

AVS APPLICATIONS GUIDE

Release 4
May, 1992

Advanced Visual Systems Inc.

Part Number: 320-0015-02, Rev B

NOTICE

This document, and the software and other products described or referenced in it, are confidential and proprietary products of Advanced Visual Systems Inc. (AVS Inc.) or its licensors. They are provided under, and are subject to, the terms and conditions of a written license agreement between AVS Inc. 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 SOFTWARE DESCRIBED HEREIN, SHALL BE DEEMED TO BE A WARRANTY BY AVS INC. FOR ANY PURPOSE OR GIVE RISE TO ANY LIABILITY OF AVS INC. WHATSOEVER. AVS INC. 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.

AVS INC. 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 AVS INC. 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 AVS Inc. for more detailed and current information.

Copyright © 1989, 1990, 1991, 1992
Advanced Visual Systems Inc.
All Rights Reserved

AVS is a trademark of Advanced Visual Systems Inc.

STARVENT is a registered trademark of Stardent Computer Inc.

IBM is a registered trademark of International Business Machines Corporation.

AIX, AIXwindows, and RISC System/6000 are trademarks of International Business Machines Corporation.

DEC and VAX are registered trademarks of Digital Equipment Corporation.

NFS was created and developed by, and is a trademark of Sun Microsystems, Inc.

HP is a trademark of Hewlett-Packard.

CRAY is a registered trademark of Cray Research, Inc.

Sun Microsystems is a registered trademark of Sun Microsystems, Inc.

SPARC is a registered trademark of SPARC International.

SPARCstation is a registered trademark of SPARC International, licensed exclusively to Sun Microsystems, Inc.

OpenWindows, SunOS, XDR, and XGL are trademarks of Sun Microsystems, Inc.

UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc.

Motif is a trademark of the Open Software Foundation.

IRIS and Silicon Graphics are registered trademarks of Silicon Graphics, Inc.

IRIX, IRIS Indigo, IRIS GL, Elan Graphics, and Personal IRIS are trademarks of Silicon Graphics, Inc.

Mathematica is a trademark of Wolfram Research, Inc.

X WINDOW SYSTEM is a trademark of MIT.

PostScript is a registered trademark of Adobe Systems, Inc.

FLEXIm is a trademark of Highland Software, Inc.

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.

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).

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

TABLE OF CONTENTS

1 AVS Data Interchange Application: ADIA

Overview	1-1
AVS Data Interchange Application: ADIA	1-1
System Components	1-2
file descriptor Module	1-2
Specifying Data	1-2
Assisting Users with File Based Specification	1-3
Byte Offsets, Strides, and Other Means of Locating Data	1-3
Variables and Equations	1-4
User Defined Variables	1-5
The Data Form	1-6
data dictionary Module	1-6
Tutorial: Reading in the AVS .x Image Format	1-6
Tutorial: Reading in a PLOT3D Data File	1-11
AVS Control Panel Widgets	1-16
Select Data File Browser	1-16
Read and Write Form Buttons	1-16
Send Data	1-16
Header Information	1-16
Variable List	1-16
Data File Widgets	1-17
AVS Field Description Form Widgets	1-17
Field Component Radio Buttons	1-17
Value Selection Browser	1-17
User Input and File Based Specification Radio Buttons	1-17
Enter Value Typein	1-18
Datafile Format Radio Buttons	1-18
Binary and XDR	1-18
ASCII	1-18
Data Type Radio Buttons	1-20
Logical File Radio Buttons	1-21
data dictionary Module	1-21
Select Data File Browser	1-22
Read Form Button	1-22
Send Data	1-22

Header Information	1-22
General Order of Operation	1-22

2 **The AVS Module Generator**

Overview	2-1
General Module Structure	2-1
Subroutine Modules	2-2
Coroutine Modules	2-5
Example Session	2-6
Initiating the Module Generator	2-6
Specifying the Module's Structure	2-8
Creating an Executable Module	2-11
Adding Code to the USER-SPECIFIED CODE SECTIONS	2-12
Hints	2-13
Detailed Description of Controls	2-14
Top-Level Controls	2-14
Module Description	2-14
Module Name	2-15
Module Type	2-15
C vs. FORTRAN	2-15
Subroutines vs. Coroutines	2-15
Editing Ports and Parameters	2-16
Unix Specification Tools	2-16
Source File Name	2-17
Include Hints	2-17
Writing Module Source Files	2-17
Reading Module Source Files	2-17
Writing Makefiles	2-18
Writing Manual Pages	2-19
Compiling Modules	2-19
Editing a Module's Source Code	2-20
Loading a Compiled Module into AVS	2-21
Debugging a Module	2-21
Port Editing	2-21
Field Editor	2-22
Geometry Editor	2-23
Parameter Editing	2-24

3 **The Data Viewer**

Why a Data Viewer?	3-1
Starting the Data Viewer	3-3
Leaving the Data Viewer	3-4

4 **The Data Viewer Interface**

Introduction	4-1
The Menu Bar	4-1
The Control List	4-5
The Control Panel	4-8
Field Legend	4-10
Control Widgets	4-11
The Output Window	4-12
Geometry Output Windows	4-12
Mechanics	4-12
Mouse Buttons	4-13
Geometry Output Window Buttons	4-14
Geometry Menu	4-15
Graph Output Windows	4-16
Image Output Windows	4-16
The Techniques	4-18

TABLE OF CONTENTS

AVS DATA INTERCHANGE APPLICATION: ADIA

Overview

AVS includes a facility to import external datasets into AVS fields—the **AVS Data Interchange Application** (ADIA). ADIA improves upon the facility provided in the **read field** module in these ways:

- It can read 16-bit "halfword" data. Many medical imaging applications produce 12-bit data in two 8-byte halfwords. (The data will be represented in the AVS field as 32-bit integers.)
- It supports variables and expressions, making it possible to define skips and spaces necessary to read the input data as a function, rather than as an absolute number.
- It is an interactive facility contained within AVS—one no longer needs to use a text editor to externally edit an ASCII header.
- It can read data format information, such as the dimensions of a dataset, from the input dataset and use these values as part of its variables and expressions.
- It creates ASCII data forms that define how to read a particular input dataset format. These data forms can be reused and exchanged.

AVS Data Interchange Application: ADIA

The problem of importing data from one format into an application that understands a completely different format can be complex. Within AVS, there are three techniques that you can apply to solving this problem. They include:

- writing a UNIX shell level filter program that converts an external data file into an AVS field file.
- writing a module that will read an external data format and convert it into an AVS field.
- using the **read field** module's data parsing mode read capabilities

System Components

The first two techniques require that you be able to write a program in either C or FORTRAN. The third technique doesn't require any programming, but it does require that you create a unique field file for each data file being read.

The AVS Data Interchange Application (ADIA) was developed in order to overcome some of the limitations of these techniques. ADIA allows you to describe an external file format and correlate it to an AVS field. In turn, data from a foreign format can be read from an external file into AVS and converted into an AVS field.

System Components

Integrated within AVS, ADIA is composed of two modules. The first module, called **file descriptor**, allows you to specify the external data file format by filling in a form. Once an external file format has been described, the data form can be saved for later use. The second module, called **data dictionary**, will allow you to import files using existing data forms that were created with the **file descriptor** module.

file descriptor Module

At the heart of ADIA is the **file descriptor** module. This module presents you with a series of forms. These forms allow you to describe where in the external data file the requisite information is located in order to convert an external file format into an AVS field.

Specifying Data

When specifying the components of a field, you can choose one of two methods for entering a value. In the first case, **User Input Specification**, you will be presented with one or more typeins where values can be typed directly. As an example, if an image is being read in and you are specifying the **Dimensions of Computational Space**, you would type **2** into the value typein. Sometimes a simple number is not enough to specify a value. In this case, variables and equations can be used in the typein. The use of variables and equations is described in the **Variables and Equations** section below.

The second method for entering values is **File Based Specification**. This method is used when an input element can be found in a disk file. For example, suppose the image file contains the dimensions of the image as the first two elements of the image file, and the dimensions are represented as 32 bit integers. When specifying where in the datafile to find the dimensions of the dataset, you would give a byte offset of zero for the first dimension and a byte offset of four for the second dimension.

Assisting Users with File Based Specification

There are several problems with file based specification. The first problem is that you may not know exactly where in the file a particular data value is located. A second problem is that you don't know what type the data is (i.e., is it a float or an integer?). Binary data files can be particularly difficult to deal with in these cases.

In order to help you with these problems, ADIA provides you with two mechanisms which can help you. The first mechanism is a range restriction facility. This allows you to pre-specify a reasonable range for a particular data element. For instance, when reading in the vector length of a data file, 15 might be the expected maximum. If during the file read process, 1024 is read in, an error message will appear and the process of reading the file will halt. This error checking will help prohibit outrageous data values being sent through an AVS network.

The second mechanism is a preview capability within the **file descriptor** module. At any time, you are able to request a look at the header information in a file. In the case of a field, this means that you can look at what the dimensions, vector length, etc., are for a particular file. Although this capability is duplicated by the **print field** module, sometimes a data type will be specified so improperly that it won't even be created. In order to preview a data file's header, select the **header information** button from the **file descriptor** control panel.

Byte Offsets, Strides, and Other Means of Locating Data

ADIA works on files where data can be found in regularly predictable places. Typically, an individual data element in a file is referred to by its location (in bytes) from the beginning of the file. This is referred to as the **Byte Offset**. After the first data element has been located, subsequent data element locations, are expressed as being at regular distances from each other. This is referred to as the **Stride**. **Stride** assumes that you are "standing on" the data value just read and specifies how many "strides" must be taken to get to the next data element. In a file where the data is stored contiguously, the **Stride** is **1**. In an AVS image file, the pixels are stored: alpha, red, green, blue. In this case the **Stride** for reading any one of these data elements would be **4**. This is because, after having read the first alpha value, you have to "stride" 4 bytes to get to the next one.

For ASCII files, ADIA provides additional tools for specifying where data lives in the file. Besides offering **Byte Offset** and **Stride** (in bytes), ADIA offers the ability to express the initial data location in line and word numbers or

through doing token searches. Let's look at an example. Suppose the data file looks like this:

Table 1-1. Sample ASCII File

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

The data set A starts on line 2 at word number 4. The data for B starts at line 2 on word number 5. Both A and B have a **Stride** of 5 since if you have read item A1, you need to “stride” 5 items to get to A2.

Searching for the starting point of a data element by using tokens may be more useful for a file which has names defining the data elements. This is similar to the AVS field file header:

```
# AVS field file
# this is a header file for a field to be
# used in conjunction with the build a field module of AVS
#
ndim = 3
dim1 = 64
dim2 = 64
dim3 = 64
nspace = 3
veclen = 1
data = byte
field = uniform
```

In order to read the number of dimensions (**ndim**), you want to look for the associated word **ndim**. After you have found the correct word, then the ASCII string which contains that number is the third number on the line (3 in this example). ADIA provides tools for specifying offset lines and words as well as for searching by token names. See the Section labelled “AVS Control Panel Widgets” at the end of this document for a detailed look at how this is done.

