_normals(*obj, flags*)
GEOMobj **obj;*
int *flags;* /* 0 or GEOM_FACET_NORMALS */

Generates surface normals for **GEOM_MESH** or **GEOM_POLYHEDRON** objects. By default, it assumes that the object is an approximation of a smooth surface. If the flags field is **GEOM_FACET_NORMALS** and the object is a polyhedron, a separate normal is generated for each vertex. Currently this is done by duplicating vertices and hence decreases the performance of the resulting object. Normals generated by this routine are guaranteed to be of unit length.

GEOMnormalize_normals

GEOMnormalize_normals(*obj*)
GEOMobj **obj;*

Normalizes (converts to unit length) the normals of the object specified. Normals are normalized automatically by the **GEOMgen_normals** routine.

GEOMset_computed_extent

GEOMset_computed_extent(*obj, extent*)
GEOMobj **obj;*
float *extent[6];*

Sets the extent of the object to the *extent* passed in. The *extent* passed in should contain in order: *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*. This can be used in conjunction with either **auto_transform** routine to perform arbitrary scaling and translation of objects.

GEOMset_extent

GEOMset_extent(*obj*)
GEOMobj **obj;*

Sets the extent of the object given. Currently, it is not implemented properly for objects of type **GEOM_SPHERE.**

GEOMset_object_group

GEOMset_object_group(*obj, name*)
GEOMobj **obj;*
char **name;*

This routine is used when storing multiple AVS objects (groups of *geom* objects) in a single *geom* file. By default, when AVS reads a *geom* file it places all *geom* objects in that file into a single AVS object. The **read_subset** script language command can read a subset of the *geom* objects in a *geom* file and place only those *geom* objects into an AVS object. Each *geom* object to be placed into the same AVS object must have the same group name, which is added by the **GEOMset_object_group** routine. The **read_subset** command takes a group name as an argument and places all *geom* objects in the *geom* file that have that group name into a single AVS object. The **read_subset** command ignores all *geom* objects in the *geom* file that do not have that group name.

GEOMset_pickable

GEOMset_pickable(*obj, pickable*)
GEOMobj **obj;*
unsigned long *pickable;*

This routine sets the pickable state of an object. If multiple objects are placed in a *geom* file, by default they are not pickable individually. If this attribute is set to "1", they can be picked individually when running the AVS viewing application.

GEOMunion_extents

GEOMunion_extents(*obj1, obj2*)
GEOMobj **obj1,* **obj2;*

Sets the extent of *obj1* to include the extent of *obj2*. It generates the extents of both objects if they aren't set already.

OBJECT PROPERTY ROUTINES

A feature of the *geom* library allows arbitrary value lists to be associated with each object. These value lists can then be interpreted by packages reading in the objects. The object format supports values that are arbitrarily long. Currently only integer values are supported by the subroutine interface.

GEOMadd_int_value

GEOMadd_int_value(*obj, type, value*)
GEOMobj **obj;*
int *type;*
int *value;*

Adds an integer property to an object. Currently the only fully supported property of an object is the color (type **GEOM_COLOR**).

GEOMquery_int_value

int
GEOMquery_int_value(*obj, type, value*)

GEOMobj **obj;*
int *type;*
int **value;*

An integer value can be queried with this routine. The type is an integer value. The only fully supported property type is **GEOM_COLOR.** Its value can be queried with:

GEOMquery_int_value(*obj, GEOM_COLOR, &color***);**

This routine returns 0 if no color was associated with the object.

GEOMset_color

GEOMset_color(*obj, color***)**
GEOMobj **obj;*
unsigned long *color;*

Sets the *color* property of an object (by calling the **GEOMadd_int_value** routine).

OBJECT TEXTURE-MAPPING ROUTINES

Each surface object can have *uv* data associated with each vertex. The *uv* data consists of two floating point values per vertex, which specify a mapping into a texture map. The value of *u*=0, *v*=0 is the index into the upper left hand corner of the texture map; *u*=1, *v*=1 is the lower right hand corner. These values are stored in memory as an array of floating point values.