Variables and Equations

In many cases the size or location of a particular element is related to another value. For instance, data elements may be located within a file at offsets that are dictated by the data set's dimensions. In cases such as these, it is very useful to be able to specify an element using variables (e.g. dim1 * dim2). There are a number of predefined variables that can be used in the data forms. The list of variables includes:

- **variable** (user definable description)
- **ndim** (compute space)

- **nspace** (physical space)
- **veclen** (vector length)
- **dim1** (first dimension)
- **dim2** (second dimension)
- **dimN** (Nth dimension)
- **byte** (sizeof byte - typically 1)
- **int** (sizeof int - typically 4)
- **float** (sizeof float - typically 4)
- **double** (sizeof double - typically 8)

There are also valid math symbols. They include the symbols:

- + (Addition)
- - (Subtraction)
- / (Division)
- * (Multiplication)
- ('and ') (Open and Close Parentheses)

As with elements such as vector length, variables can be specified by User Input Specification or File Based Specification methods. Also, you are allowed the flexibility of creating your own variables beyond the list of presupplied variables. A list of all variables can be shown by selecting the **variable list** button from the **file descriptor** control panel. Also, the variable list and their current values can be displayed by selecting the **header information** button.

User Defined Variables

Located on the AVS **Field Description** form are a set of radio buttons called the **Field Component** radio buttons. The last button is labeled **Variables**. By selecting the **Variables** radio button, you can create your own user defined variables. Up to 25 user defined variables can be created.

For example, if you want to create a variable that equals the product of the field's dimensions, you would do the following.

1. Select **Variables** from the Field Component radio buttons
2. Give a name to the variable, such as *dimsize*, by typing *dimsize* into the **Variable Name** typein widget.
3. Let's assume that the field we are reading in is 3D. Since *dimsize* is the product of the field's dimensions, *dimsize* is equal to $dim1 * dim2 * dim3$. So by typing the expression $dim1 * dim2 * dim3$ into the **Enter Value** typein, we have set the variable *dimsize* equal to the product of the field's dimensions.
4. When the field is read from disk, *dimsize* will be automatically computed from the field's dimensions.

User defined variables can also be assigned their values by directly reading the value from the input file. This can be accomplished using File Based Spec-

Tutorial: Reading in the AVS .x Image Format

ification. See the section above, “Assisting Users with File Based Specification.”

One note of caution: be careful not to create circularly defined variables. For example, if you create the following two variables, the **file descriptor** module will not be able to resolve them.

```
user_1 = dim1 * user_2
user_2 = dim2 * user_1
```

The Data Form

Once you have completed the description of the external data file, a **data form** can be created. A data form is an ASCII file that contains the information describing how to convert an external file format into an AVS field.

Since ADIA’s data forms are ASCII files, they can be transported from one workstation type to another. Users are encouraged to submit these forms to the AVS International Center. It is much easier to support and maintain a collection of forms than a collection of modules. In addition, you are able to use the form from one file format to help in the specification of a form for a similar but different file format.

data dictionary Module

Once a data form has been specified in the **file descriptor** module, it can be used by the **data dictionary** module. In order to read an external data file, you specify a data form to use from a file browser. Next, specify the data file from a file browser. At this point, the file can be read in and converted to a particular AVS data type and then fed into an AVS network.

There are sample data forms for reading AVS image and volume format data in the `/usr/avs/data/adia` directory.

Tutorial: Reading in the AVS .x Image Format

This tutorial goes through the steps of creating a form for a two dimensional AVS .x image dataset. It is assumed that there is no form available that is similar to the .x file format, so the tutorial will start from the beginning.

The .x file format is quite simple (Figure 1-1.). First of all, the file is a binary file. At the beginning of the file are two integers. The first integer is the width of the image and the second integer is the height of the image. The image data

follows these two integers. This data is stored as a stream of bytes where the bytes are in the following order: alpha, red, green, and blue.

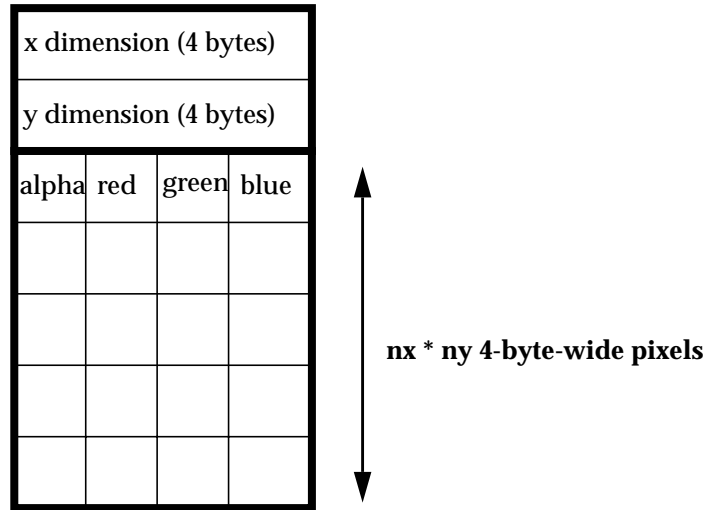


Figure 1-1: AVS.x Image Format

The output target is an AVS field with the form *2D 4-vector byte uniform*.

5. After starting up the Network Editor, drag the **file descriptor** module from the **Data Input** module column into the Network workspace with the left mouse button. Once the module has started up, a set of widgets will appear in the AVS control panel (see Figure 1-2).
6. The **AVS Field Description Form** will appear in the middle of the display (see Figure 1-3). This form contains all of the items that need to be filled in order to completely specify an external data file to be converted into an AVS field. Once the field form has appeared, the following steps should be taken.
7. Set the computational space dimensions for the field. This is equivalent to **ndim** for a field. Select the **Dimensions of Compute Space** radio button. Type the value **2** into the **Enter Value** widget. (See Figure 1-3).
8. Set the physical space dimensions for the field. This is equivalent to **nspac** for a field. Select the **Dimensions of Physical Space** radio button. Type the value **2** into the **Enter Value** widget.
9. Set the vector length for the field. This is equivalent to **veclen** for a field. Select the **Vector Length** radio button. Type the value **4** into the **Enter Value** widget.
10. Set the labels for the field. (This is optional.) Select the **Labels** radio button. The **Value Selection** browser will appear with five names in it (label 1 through label 4 and all labels). The four bytes of an AVS image are referred to as "alpha," "red," "green," and "blue." Select **label 1** from the list and then type the value **alpha** into the **Enter Value** widget. Select **label 2** from the list and then type the value **red** into the **Enter Value** widget. Select **label 3** from the list and

```
../ (avs)
adia
chemistry
colormap
```

Figure 1-2: file descriptor Main Control Panel

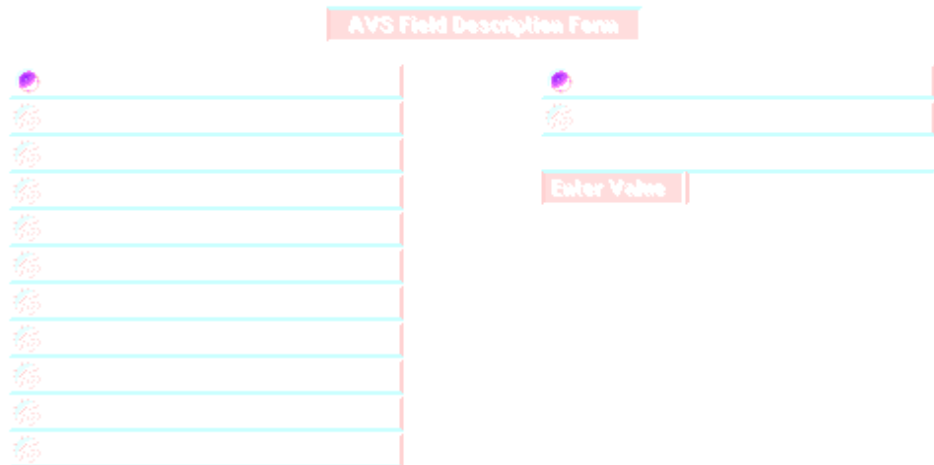


Figure 1-3: AVS File Description Form with Compute Space Set

then type the value **green** into the **Enter Value** widget. Select **label 4** from the list and then type the value **blue** into the **Enter Value** widget. (See Figure 1-4)

11. For this example, Units don't have to be set. Units, like Labels, are optional.
12. Set the dimensions for the field. This is equivalent to dim1, dim2... dimN for a field. Since the x and y dimensions of an AVS .x image dataset are specified within the file in the first eight bytes, the di-

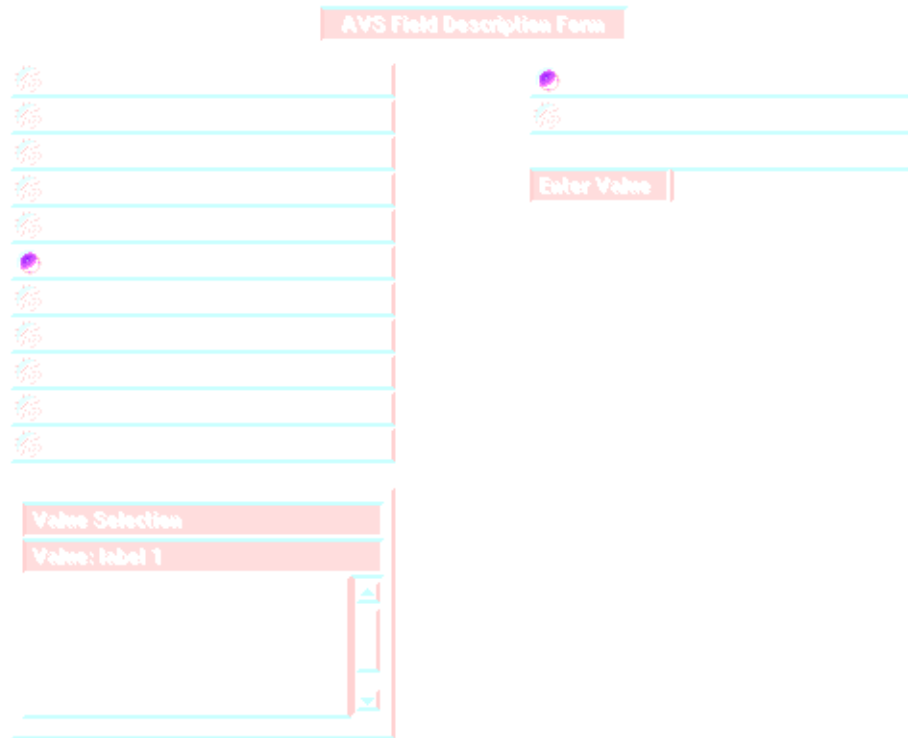


Figure 1-4: Setting Labels

mensions will be read from the file itself. Select the **Dimensions** radio button. The **Value Selection** browser will appear with three names in it (dimension 1, dimension 2 and all dimensions). Select **dimension 1** from the list. This will be the x dimension. Then, select the **File Based Specification** radio button. The default values for finding and reading **dimension 1** are the correct values, so no changes need to be made.

13. Continue setting the dimensions for the field. Select **dimension 2** (for y) from the list and then select the **File Based Specification** radio button. The second dimension is right after the first dimension in the coordinate file, so enter the value **4** in the **Byte Offset** typein. The value **4** is equivalent to skipping over the first 4 bytes in the data file (4 bytes = size of an integer). (See Figure 1-5)
14. Now establish how to read the data values from the file. This is equivalent to variable 1, variable 2 ... variable N for a field. Since the data of an AVS .x image dataset are found in the data file, scalar data values will be specified from information within the file. Select the **Data** radio button. The **Value Selection** browser will appear with five names in it (scalar_value 1 through scalar_value 4 and all data values). The data in the image file is contiguous, so all four scalar values can be read in at once. Select **all data values** from the list and then select the **File Based Specification** radio button. The scalar data in the data file is in byte format, so you must select **byte** from the data type radio buttons. Since the first scalar value occurs after the dimensions in the file, type **8** into the **Byte Offset** typein.



Figure 1-5: Setting the Dimensions

You don't have to change the **Stride** typein since the data is contiguous within the file.

15. The specification for the file is now complete. In order to read in a sample file, you must specify which data file to read. Select the **Browser for File 1** button in the AVS control panel. Select the file /usr/avs/data/image/mandrill.x from the **Select Data File** file browser.
16. Now that everything is specified, the data can be read from disk by selecting the **send data** button in the AVS control panel. Once the file has been read, a field will be output on the **file descriptor** module's output port. You can hook this port up to the **display image** module to check out your results (see Figure 1-6).

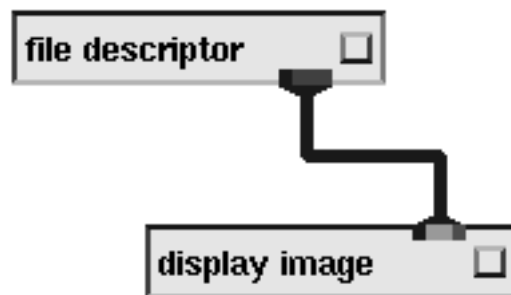


Figure 1-6: Debug Network for .x Images

17. You can also save the image form away for later use. Select the **write form** button in the AVS control panel. Now, using the file browser, select a filename to save the form into. Once you have chosen the filename, the form file is automatically created. The form you create should look something like:

```
# AVS file descriptor: Version 1.0
field {
  compute_space 2
  physical_space 2
  vector_length 4
  data_type byte
  uniform uniform
  label_1 alpha
  label_2 red
  label_3 green
  label_4 blue
  dimension_1 {
    value 0
    file_number 1
    file_format binary
    data_type integer
    max_value 1000
  }
  dimension_2 {
    value 4
    file_number 1
    file_format binary
    data_type integer
    max_value 1000
  }
  all_data {
    value 8
    file_number 1
    file_format binary
    data_type byte
    max_value ***
    stride 1
  }
  coordinate_1 0
  coordinate_2 0
}
```

Tutorial: Reading in a PLOT3D Data File

This tutorial goes through the steps to create a form for a 3D/whole PLOT3D dataset. It is assumed that there is no form available that is similar to the PLOT3D file format, so the tutorial will start from the beginning.

PLOT3D data comes in two separate files. One data file contains the coordinate information and the other data file contains the data values at each coordinate. Both of these data files are binary files. In the coordinate file (Figure 1-7), the dimensions of the dataset are stored as floating point numbers. These

numbers appear at the very beginning of the file. The x dimension is the first number, the y dimension is the second number and the z dimension is the third number. Directly after the dimensions, coordinate information is stored. First, all of the x coordinate values are stored. Second, all of the y coordinate values are stored, and third, all of the z coordinate values are stored.

x dimension	y dimension	z dimension
nx * ny * nz X Coordinates		
nx * ny * nz Y Coordinates		
nx * ny * nz Z Coordinates		

Figure 1-7: PLOT3D Coordinate File

The data file (Figure 1-8) contains the data which is found at every node loca-

7 floating point numbers to be skipped over
nx * ny * nz Density Values (float)
nx * ny * nz X-Momentum Values (float)
nx * ny * nz Y-Momentum Values (float)
nx * ny * nz Z-Momentum Values (float)
nx * ny * nz Stagnation Values (float)

Figure 1-8: PLOT3D Data File

tion in the coordinate file. There are five different scalar values in the file. They are found after a header of seven floating point numbers. Each scalar value is stored in a contiguous block as with the coordinate values.

The output target is an AVS field with the form *3D 5-vector float irregular*.

1. After starting up the Network Editor, drag the **file descriptor** module from the **Data Input** module column into the Network workspace with the left mouse button. Once the module has started up, a set of widgets will appear in the AVS control panel. Also, the **AVS Field Description Form** will appear in the middle of the display. This form contains all of the items that need to be filled in order to completely specify an external data file so that it can be converted into an AVS field. Once the field form has appeared, the following steps should be taken.
2. Set the number of input data files by typing **2** in the **Number of Data Files** typein in the AVS control panel.
3. Specify the logical names to be used for the data files. Set the logical name for the coordinate file by entering **coord** in the **Logical Name for File 1** typein. Set the logical name for the data values file by entering **data** in the **Logical Name for File 2** typein.
4. Set the computational space dimensions for the field in the field form description. This is equivalent to `ndim` for a field. Type the value **3** into the **Enter Value** widget.
5. Set the physical space dimensions for the field. This is equivalent to `nspac` for a field. Select the **Dimensions of Physical Space** radio button. Type the value **3** into the **Enter Value** widget.
6. Set the vector length for the field. This is equivalent to `veclen` for a field. Select the **Vector Length** radio button. Type the value **5** into the **Enter Value** widget.
7. Set the data type for the field. This is equivalent to `data` for a field. Select the **Data Type** radio button. Select the **float** radio button.
8. Set the coordinate mapping for the field. This is equivalent to `irreg` for a field. Select the **irregular** radio button.
9. Set the labels for the field. Select the **Labels** radio button. The **Value Selection** browser will appear with six names in it (label 1 through label 5 and all labels). Select **label 1** from the list and then type the value **density** into the **Enter Value** widget. Select **label 2** from the list and then type the value **x-momentum** into the **Enter Value** widget. Select **label 3** from the list and then type the value **y-momentum** into the **Enter Value** widget. Select **label 4** from the list and then type the value **z-momentum** into the **Enter Value** widget. Select **label 5** from the list and then type the value **stagnation** into the **Enter Value** widget.
10. For this example, Units don't have to be set. Both Labels and Units are optional.
11. Set the dimensions for the field. This is equivalent to `dim1, dim2... dimN` for a field. Since the dimensions of a PLOT3D dataset can be found in the coordinate file (`coord`), the dimensions will be read from information within the file. The **Value Selection** browser will appear with four names in it (dimension 1 through dimension 3 and all dimensions). Select **dimension 1** from the list and then select the **File Based Specification** radio button. The default values for **dimension 1** are the correct values, so no changes need to be made. You may want to set a lower value for the **Enter Maximum**

Value typein. This establishes what you think is a reasonable limit for this dimension. If there is something wrong with the way the input instructions parse the input file and a nonsense number is read by mistake, **Enter Maximum Value** may discover it.

12. Continue setting the dimensions for the field. Select **dimension 2** from the list and then select the **File Based Specification** radio button. The second dimension is right after the first dimension in the coordinate file, so enter the value **4** in the **Byte Offset** typein. The value **4** is equivalent to skipping over the first 4 bytes in the data file (4 bytes = size of an integer). Select **dimension 3** from the list and then select the **File Based Specification** radio button. The third dimension is right after the second dimension in the coordinate file, so enter the value **8** in the **Byte Offset** typein.
13. Set the data values for the field. This is equivalent to variable 1, variable 2, variable N for a field. Since the data of a PLOT3D dataset can be found in the data file (data), scalar data values will be specified from information within the file. The **Value Selection** browser will appear with six names in it (scalar_value 1 through scalar_value 5 and all data values). Select **scalar_value 1** from the list and then select the **File Based Specification** radio button. The scalar data in the data file is in floating point format, so you must select the **float** from the data type radio buttons. Select **data** from the input file radio buttons. Since the first scalar value occurs after the first 28 bytes in the file, type **28** into the **Byte Offset** typein. You don't have to change the **Stride** typein since the data is contiguous within the PLOT3D data file.
14. To set the data value for scalar value 2, select **scalar_value 2** from the list and then select the **File Based Specification** radio button. The scalar data in the data file is in floating point format, so you must select the **float** from the data type radio buttons. Select **data** from the input file radio buttons. Since the second scalar value occurs after the first scalar value, you must skip over it. Type **28+(dim1*dim2*dim3*4)** into the **Byte Offset** typein. This equation will be evaluated as the data file is read in and the dataset's dimensions are set.
15. For scalar values 3, 4 and 5, do the same as with scalar value 2. However, the **Byte Offset** should be set to:
 - scalar value 3: $28+(\text{dim1}*\text{dim2}*\text{dim3}*8)$
 - scalar value 4: $28+(\text{dim1}*\text{dim2}*\text{dim3}*12)$
 - scalar value 5: $28+(\text{dim1}*\text{dim2}*\text{dim3}*16)$
16. Set the coordinate values for the field. This is equivalent to coord 1, coord 2, coord N for a field. Since the coordinates of a PLOT3D dataset can be found in the coordinate file (coord), coordinate values will be specified from information within the file. The **Value Selection** browser will appear with four names in it (coordinate 1 through coordinate 3 and all points). Select **coordinate 1** from the list and then select the **File Based Specification** radio button. The coordinate data in the data file is in floating point format, so you must select the **float** from the data type radio buttons. Since the first scalar value occurs after the first 12 bytes in the file, type **12** into the

Byte Offset typein. You don't have to change the **Stride** typein since the coordinate data is contiguous within the PLOT3D coordinate file.

17. To set the coordinate value for coordinate 2, select **coordinate 2** from the list and then select the **File Based Specification** radio button. The coordinate data in the data file is in floating point format, so you must select the **float** from the data type radio buttons. Since the second coordinate value occurs after the first coordinate value, you must skip over it. Type $12+(\text{dim1}*\text{dim2}*\text{dim3}*4)$ into the **Byte Offset** typein. This equation will be evaluated as the data file is read in and the dataset's dimensions are set.
18. For coordinate 3 do the same as with coordinate 2. (See Figure 1-9.) However, the **Byte Offset** should be set to:
coordinate 3: $12+(\text{dim1}*\text{dim2}*\text{dim3}*8)$



Figure 1-9: Setting Up To Read Points

19. The specification for a three dimension PLOT3D file is now complete. In order to read in a sample file, you must specify which data files to read. Select the **Browser for File 1** button in the AVS control panel. Select the file `/usr/avs/data/plot3d/bluntnfx.bin` from the **Select Data File** file browser. Select the **Browser for File 2** button in the AVS control panel. Select the file `/usr/avs/data/plot3d/bluntnfq.bin` from the **Select Data File** file browser.
20. Now that everything is specified, the data can be read from disk by selecting the **send data** button in the AVS control panel. Once the

file has been read, a field will be output on the **file descriptor** module's output port. You can hook this port up to the **print field** module to check out your results or to a complete AVS network with **volume bounds** to check the reading of the coordinate information (the "shape" of the data volume) and other mapper modules to display the data values.