GEOMadd_polytriangle_uvs

GEOMadd_polytriangle_uvs(*obj, uvs, i, n, alloc***)**
GEOMobj **obj;*
float **uvs;*
int *i;*
int *n;*
int *alloc;*

Each polytriangle object has an array of polytriangle strips. This routine is used to add *uv* data to a polytriangle object. These polytriangle strips are kept in an array in the order in which they were added: the first polytriangle is index 0, the second is index 1, etc. This routine adds *uv* data for a single polytriangle strip. The index of the polytriangle strip is the variable *i*. This polytriangle strip should have *n* vertices. The *alloc* parameter is **GEOM_DONT_COPY_DATA** if the data has been allocated using the **malloc**(3C) routine, and has not been freed by the application. It should be **GEOM_COPY_DATA** otherwise.

GEOMadd_uvs

GEOMadd_uvs(*obj, uvs, n, alloc***)**
GEOMobj **obj;*
float **uvs;*
int *n;*
int *alloc;*

This routine adds the *uv* data to an object of type **GEOM_POLYHEDRON**, or **GEOM_MESH.** *n* should specify the number of vertices in the object. The *alloc* parameter is **GEOM_DONT_COPY_DATA** if the data has been allocated using the **malloc**(3C) routine, and has not been freed by the application. It should be **GEOM_COPY_DATA** otherwise.

GEOMcreate_mesh_uvs

GEOMcreate_mesh_uvs(*obj, umin, vmin, umax, vmax***)**
GEOMobj **obj;*
double *umin, vmin, umax, vmax;*

This routine creates *uv* data for a mesh object such that the 0,0 vertex in the mesh will have *u*=0, *v*=0, and the *n,m* vertex in the mesh will have *u*=1, *v*=1. It is an error to use this routine with an object that is not of type **GEOM_MESH.**

GEOMdestroy_uvs

GEOMdestroy_uvs(*obj*)
GEOMobj **obj;*

This routine takes an object that has *uv* data for each vertex and turns it into an object that doesn't have *uv* data for each vertex.

OBJECT FILE UTILITIES

The *geom* library supports reading and writing of objects of type **GEOM_POLYTRI** and **GEOM_SPHERE.** Since these are the most efficient formats for Stellar hardware, an application increases performance by converting to these object types before writing data to a file.

GEOMread_obj

GEOMobj *
GEOMread_obj(*fd, flags*)
int *fd;*
int *flags;*

Performs a read operation on the file descriptor given and interprets the data it finds as a *geom* object. Data can be stripped off by specifying the **GEOM_NORMALS** flag (to strip off the normals), the **GEOM_VCOLORS** flag (to strip of the colors), or the OR of these values (to strip off normals and colors). A *flags* value of 0 means leave the data intact.

GEOMwrite_obj

GEOMwrite_obj(*obj, fd, flags*)
GEOMobj **obj;*
int *fd;*
int *flags;*

Writes a *geom* object to a file. The *fd* parameter is a file descriptor representing the file or device to write to. Data can be stripped off by specifying the **GEOM_NORMALS** flag (to strip off the normals), the **GEOM_VCOLORS** flag (to strip of the colors), or the OR of these values (to strip off normals and colors). A flags value of 0 means leave the data intact.

GEOMwrite_text

GEOMwrite_text(*obj, fp, flags*)
GEOMobj **obj;*
FILE **fp;*
int *flags;*

This routine writes an ASCII version of the *geom* object specified to the stream *fp*. This routine is implemented for all *geom* object types and is useful for debugging or transporting *geom* data to different architectures (although its usefulness is limited because of the current lack of a "read object" routine).

AVS MODULE INTERFACE ROUTINES

A geometry object in an AVS module describes changes to the geometry of a particular scene that is represented by a module input or output. AVS allows a user module to create geometry objects as outputs; in this release it does not support using them as inputs. Geometry output is typically used as input to a gemoetry renderer module such as the geometry viewer.

A geometry data object is called an *edit list*. This is an arbitrarily long list of changes to be made in the current scene. Each change pertains to a particular object or light source. Changes are made in the order specified in the edit list. The AVS data type for an edit list is **GEOMedit_list**. A C language module computation routine declares an argument representing an input port or parameter as **GEOMedit_list** and an argument representing an output port as **GEOMedit_list *** (note the single asterisk).