AVS Control Panel Widgets

When the **file descriptor** module is instantiated, two sets of widgets appear. The first set will appear in the AVS control panel as in Figure 1-2.

Select Data File Browser

The top widget is the **Select Data File** browser. This browser serves multiple roles. Depending on which file mode is active, selecting a file from this browser has a different action. For instance, if **read form** is currently active, after choosing a form file from the browser, the file will be read and the form's current parameter settings will be updated.

Read and Write Form Buttons

By selecting either the **read form** or **write form** button, you put the module into a mode where the next file chosen in the browser will be interpreted as a form file. In the read case, the form will be read in and the form's parameter settings will be updated accordingly. In the write case, the form's current parameter settings will be written out to disk in the form file format.

Send Data

When the **send data** button is selected, the previously specified input file(s) is read in based on the current form's parameter settings. A field is created and then output on the module's output port.

Header Information

When the **header information** button is selected, the previously specified input file(s) is read in based on the current form's parameter settings. A dialog box is then displayed with the field header settings for the current input file(s).

Variable List

When the **variable list** button is selected, a list of all standard and user supplied variables is displayed.

Data File Widgets

The data file widgets will vary depending on the number of input files that are needed to describe the data. For instance, an AVS .x image takes only one input file, whereas a PLOT3D dataset requires two separate input files, one for coordinates and one for data. (Note: a maximum of 5 data files can be specified.)

The **Number of Data Files** typein allows you to specify how many data files are required. (I.e., 1 for the AVS .x image and 2 for PLOT3D). When more than one input file is required, a typein and toggle button are created for each file. The typein, **Logical Name for File N** is used to provide a tag name for input files. The toggle, **Browser for File N** is used to set the **Select Data File** browser so that a file selected from the browser is tied to the logical file.

AVS Field Description Form Widgets

The second set of widgets that appear after the **file descriptor** module is instantiated are displayed in a popup panel. This panel is called the **AVS Field Description Form**. Figure 1-3 shows an example of this panel.

Field Component Radio Buttons

In the upper left quadrant of the panel are the field component radio buttons. These radio buttons contain all of the components of a field that need to be described in order to convert data from one format into the AVS field format.

Value Selection Browser

Some of the field components contain subcomponents. For instance, labels, units and data have a total of *vector length* subcomponents. Dimensions have a total of *compute space* subcomponents. Points have a total of *physical space* subcomponents. The **Value Selection** browser allows you to select these subcomponents. The browser appears in the lower left quadrant of the panel.

When specifying any of the subcomponents, use the browser to select the appropriate subcomponent. If all subcomponents are the same, or if they are contiguous within a file, then select the **all subcomponents** value at the bottom of the browser's list.

User Input and File Based Specification Radio Buttons

In the upper right corner of the panel are the **User Input** and **File Based Specification** radio buttons. When a field component is not specified by the external data file, **User Input Specification** should be selected. (e.g. The AVS .x image file does not specify its own vector length. You must enter the value 4.)

When a field component is specified by the external data file **File Based Specification** should be selected. (e.g. The AVS .x image file does specify its own dimensions, so they should be read from the input file.)

Enter Value Typein

When a field component is user specified rather than read from the input file, the **Enter Value** typein appears. This typein widget accepts either numbers, variables or equations. See the description of variables and equations in the section above.

Datafile Format Radio Buttons

When a field component is file based, the datafile format radio buttons appear. These buttons allow you to choose between binary, ASCII, or XDR format files. For instance, by selecting the **binary** radio button, the field component will be read from the file in binary format and converted accordingly. In the ASCII case, the field component will be read in as an ASCII string and converted accordingly.

Binary and XDR

When **binary** or **XDR** format is chosen, two typein widgets appear. The first typein is for **Byte Offset**. This typein specifies exactly how many bytes are to be skipped in the file before the field component is located. The second typein is for **Stride**. **Stride** is used to specify a skip factor between each subcomponent. For instance, in the AVS .x image file, data is stored as alpha, red, green, blue. In order to read in all of the alpha values, you would need to specify a stride of 4.

ASCII

When **ASCII** format is chosen, a set of radio buttons and typein widgets appear. The radio buttons allow you to select how to search forward in the file to get to the field component.

In the **byte offset** case, the two typeins are the same as in the binary and XDR cases. See the description above.

In the **Line #** and **Word #** case, four typeins appear. The **Line #** typein is used to specify an exact line number to go to in the file. The **Word #** typein is used to specify which word in the line to go to. The **Comment Character** typein is used to specify the character that begins comment lines. If the value for this typein is not empty, then whenever comment lines appear, they will not be counted as a line and will be skipped over. The **Stride** typein is used to specify a skip factor between each subcomponent. See Figure 1-10.

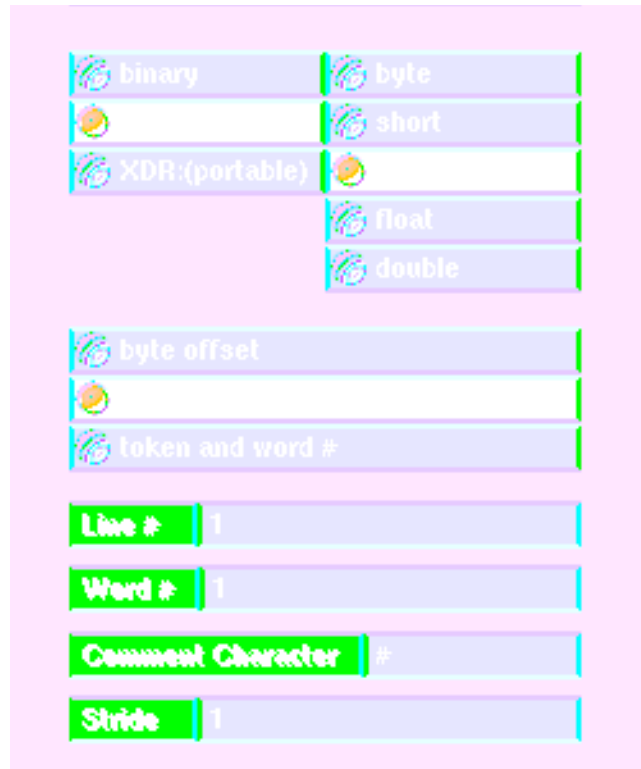


Figure 1-10: Line # and Word # Control Panel

Let's look at an example. Suppose the data file looks like this:

Table 1-2. Sample ASCII File

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

To read all the **A** data using the **Line #** and **Word#** mode, you would set the **Line #** to be 2, the **Word #** to be 4, and the **Stride** to be 5. To read all the **B** data you would set the **Line #** to be 2, the **Word #** to be 5, and the **Stride** to be 5.

In the **Token** and **Word #** case, three typeins appear. The **Token** typein is used to specify a string to look for in the file. This string must appear at the beginning of a line. The **Word #** typein is used to specify which word in the line to go to. The **Stride** typein is used to specify a skip factor between each subcomponent. See Figure 1-11.. The **Token** and **Word#** option is more useful for a file that has names defining the data elements. This is similar to the AVS field file header:

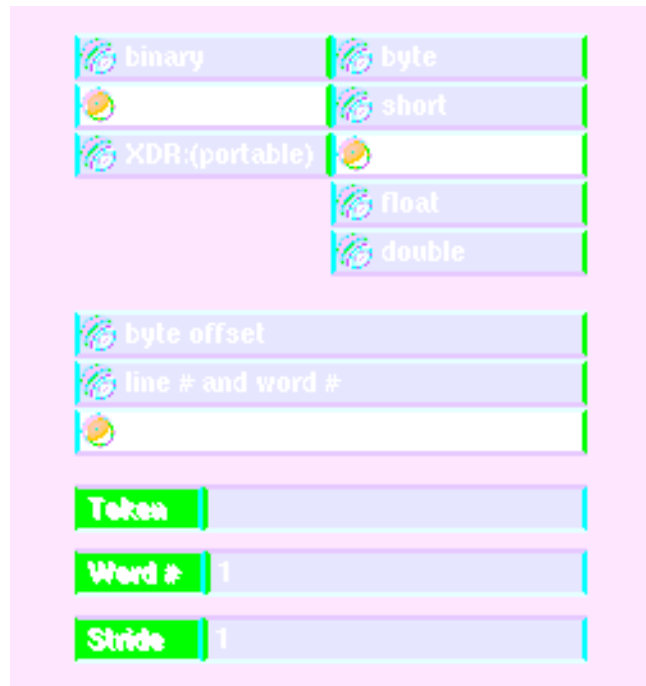


Figure 1-11Token and Word # Control Panel

```
# AVS field file
# this is a header file for a field to be
# used in conjunction with the build a field module of AVS
#
ndim = 3
im1 = 64
dim2 = 64
dim3 = 64
nspace = 3
veclen = 1
data = byte
field = uniform
```

In order to read the number of dimensions (**ndim**), you would set the **Token** to be 'ndim', the **Word#** to be 3, and the **Stride** is irrelevant. In this example, you could also read the number of dimensions using the **Line#** and **Word #** mode by setting the **Comment Char** to '#' (this skips over the comment lines), the **Line #** to 1, and the **Word#** to 3. Again, for reading a single element, the **Stride** is not relevant.

Data Type Radio Buttons

The **Data Type** radio buttons specify what format the data is when read from the input file. For instance, in PLOT3D files, all data and coordinates are written out in floating point format.

Logical File Radio Buttons

Whenever a data for a particular file format comes from multiple files (e.g., PLOT3D), a set of radio buttons will appear with the logical names for the input files as settings. When specifying a field component, one of these logical names must be chosen.

data dictionary Module

The **data dictionary** module allows you to read in external data files once a form has been created and saved by the **file descriptor** module. A form contains all of the information necessary to convert a file on disk into an AVS datatype.

When the **data dictionary** module is instantiated, one set of widgets appear. The widgets will appear in the AVS control panel as in Figure 1-12.



Figure 1-12: Data Dictionary Control Panel

Select Data File Browser

The top widget is the **Select Data File** browser. This browser serves multiple roles. Depending on which file mode is active, selecting a file from this browser has a different action. For instance, if **read form** is currently active, after choosing a form file from the browser, the file will be read and the form's current parameter settings will be updated.

Read Form Button

By selecting either the **read form** button, you put the module into a mode where the next file chosen in the browser will be interpreted as a form file. In the read case, the form will be read in and the form's parameter settings will be updated accordingly.

Send Data

When the **send data** button is selected, the previously specified input file(s) is read in based on the current form's parameter settings. A field is created and then output on the module's output port.

Header Information

When the **header information** button is selected, the previously specified input file(s) is read in based on the current form's parameter settings. A dialog box is then displayed with the field header settings for the current input file(s).

General Order of Operation

To use **data dictionary**, proceed through the interface in this order:

1. Press the **read form** button. This attaches the file browser to the read form function.
2. Use the Select Data File browser to specify a data form file. Upon selecting or typing in a filename, the data form will be read.
3. Data forms require one or more input files. For example, there may be one input file containing data, and another input file containing coordinate information. The number of input files required is shown by the number of **Browser for File *n*** buttons.

For each input file required, press **Browser for File *n*** and then use the Select Data File browser to establish which actual file corresponds to file *n*. Work down the list establishing these logical file to real file correspondences. No data will be read yet.

4. If you wish, examine the contents of the data form with the header information function. If the data form specifies that part of the input parsing instructions will come from the input file itself (e.g., the dimensions of the data), then the input files(s) will be read in at this point according to the correspondences established in step 3.

5. When all logical file to real file correspondences have been defined, press the **send data** button to actually read the input data file(s) and convert it to an AVS field using the rules in the data form.

THE AVS MODULE GENERATOR

Overview

The **AVS Module Generator** is an AVS module that generates skeleton source code, Makefiles, and documentation templates of AVS modules. It is intended to make it easier to create, test, and maintain AVS modules. The Module Generator can generate module skeletons in C or FORTRAN for both coroutines and subroutines. It allows you to add new input/output ports and parameters to modules, and perform software engineering functions such as compiling modules, loading them into AVS, debugging, and modifying them. The Module Generator will produce almost all of the AVS-specific code such as declaration of ports and parameters, module initialization functions, and so on. It also creates areas in the source code for users to place their own routines which make the module fully functional.

Although one purpose of the Module Generator is to free users from having to consult the documentation, in order to understand what the Module Generator is providing you, you will probably want to familiarize yourself with the "AVS Modules" chapter and the "AVS Routines," "Examples of AVS Modules," "FORTRAN Fields," and "Geometry Library" appendices of the *AVS Developer's Guide*.

General Module Structure

One of the things that makes a program like the Module Generator possible is that AVS modules generally have a fixed structure. The Module Generator takes advantage of this by creating source code with that structure. There are two flavors of AVS modules: subroutines and coroutines (see the "AVS Modules" chapter in the *Developer's Guide*). The following sections will describe the assumptions that have been made for each of these structures and how the Module Generator takes advantage of these assumptions.

Subroutine Modules

For our purposes, an AVS subroutine module has this structure:

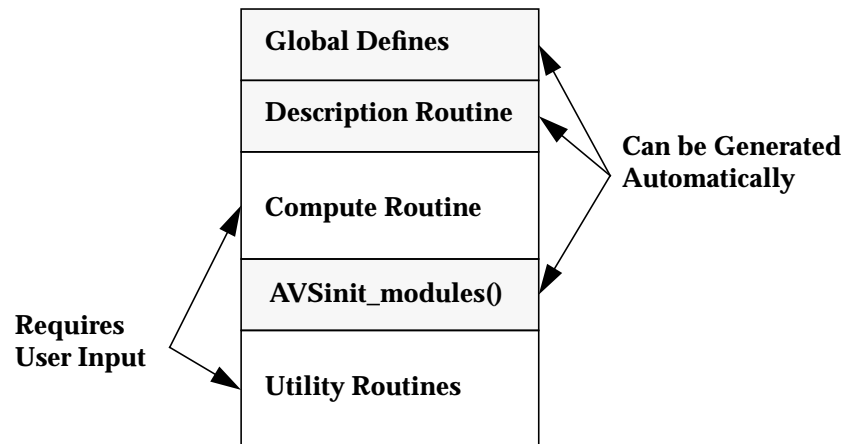


Figure 2-1: Subroutine Module Structure

In the simplest case, there are no global defines or utility routines. In this case the Module Generator will produce all the remaining source code. Here is the simplest AVS subroutine module, (in C), that the Module Generator generates. This example has no ports or parameters declared.

```
/* mod_gen Version 1 */
/* Module Name: "simple" (Input) (Subroutine) */
/* Author: Author's Name */
/* Date Created: Sun Mar 1 11:08:37 1992 */
/* */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as */
/* mod_gen needs it in order to read module sources back in. */
/* */
/* End of Module Description Comments */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, Etc.*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1 */

/* *****/
/* Module Description */
/* *****/
int simple_desc()
{
    int in_port, out_port, param;
    extern int simple_compute();
```



```

AVSset_module_name("simple", MODULE_DATA);
AVSset_compute_proc(simple_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2 */
return(1);
}

/* *****/
/* Module Compute Routine */
/* *****/
int simple_compute()
{
/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY) */
/* <---- END OF USER-SUPPLIED CODE SECTION #3 */
return(1);
}

/* *****/
/* Initialization for modules contained in this file. */
/* *****/
int ((*mod_list[])) = {
simple_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, UTILITIES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4 */

```

Here is the same module written in FORTRAN:

```

C mod_gen Version 1
C Module Name: "simple" (Input) (Subroutine)
C Author: Author's Name
C Date Created: Sun Mar 1 11:15:17 1992
C
C This file is automatically generated by the Module Generator (mod_gen)
C Please do not modify or move the contents of this comment block as
C mod_gen needs it in order to read module sources back in.
C
C End of Module Description Comments

C *****/
C Module Description
C *****/
integer function simple_desc()
implicit none
include 'avs/avs.inc'

integer in_port, out_port, param
external simple_compute

```

General Module Structure

```
integer simple_compute

call AVSset_module_name('simple', 'data')
call AVSset_module_flags(single_arg_data)
call AVSset_compute_proc(simple_compute)
C ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)
C <---- END OF USER-SUPPLIED CODE SECTION #2
simple_desc = 1
return
end

C *****
C Module Compute Routine
C *****
integer function simple_compute()
implicit none
include 'avs/avs.inc'

C ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)
C <---- END OF USER-SUPPLIED CODE SECTION #3
simple_compute = 1
return
end

C *****
C Initialization for modules contained in this file.
C *****
subroutine AVSinit_modules
include 'avs/avs.inc'

external simple_desc
integer simple_desc
call AVSmodule_from_desc(simple_desc)
end

C ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, UTILITIES)
C <---- END OF USER-SUPPLIED CODE SECTION #4
```

These are fully functioning modules; you can compile, load, and run them. If you were to do so, they would create an icon with the name "simple" in the module palette with no input ports or output ports. You can drag it down to the execution area and it would start running, but would have no parameter widgets.

There are several interesting things to note about these examples.

- The Module Generator distinguishes between "USER-SUPPLIED" and automatically synthesized regions. One necessary section of code is the comment block at the top of the file. This contains the information about the module which is used to produce the template code. This includes the module name, author, and date, as well as descriptions of the module structure. Except for the contents of the "USER-SUPPLIED" sections, this comment block is the only thing that the Module Generator needs to create a module source skeleton from scratch.

- There are several "USER-SUPPLIED" code regions. Any code that is inserted in these regions is retained unchanged from one generation of the source code to the next. There are USER-SUPPLIED sections in each of the Global Defines (C only), module description, module compute, and utility routine areas. There is more detail on the USER-SUPPLIED code regions later in this document.
- The Module Generator provides calls to `AVSinit_modules()` and `AVSmodule_from_desc()`. AVS expects to find these routines when building modules.

Coroutine Modules

AVS coroutine modules are like subroutine modules except that they can run asynchronously from the rest of the AVS modules in a network. For this reason, they must have their own `main()` routines and must process port and parameter arguments in a different fashion. The overall structure of an AVS coroutine is:

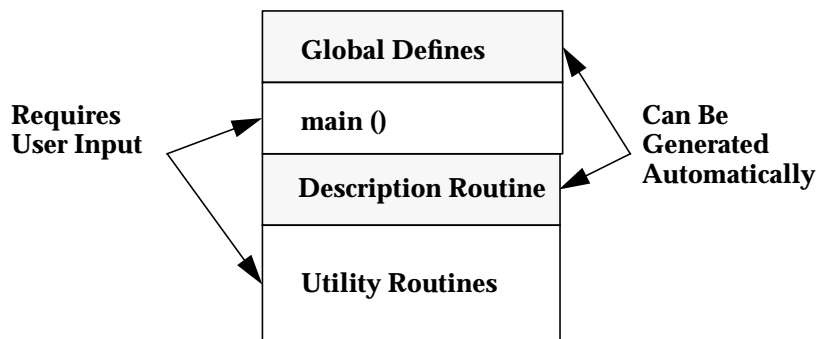


Figure 2-2: Coroutine Structure

The main program loop is put into the `main()` routine which also calls `AVScorout_init()` with the name of the Description Routine.

Here is the example (in C) from before, but expressed as a coroutine:

```

/* mod_gen Version 1 */
/* Module Name: "simple" (Input) (Coroutine) */
/* Author: Author's Name */
/* Date Created: Sun Mar 1 11:15:46 1992 */
/* */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as */
/* mod_gen needs it in order to read module sources back in. */
/* */
/* End of Module Description Comments */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>

```

Example Session

```
/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, Etc. */
/* <---- END OF USER-SUPPLIED CODE SECTION #1 */

/* *****/
/* Coroutine Main Routine */
/* *****/
int main(argc, argv)
int argc;
char **argv;
{
    int simple_desc();
    AVScorout_init(argc,argv,simple_desc);
}

/* *****/
/* Module Description */
/* *****/
int simple_desc()
{
    int in_port, out_port, param;
    AVSset_module_name("simple", MODULE_DATA);

/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2 */
    return(1);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, UTILITIES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4 */
```

Example Session

Before delving into the specifics of the user interface, let's look at an example in which we will write a version of the **threshold** module in C.

This module will input an integer field, then scan through the data portion of the field looking for data elements that lie outside of a specified range. If a data element lies outside that range, the module will set the data value to 0. The module will output the modified field structure.

This example will have two ports: an input port which will be an integer field and an output port which will also be an integer field. It will have two parameters—two integer dials used to set the range of the data. One dial specifies the minimum value of the data and the other one specifies the maximum data value.

Initiating the Module Generator

The Module Generator module is found in the Network Editor's **Data Output** column in the AVS Supported module library. It has no inputs and no outputs. It is instanced (as are all AVS modules) by selecting the module with the

left mouse button and dragging it into the main Workspace. When it starts up the control panel in Figure 2-3 is displayed.

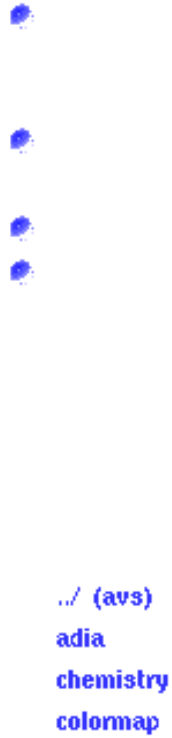


Figure 2-3: Module Generator Main Control Panel

Specifying the Module's Structure

The top half of this control panel controls the module's contents, while the bottom half directs the actions associated with the source code. We'll work our way down the control panel, "filling in the blanks" along the way.