Each object or light is referred to by a name which is an ASCII string. Any object that doesn't already exist is created the first time an attempt to change that particular object is made. By default, an object name is modified by the port through which it is communicated. This prevents two different modules from modifying each other's objects. For example, two "plate" modules would each try to modify the data for the object named "plate". Since the name is modified by the port, the first plate module modifies "plate.0", and the second modifies "plate.1". When it is desirable for a module to use the absolute name of an object, it can precede the object name by a % character (e.g., "%plate").

AVS has routines that allow a module to change several properties of an object in an edit list:

❑ The geometric data defining the object

❑ Surface or line color

❑ Render mode (Gouraud, Phong, wireframe, etc.)

❑ Parent (the name of the parent object)

❑ Material properties

❑ Transformation

The name of each light source in an edit list is a string of the form "light*n*", where *n* is an integer from 1 through 16. The name of each camera in an edit list is a string of the form "camera*n*", where *n* is an integer from 1 through the number of defined views.

Each time a module is invoked, it starts with an empty edit list. It places into the edit list changes that it wants to be made for this invocation. In creating and using edit lists, geometry objects, and light sources, a module uses routines in the *geom* library. A module typically uses the following steps in preparing an edit list for output:

❑ Initialize the edit list, using **GEOMinit_edit_list**. This creates a new list or empties an existing list.

❑ Create and modify geometry objects or lights sources, using routines in the *geom* library.

❑ Modify the edit list, using routines whose names begin with **GEOMedit** in C (such as **GEOMedit_geometry**).

❑ For a coroutine module, use **AVScorout_output** to output the list, and then use **GEOMdestroy_edit_list** to deallocate the list.

A module must deallocate an existing edit list before reusing the list. For a subroutine module, the edit list passed to the module as an output argument is the edit list the module created on its last execution. The module must deallocate this list at the start of each invocation of the module, normally by calling the **GEOMinit_edit_list** routine before modifying the list. A coroutine module can use **GEOMdestroy_edit_list** to deallocate a list after calling **AVScorout_output**.

GEOMdestroy_edit_list

GEOMdestroy_edit_list(*list***)**
GEOMedit_list *list;*

Destroys an existing edit list.

GEOMedit_color

```
GEOMedit_color(list, name, color)
GEOMedit_list   list;
char            *name;
float           color[3];
```

Sets the color of the object named *name*. The *color* argument is an RGB triple, each float in the range 0.0 to 1.0. If the *name* argument is "camera*n*", where *n* is an integer ranging from 1 to the number of views, this routine sets the background color for the view specified by the index *n*. If the *name* argument is "light*n*", where *n* is an integer ranging from 1 to the number of light sources, this routine sets the light source color for the light source specified by the index *n*.

GEOMedit_concat_matrix

```
GEOMedit_concat_matrix(list, name, matrix)
GEOMedit_list   list;
char            *name;
float           matrix[4][4];
```

Post-concatenates *matrix* to the matrix of the object named *name*. If the *name* argument is "camera*n*", where *n* is an integer ranging from 1 to the number of views, this routine post-concatenates *matrix* to the camera matrix for the view specified by the index *n*. If the *name* argument is "light*n*", where *n* is an integer ranging from 1 to the number of light sources, this routine post-concatenates *matrix* to the light matrix for the light source specified by the index *n*.

GEOMedit_geometry

```
GEOMedit_geometry(list, name, obj)
GEOMedit_list   list;
char            *name;
GEOMobj         *obj;
```

Specifies a change in the geometry for an object named *name* in the edit list *list*. The first edit geometry entry in an edit list for a specific object replaces all existing geometry for this object with the geometry specified by *obj*. All other edit geometry entries for the object named *name* simply add additional geometry to that object.

Entering geometry into an edit list does not copy the geometric description of the object. Instead, it creates a reference to the **GEOMobj** specified in the call to **GEOMedit_geometry**. This means that a module must take care not to modify the **GEOMobj** until the edit list has been destroyed. The module must also destroy its own reference to the **GEOMobj**, using **GEOMdestroy_obj**, when it is finished with the geometry. For most purposes, a call to **GEOMdestroy_obj** should be made after every call to **GEOMedit_geometry**.