The first item to fill in is the module name. Since we want to generate a **threshold** module, we first type **Ctrl-U** in the **Module Name** widget (to clear it) and then type the name **threshold**. Module names must start with a character but can contain blanks, digits, and underscores.

We want to make our module a **Filter** module because it processes a field into a new field. In general, modules which either import or create data reside in the **Input** column. Modules which input and output the same kind of data reside in the **Filters** column. Modules which process data and create geometries from that data live in the **Mappers** column and modules which output data to the screen or to disk live in the **Output** column.

To make our module a **Filter**, we select the **Filter** option from the following choices: **Input, Filter, Mapper, Output**. The **C** option is the default option so we'll leave it alone. We want our module to be a subroutine, which is also the default so we'll also leave that option alone.



Figure 2-4: Control Panel for Threshold Module

Now we need to describe this module's input and output ports. For simplicity's sake, let's assume that the threshold module will work only on integer fields, i.e., it will accept an N-dimensional integer field, with any number of vector elements per node, and produce a field of the same type with the values thresholded to a specific range. The module can be made more general purpose, but this requires more processing stages in the compute routine; you must have code which analyzes the field structure and does selective processing based on the data type, vector length, etc.

We will use the port editors to specify the port information. When we select the **Edit Input Port** button, two new windows pop up on the screen: the **Input Port Editor** and the **Field Editor**. (See Figure 2-5 and Figure 2-6)



Figure 2-5: Input Port Editor

Notice that there are six available ports named "Unused 0" through "Unused 5". These ports are "turned on" by changing their names from "Unused *n*" to the name of the port. For instance, we will declare port 0 to be named "input" by typing that name in the Input Port Name window. This is set up to be both a required port and an AVS field port as the default. To turn ports off, make the name of that port be "Unused". We can specify that we want to limit this port to only accept integers by selecting the **integer** option under the **Data Type** menu in the Field Editor window.

The Module Generator Port Editors have buttons for all valid AVS port types including: field, ucd, geom, colormap, pixmap, molecule, integer, real, string, user defined data (user_data), and upstream geometry and transformation information. Integer, real, and string data types can be either ports or parameters. Parameters are generated and controlled internally by the module while ports get their information from other modules.

The output ports for our module are specified in a similar manner: select the **Edit Output Port** option on the main control panel, name the selected output port in the **Output Port Editor**, and then refine the selection using the **Field**

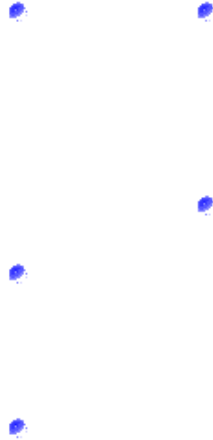


Figure 2-6: Field Editor

Editor. In this case, we choose the **Data Type** to be of type **integer**. This will produce code for an N-dimensional, integer field of unspecified vector length.

Finally, the parameters and user interface widgets controlling the parameters are specified. This is done by selecting the **Edit Parameters** button found on the main control panel. Doing this causes any other open panels to close down and the **Parameter Editor** control panel to appear (Figure 2-7). The mechanism for "turning on" specific parameters is similar to that of ports, except the Module Generator provides up to 100 parameters and only six input and output ports. To activate a parameter, first select the parameter in the browser labelled "Parameter" and then give the parameter a name other than "Unused". You can deactivate unwanted parameters by naming them "Unused".)

There is a two-stage process for selecting the widget type which will be created. The first stage is to decide what type of parameter you wish to use. The choices are: integer, real, string, string_block, choice, color_editor, and track. Once the type of parameter is selected, choices for widgets for that parameter type are displayed. Here is a list of the widget choices associated with each parameter type:

- **integer:** idial, islider, typein_integer, toggle, tristate, oneshot
- **real:** dial, slider, typein_real
- **string:** typein, text, browser, text_browser
- **string_block:** text_block_browser, textblock
- **choice:** radio_buttons, choice_browser



Figure 2-7: Parameter Editor

- **color_editor:** (no choices)
- **track:** (no choices)

We will first choose Unused 0, name it **Min Value**, select **integer** (the default is **idial**). We can now set the range and default value for the dial. The defaults are for a dial to go from 0 to 1 with an initial value of 0. For the sake of this example, let's assume that the range of data will be from 0 to 100. We will set the **Integer Max** for the **Min Value** idial to be 100.

The second parameter we want for our module is the maximum threshold. We choose Unused 1, name it **Max Value**, select **integer**, set **Integer Max** to be 100, and the **Integer Default** to be 100.

Creating an Executable Module

Now, the module is fully specified and it is time to write the source code out to disk. We specify that the source code will live in a file called `/tmp/threshold.c` by using the file browser called **Source File**. Type the name `/tmp/threshold.c` into the dialog box that appears when you hit the button labelled **New File** in the **Source File** browser. Write out the source code skeleton by hitting the button labelled **Write Source**.

Adding Code to the USER-SPECIFIED CODE SECTIONS

At this point, the source skeleton has been written out to disk and we can add in the "guts" of the compute routine. We hit the **Edit** button which causes a text editor to appear with the module source in it.

Now, we add in the code that does the work of the routine. We will add two pieces of code to make this module work. The first is a piece added to the description routine which causes the output field to be automatically generated as the same size as the input field. This line looks like

```
AVSinitialize_output(in_port, out_port);
```

Now in the compute routine, the following code is added between the comments describing the USER-SUPPLIED CODE SECTIONS:

```
int i, j, k;
for (k = 0; k < MAXZ(input); k++)
  for (j = 0; j < MAXY(input); j++)
    for (i = 0; i < MAXX(input); i++)
      if (I3D(input, i, j, k) > Max_Value) {
        I3D(*output, i, j, k) = 0;
      } else if (I3D(input, i, j, k) < Min_Value) {
        I3D(*output, i, j, k) = 0;
      } else {
        I3D(*output, i, j, k) = I3D(input, i, j, k);
      }
}
```

The module is compiled by hitting the **Compile (opt)** button. It is then loaded into AVS by hitting the **Load** button. At this point, the module **threshold** appears in the **Filters** column of the AVS Network Editor. We can drag it down into the workspace and test it.

To write a FORTRAN version of this module, we simply change the language button from **C** to **Fortran** and hit the **Write Source** button again. The Module Generator informs us that it is changing the file suffix to ".f" and the file */tmp/threshold.f* is written out to disk. At this point we can do what we did above: edit the file to add the inner loop, compile, load, and test it.

Adding Code to the USER-SPECIFIED CODE SECTIONS

If the Module Generator only wrote an initial module skeleton, it would be of limited utility. What is really needed is a tool which allows you to write a module, read it back in, add new ports and/or parameters and write it back out again without disrupting the work which you've already done to it. To accomplish this, we have added USER-SPECIFIED CODE SECTIONS to the module structure. These sections are surrounded by comments which look something like this:

```
/* ----> START OF USER-SUPPLIED CODE SECTION #1*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1 */
```

In FORTRAN, these comments look like:

```
C ----> START OF USER-SUPPLIED CODE SECTION #1
C <---- END OF USER-SUPPLIED CODE SECTION #1
```

Anything you add between these comment pairs is preserved and written back out when the source code is next written.

There are four such regions in a subroutine module and three in a coroutine module. In a subroutine module, there are USER-SPECIFIED areas at the top for header information, in the description routine, in the compute routine, and at the bottom of the module for utility routines. In a coroutine module, there are sections at the top for defines and includes, in the main routine, and at the bottom for utility routines.

Hints

The Module Generator has a lot of knowledge built into it regarding the freeing and allocation of memory for fields, geoms, and ucd structures. It normally does not give you these hints, but there is a button in the main menu called **Include Hints** which, when selected, causes this information to be put into the compute routine of the module. Once you have hit the **Include Hints** button, the module may not be compilable and may need some tinkering with.

For **field** structures, the hints give you the code fragments for freeing old field structures and allocating new ones. One of the problems associated with field structures is that even though modules can accept any kind of input field, they can only output one specific field at a time. Although the port may be very general (a module can output a float field on one pass and an integer field on the next pass), the output field for any specific output must be completely specified. This means specifying the dimensions, physical space, data type, vector length and uniform qualities. For instance, trying to allocate a "field integer" will fail whereas allocating a "field integer 3D 3-space uniform 1-vector" will succeed. If you completely specify the field using the Module Generator's Field Editor, no more work is required and the code generated by the hints is complete. Less specified fields will cause a comment to be printed like:

```
/* YOU MUST FILL IN THE FOLLOWING "field" STRING */
/* WITH A MORE SPECIFIC DESCRIPTION! */
```

For **geom** outputs, once the type of geometry is selected (from the choice of Mesh, Polyhedron, Polytri, Spheres, and Label), all appropriate code for creating objects and edit lists, adding objects to the edit list, and destroying objects are created. In all cases, there may be several different ways of creating an object. All the possible choices are shown, and the inappropriate ones should be edited out.

For **ucd** structures, the hint code which is generated has much to do with freeing the old structure, allocating the new one, and using the supplied access routines to set up node positions, extents, labels, and so on.

Detailed Description of Controls

The hints are especially useful for coroutines. Most coroutines have the same structure: the main program is an infinite loop which contains an `AVScorout_wait()` call and then calls to `AVScorout_input()` and `AVScorout_output()` to process ports and parameters. Since this is not the only possible structure for coroutines, it is only included in the code when the **Include Hints** button is selected.

Detailed Description of Controls

This section goes through each panel and describes in detail each collection of controls as they will be encountered.

Top-Level Controls

The main control panel is divided vertically into two parts. The top set of controls is used to specify the *contents* of the module to be generated. The bottom set of controls is used to generate the module sources and control the editing, compiling, loading and debugging of the module.

The vertical order of the controls in the top-level menu corresponds to the process of generating modules. The typical way to use the Module Generator is to start at the top of the top-level menu and work down the screen completing each function on the way.

The overall process of using the Module Generator can be thought of in these terms:

```
Specify the module's structure
Write out the source
Edit the module to add your own code
Compile the module
Load the module into AVS
Test/Debug the module
Document and Maintain the module
```

Typically the Test/Debug phase is a development cycle which looks something like: write out the source, add your own code, try compiling, fix the compilation errors, compile again, load the executable into AVS, run the module with the debugger, find the bugs, re-edit the source, maybe read the source back in and add some new ports or parameters, write the source back out, re-edit it, re-compile it, re-test it, and so on until the module is working properly.

Module Description

These controls (Figure 2-8) are used to specify the contents of the module. They include controls for indicating the module name, language, style, as well as controls for editing input and output ports and specifying module parameters

Module Name

The **Module Name** is a character typein. The default is \$NULL. You must backup over this or type **Ctrl-U** before inserting your own name. Names must start with a letter, but can contain digits, spaces, or underscores (_). You must supply a module name if you intend to write a module's source code out. If you are reading a module's source code into the Module Generator, then this field will be filled in automatically.



Figure 2-8: Module Description Controls

Module Type

The **Module Type** is a choice-type selector with the following choices: **Input**, **Filter**, **Mapper**, **Output**. The default is **Input**. This indicates the column in the AVS Network Editor where the module will appear when it is loaded into AVS. Whatever is selected here does not affect the functionality of the module.

C vs. FORTRAN

The language controller offers a choice between **C** and **Fortran**. The default is **C**. One of the interesting features of the Module Generator is that you can read a module that is written in **C** and write it out in **FORTRAN** (provided that they have the Module Generator header information built into them). Everything but code inserted in the "USER-SUPPLIED" areas is translated automatically.

Subroutines vs. Coroutines

The Module Generator can generate skeleton source code for either AVS **Subroutine** (synchronous) or AVS **Coroutine** (asynchronous) modules. The default is to generate **Subroutine** code. One of the main differences between these forms is that the coroutine module has a **main()** routine in source for the module. Subroutine module are linked with a **main()** routine found in the AVS libraries.

Editing Ports and Parameters

The buttons labeled **Edit Input Port**, **Edit Output Port**, and **Edit Parameters** are used to bring up sub-menus. These appear to the right of the control panel on the screen. Each of these sub-menus will be discussed in detail in following sections. Selecting any one of these options causes the others' sub-menus to be hidden and the relevant sub-menus for the currently selected option to be displayed.

Unix Specification Tools

The remaining top-level controls (Figure 2-9) are used to specify a variety of actions you can perform on the source code. These sub-menus include reading and writing the source files, generating Makefiles, generating manual pages, editing the source, compiling the module, loading it into AVS, and bringing up a window with the debugger in it.



```
../ (avs)
adia
chemistry
colomap
```

Figure 2-9: UNIX Specification Tools

Source File Name

A name for the module's source code file must *always* be specified, regardless of whether you are reading the module in or writing one out. A standard AVS file browser is supplied for this purpose. Note that this one browser is shared by both the **Write Source** and **Read Source** options. Files are not written or read until these buttons are pressed.

Include Hints

Normally hints are not included in the source code as it is written out. To have them be included, select the **Include Hints** button prior to writing out the source code.

The Module Generator has a lot of knowledge built into it about how to allocate and free fields, how to build geom edit lists, and how to create ucd structures. It only does this for fields, geoms, and ucds which are specified as output ports. A "hints" area in the compute routine is written out which contains the sample code. These lines of code make suggestions about appropriate ways to free and allocate data. If you want to use these lines of code, simply transfer them to within the "USER-SUPPLIED" code sections referred to earlier.

Writing Module Source Files

In order to write out a module's source code, you must specify both a module name and a valid source file. Optionally, you can also specify various ports and parameters you want associated with the module. If the source file specified already exists, an AVS warning window will appear asking whether or not you want to overwrite the existing file.

Note: A special case occurs when you are modifying an existing source file. Typically, the process is to write out a first instance of a module, edit it, read it back in, modify the ports or parameters, and write it out again. If you forget to read the source back in, you will overwrite the original file and lose whatever changes you have added to the source.

Appropriate suffixes are added when they are not already supplied. When you are generating a C file, if you did not specify a *.c* suffix for the file one will be added for you. Likewise, *.f* suffixes are added when not supplied for Fortran files. If you are switching from C to Fortran, the Module Generator will notify you that it is also changing the suffix.

Reading Module Source Files

Reading in a module's source code only requires two things: (1) the **Source File** name must be specified; and (2) the file must be a valid file previously generated by the Module Generator. Valid files are identified by the comment block at the top of the file. Here is a sample header block:

```
/* mod_gen Version 1 */
/* Module Name: "read 16 bit image" (Input) (Subroutine) */
/* Author: Author's Name */
/* Date Created: Tue Nov 19 13:12:35 1991 */
/* */
/* This file is automatically generated by the Module Generator
(mod_gen)*/
/* Please do not modify or move the contents of this comment
block as */
/* mod_gen needs it in order to read module sources back in. */
/* */
/* output 0 "output" field 2D 2-space 1-vector uniform integer*/
/* param 0 "skip bytes" typein_integer 0 0 1 */
/* param 1 "width" typein_integer 0 0 1 */
/* param 2 "height" typein_integer 0 0 1 */
/* param 3 "filename" browser */
/* End of Module Description Comments */
```

In this header, the first line contains the magic phrase `/* mod_gen Version` followed by the current Module Generator version number. The next line is scanned to pick up the module name (in quotes), its type, and its synchronicity. The next seven lines must be there, but are skipped over. Then the Module Generator proceeds to parse each line until it encounters the line containing `/* End of Module Description Comments`. Each parsed line is a description of the ports and parameters which have been defined for this module.

After a file has been read in, it is assumed that it will be modified and written back out. To this end, the file is kept open and when it is written back out, the Module Generator scans the input file looking for lines containing comments labeled something like:

```
/* ----> START OF USER-SUPPLIED CODE SECTION #1 */
/* <---- END OF USER-SUPPLIED CODE SECTION #1 */
```

Any code found between these comment pairs will be included in the output file. This is the mechanism that allows you to add your own code into the modules, read it back in, modify it and write it out while preserving your modifications. There are four such "user reserved" sections in the average C module: one for global definitions, include files, etc., one in the module description routine, one in the module compute routine, and one at the end of the file for including utility and auxiliary routines. There are three such areas in the average FORTRAN module (since there is no global definition area).

Writing Makefiles

UNIX Makefiles are written whenever the **Write Makefile** button or either of the **Compile** buttons are hit. The Makefiles are written into the same directory as the source code and are appended with the module's executable name. For instance, if your module name is **fred** and you call the source file `/tmp/fred.c`, then the Module Generator will write the Makefile for **fred** into `/tmp/Makefile.-fred`. This can then be used by typing:

```
make -f Makefile.fred
```


Likewise, if your module name is "read 16 bit data" and the source file is named `/user/joe/read_16_bit_data.c`, then the Makefile will be written into `/user/joe/Makefile.read_16_bit_data`.

Makefiles make use of `/usr/avs/include/Makeinclude` file which differs from hardware system to hardware system. This is how the Module Generator is able to correctly compile modules on each different platform.

Writing Manual Pages

The **Write Man Page** button is used to generate template manual pages in the same way that the **Write Source** button generates template module source files. Like the Makefiles, the man pages are written in the same directory as the source code, using the source code file prefix as the prefix for the man page as well. The suffix `.txt` is appended to the prefix to create the man page name. As in the above example, if the source file is `/user/joe/read_16_bit_data.c`, then the man page template will be written into `/user/joe/read_16_bit_data.txt`.

The manual page is divided into standard AVS manual page sections: Name, Summary, Description, Inputs, Outputs, Parameters, Example Networks, Related Modules, and See Also. The Module Generator fills in these categories as best it knows how.

The **Write Man Page** feature has two options: generating pre-formatted pages and generating *nroff* or *troff* format man pages. The default is to generate man pages in the pre-formatted format. To turn on the *troff* formatted man pages, you need to set the environment variable `AVS_MG_TROFF` prior to starting up AVS. Then, when the **Write Man Page** option is selected, the *troff* formatted files will be generated rather than the pre-formatted files.

Compiling Modules

Compiling modules is accomplished by using either of the **Compile** buttons in the UNIX Specification Tools. When you ask the Module Generator to do this, the following actually happens:

1. Writing the Makefile for the executable in the directory in which the source code is written.
2. Writing a script file containing a *make* command to `/tmp/mod_gen_compile`.
3. Creating an *xterm* (in the middle of your screen) which executes that script. The temporary script file, `/tmp/mod_gen_compile`, is removed after the compilation process is complete.

Here is a typical `/tmp/mod_gen_compile` script file:

```
echo Compiling /tmp/fred.f...
cd /tmp/; make -f /tmp/Makefile.fred
@ i = $status
if ( $i ) then
echo There were compiler problems encountered! No executable was
written!
```

```
else
echo No compiler problems were encountered!
endif
echo Type RETURN to exit...
set i = $<
```

The two options are to compile the module with the debugging flags turned on or with the optimization levels for your particular platform turned on. This is accomplished by writing out different Makefiles. In the debug case, the Makefile includes the `-g` option which disables any optimization flags which may have been previously specified. In the optimized case, the `-g` compiler flag is omitted. **NOTE:** you should compile the module with the debug flags turned on if you intend to debug the module later.

The compile options uses `xterm` to run the compilation in. On some platforms, either there is no `xterm` program or `xterm` is not in the default path. To work around this inconsistency, the Module Generator checks for the environment variable: `AVS_XTERM`. If this variable is set, it uses what the variable is assigned to rather than `xterm`. For instance, on the SUN SPARC station, you will probably want to set the `AVS_XTERM` variable to `/usr/openwin/bin/xterm`. You must exit AVS to do this.

After the module has completed compiling, the compilation status is displayed in the `xterm` and you must hit the **Return** key on the keyboard to make the `xterm` go away. If there were errors reported, you will probably want to keep this window around while you re-edit the file and fix the errors. The Module Generator waits until the compilation process is complete and you have made the `xterm` go away.

Editing a Module's Source Code

The **Edit** button causes an `xterm` running a text editor (with the module source code loaded into it) to appear in the middle of the screen. If you have the UNIX environment variable `EDITOR` defined, the Module Generator will use your defined editor, otherwise, it will default to popping up an `xterm` running the `vi` text editor.

The **Edit** option uses `xterm` to run the selected editor in. On some platforms, either there is no `xterm` program or `xterm` is not in the default path. To work around this inconsistency, the Module Generator checks for the environment variable: `AVS_XTERM`. If this variable is set, it uses what the variable is assigned to rather than `xterm`. For instance, on the SUN SPARC station, you will probably want to set the `AVS_XTERM` variable to `/usr/share/lib/xterminfo/x/xterm`. You must exit AVS to do this.

The editor is run in background mode, so you can continue to do things with the Module Generator without exiting the editor. In order to make the editor window disappear, you must exit the editor.

Loading a Compiled Module into AVS

There are several ways to load a successfully compiled module into AVS. These include:

- Using the **Read Module(s)** button in the **Module Tools** section of the Network Editor.
- Typing: `mod_read module_name` into the AVS command line interpreter (CLI).
- Hitting the **Load** button found at the bottom of the Module Generator main control panel. An error is reported if a valid, executable module cannot be found.

Debugging a Module

The final software engineering tool is the **Debug** option for the Module Generator. When this button is hit, an *xterm* appears in the middle of your screen running `avs_dbx` on your module executable. Note that `avs_dbx` varies from hardware system to hardware system as to which debugger it will run. Please see the "AVS Modules" chapter of the *AVS Developer's Guide* for more information on using `avs_dbx`.