GEOMedit_group

```
GEOMedit_group(list, name, parent)
GEOMedit_list   list;
char            *name;
char            *parent;
```

Sets the parent of the object named *name* to be the object named*parent*. The top level object is referred to by a "NULL" name entry.

GEOMedit_light

```
GEOMedit_light(list, name, type, status)
GEOMedit_list   list;
char            *name, *type;
```

```
int                 status;
```

Changes the light source representation for a light source. The light source name is "light*n*", where *n* is an integer from 1 through 16. The *type* argument is one of "spot", "directional", "point", or "bi-directional". If the *status* argument is 1, the light source is on; if the *status* argument is 0, the light source is off.

GEOMedit_properties

```
GEOMedit_properties(list, name, ambient, diffuse, specular, spec_exp,
                    transparency, spec_col)
GEOMedit_list       list;
char                *name;
float               ambient, diffuse, specular, spec_exp;
float               transparency, spec_col[3];
```

Changes the material properties of the object named *name*. The properties are ambient, diffuse, and specular reflection coefficients; specular exponent; transparency; and specular color (as an RGB triple). Values to be changed are in the range 0.0 to 1.0 except for the specular exponent, which is greater than or equal to 1.0. If any value is –1.0, that property is not changed.

GEOMedit_render_mode

```
GEOMedit_render_mode(list, name, mode)
GEOMedit_list       list;
char                *name;
char                *mode;
```

Sets the render mode of the object named *name* to one of "gouraud", "phong", "lines", "smooth_lines", "no_light", "inherit", or "flat".

GEOMedit_set_matrix

```
GEOMedit_set_matrix(list, name, matrix)
GEOMedit_list       list;
char                *name;
float               matrix[4][4];
```

Sets the transformation matrix for the object named *name* to the matrix *matrix*. If the *name* argument is "camera*n*", where *n* is an integer ranging from 1 to the number of views, this routine sets the camera matrix for the view specified by the index *n*. If the *name* argument is "light*n*", where *n* is an integer ranging from 1 to the number of light sources, this routine sets the light matrix for the light source specified by the index *n*.

GEOMedit_visibility

```
GEOMedit_visibility(list, name, visibility)
GEOMedit_list       list;
char                *name;
int                 visibility;
```

Sets the visibility of the object named *name*. If the *visibility* argument is 1, the visibility is set to on; if the *visibility* argument is 0, the visibility is set to off; if the *visibility* argument is –1, the object is deleted.

GEOMinit_edit_list

```
GEOMedit_list
GEOMinit_edit_list(list)
GEOMedit_list       list;
```

Initializes an existing edit list (removes all existing entries). If list is **GEOM_NULL**, it returns a new empty edit list.

FORTRAN BINDING

All of the *geom* routines also have a Stellar FORTRAN calling sequence. To call a routine from a FORTRAN program, you must use a slightly different routine name and different data declarations:

❑ **Routine Name:** Replace the **GEOM** prefix with **geom_** (*note the underscore*).

❑ **Data Declarations:** The following table shows how to convert C-language data declarations into FORTRAN declarations:

C Declaration	*FORTRAN Declaration*
int *var*	**INTEGER** *var*
int **var*	**INTEGER** *var*
unsigned int *var*	**INTEGER** *var*
unsigned int **var*	**INTEGER** *var*
float *var*	**REAL** *var*
float **var*	**REAL** *var*
double *var*	**REAL** *var*
double **var*	**REAL** *var*
GEOMobj **var*	**INTEGER** *var*
GEOMobj ***var*	**INTEGER** *var*
GEOMedit_list *var*	**INTEGER** *var*
GEOMedit_list **var*	**INTEGER** *var*

Many routines allow a NULL value for some arguments. In all such cases, the constant **GEOM_NULL** must be used to represent a NULL value.

FILES

/usr/avs/include/geom.h	C language header file
/usr/avs/include/geom.inc	FORTRAN header file
/usr/avs/lib/libgeom.a	*geom* library