Port Editing

Ports for AVS modules are limited to a few cases. The Port Editors provide tools for specifying what type each port should be. The only difference between the menus for editing Input Ports and editing Output Ports is that Input Ports can be flagged as being either **required** or **optional**.

Ports are specified in a three step process: activating the port by giving it a name; specifying the port name; and refining the description. Currently, the Module Generator allows you to specify up to six input and six output ports. Inactive ports have the names "Unused 0" through "Unused 5". To activate a port, first choose one of the six ports presented and give it a name. Like module names, valid port names must start with a character and can contain digits, spaces, and underscores (`_`). To disable a previously activated port, give the port the name "Unused". The Module Generator checks to see if the name you have supplied is currently in use by another port or parameter.

After you have selected a port and given it a unique name, you must choose what data type this port will input or output. This is done with the radio buttons found at the bottom of the Port Editor. The ports can be selected from: `field`, `ucd`, `geom`, `colormap`, `pixmap`, `molecule`, `integer`, `real`, `string`, `user_data`, `upstream_geom`, and `upstream_transform`. Because you can further refine the specification for **fields** and **geoms**, additional editor control panels will appear whenever these options are selected.



Figure 2-10: Input Port Editor

The other data type which requires more information to be supplied is **User Defined Data**. Whenever a port is selected to be **user_data**, you must further specify (a) the name of the structure which is to be used and (b) a (absolute path) file name which contains that structure. This information is provided in two pop-up typeins: for input data these are labelled **IUData Struct Name** and **IUData File Name**; for output ports these are labelled **OUData Struct-Name** and **OUData File Name**. Each input and output port may have a unique user data structure associated with it.

Field Editor

For field ports, you may want to further describe the field type. This is done for two reasons: to limit the type of data you want the module to accept or produce; and, in the case of output fields, to give the Module Generator enough information about the nature of the field to produce "hint" code showing possible allocation strategies. (See Figure 2-11.)

There are five possible variables for fields which can be specified: the number of dimensions; the physical space; the data type; the vector length; and the spacing of the data. A more detailed description of what each of these represent can be found in the "AVS Data Types" chapter of the *AVS Developer's Guide*. When left "Unspecified", no restrictions are put on the field.

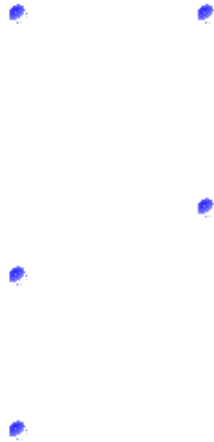


Figure 2-11: Field Editor

Suppose a module is to accept only image data (such as the **display image** module). Image data is considered to be *2D 4-vector uniform byte* data. The port would be specified as a **required field** port whose **Dimensions** are set to 2, **Physical Space** is 2, **Data Type** is *byte*, **Vector Length** is 4, and **Spacing** is *uniform*. In this case all the options are specified. If, however, you wanted this module to work on *any* dimensionality of *4-vector uniform byte* data, then you would reset the **Dimensions** and the **Physical Space** to *Unspecified*.

This version of the Module Generator has the following limitations: it currently only supports up to three dimensions, three physical spaces, and a vector length of 10. These are user interface imposed limitations (not AVS limitations) and can be overcome by editing the source code for the file. When the file is read back in, these modifications, which the Module Generator does not understand, will be lost.

Geometry Editor

Like the **Field Editor**, the **Geometry Editor** is used to refine the selection of geometry ports *for output ports only*. (See Figure 2-12.) This information is only used to create the hints section in the compute routine. There are five possible geometries that can be produced: meshes, polyhedra, polytri (and polyline) strips, spheres, and labels. Please see the "AVS Data Types" chapter and the "Geometry Library" appendix of the *AVS Developer's Guide* for a more detailed look at these constructs.

There are several calls to AVS geom library routines that can produce any of these geometric primitives. The "hints" area generated in the compute routine



Figure 2-12: Geometry Editor

for the module will illustrate what the appropriate calls are, what the arguments for those calls are, and will make suggestions about the right way to build geom edit lists to produce these geometries. This can be a valuable tool for learning how to construct geom edit lists.

Parameter Editing

There are seven basic kinds of parameters and nineteen widgets that you can use in a module. The Module Generator gives you access to all of them. These categories are:

- **integer**—dials, sliders, typeins, toggles, tristates, and oneshots
- **real**—dials, sliders, and typeins
- **string**—text, typeins, file browsers, and text browsers
- **string_block**—text block browsers and text blocks
- **choice**—radio buttons and choice_browsers
- **others**—color_editors, and trackballs

See the "AVS Library Routines" appendix of the *Developer's Guide* for detailed descriptions of these widgets. The relevant calls are: **AVSadd_parameter()**, **AVSadd_float_parameter()**, **AVSadd_parameter_prop()**, and **AVSconnect_widget()**.

When specifying **integer** and **real** parameters, you need to specify the minimum, maximum, and default values. The Module Generator has default values of 0 for default minimum, 1 for default maximum, and 0 for default parameter values.

For **string**-related parameters, these initial conditions have different meanings:

Table 2-1. String Parameter Default Settings

Widget Type	Minval	Maxval	Default
typein	IGNORED	IGNORED	TEXT
text	IGNORED	IGNORED	TEXT

Table 2-1. String Parameter Default Settings

Widget Type	Minval	Maxval	Default
browser	IGNORED	File types	Filename
text_browser	Comment Char	IGNORED	TEXT
text_block_browser	IGNORED	IGNORED	TEXT
textblock	Comment Char	IGNORED	TEXT
radio_buttons	choice list	separator	default choice
choice_browser	choice list	separator	default choice

No initial values are needed for **oneshots**, **tristates**, **toggles**, **color editors** and **track balls**.

AVS Apply

Help Data Viewers Exit

Status (press to disable)

Top Level Stack

- read field
- extract scalar
- volume bounds
- extract vector
- display pmap
- orthogonal slicer
- field to mesh
- field to mesh
- orthogonal slicer
- generate colormap
- stream lines
- isosurface
- field legend

colormap
tree



<input checked="" type="radio"/> hue	composite
<input type="radio"/> saturation	edit
<input type="radio"/> brightness	read
<input type="radio"/> opacity	write

to value 0

hi value 255



AVS Network Editor

Help Close

AVS Module Library: Supported

Data Input	Filters	Mappers	Data Output
<input type="checkbox"/> animated float	<input type="checkbox"/> animate lines	<input type="checkbox"/> arbitrary slicer	<input type="checkbox"/> compare field
<input type="checkbox"/> animated integer	<input type="checkbox"/> antialias	<input type="checkbox"/> brick	<input type="checkbox"/> display image
<input type="checkbox"/> background	<input type="checkbox"/> clamp	<input type="checkbox"/> bubbleviz	<input type="checkbox"/> display pmap
<input type="checkbox"/> boolean	<input type="checkbox"/> colorizer	<input type="checkbox"/> contour to geom	<input type="checkbox"/> display tracker
<input type="checkbox"/> character string	<input type="checkbox"/> combine scalars	<input type="checkbox"/> field legend	<input type="checkbox"/> graph viewer
<input type="checkbox"/> color range	<input type="checkbox"/> composite	<input type="checkbox"/> field to mesh	<input type="checkbox"/> image viewer
<input type="checkbox"/> euler transformation	<input type="checkbox"/> compute gradient	<input type="checkbox"/> hedgehog	<input type="checkbox"/> output postscript
<input type="checkbox"/> file browser	<input type="checkbox"/> contract	<input type="checkbox"/> image to pmap	<input type="checkbox"/> print field

AVS Network Editor

- Network Tools
- Module Tools
- Layout Editor

Read Network

Write Network

Clear Network

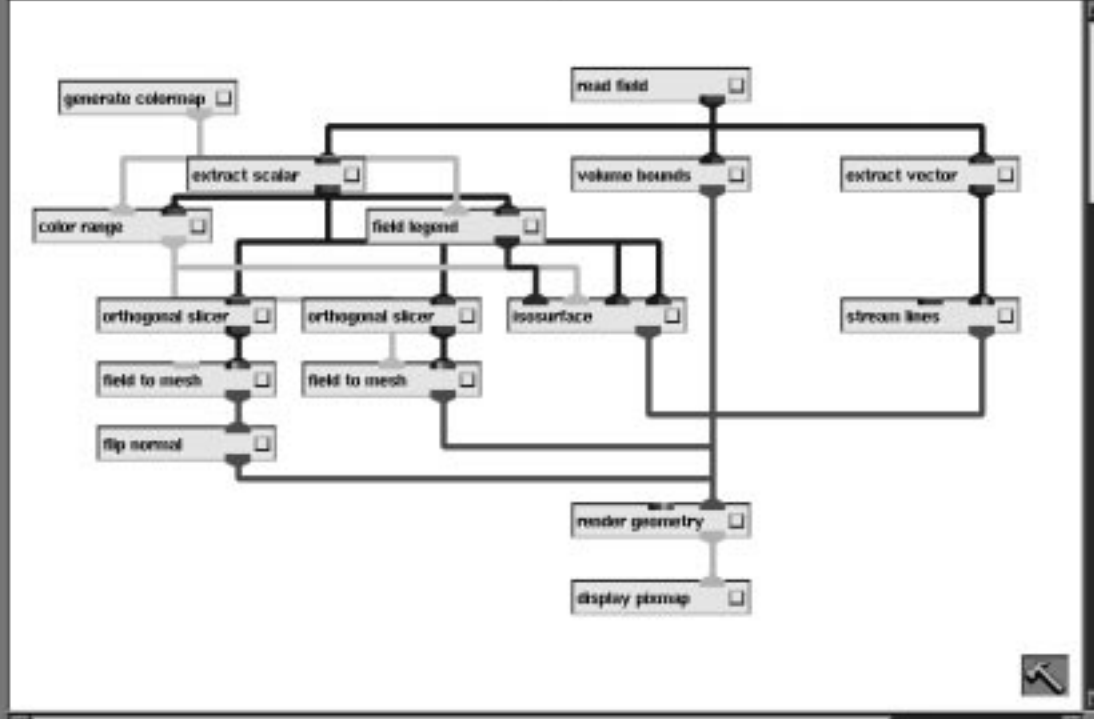
Print Network

Disable Flow Execution

Save Parameters

Restore Parameters

Network Diagram:



THE DATA VIEWER

Why a Data Viewer?

The **AVS Data Viewer** is a simplified user interface to the Application Visualization System's most commonly used scientific visualization techniques.

The centerpiece of any AVS session is a *visualization network*. (See the facing figure.) A network is essentially a graph that defines the flow of data through a series of programs, called *modules*, that process the data into a picture that can be viewed on the screen.

At the top of all networks is a *data input* module that reads data files off disk and into the network. Next may come a series of *filter* modules that pre-process the data (extract a single scalar element from a vector of data values, crop or thin out the data to a more manageable size, take a single 2D plane from a 3D volume, etc.). This is followed by one or more *mapper* modules that turn the data into a picture that is some analog representation of the numeric values: a familiar XY graph, a grid of spheres floating in 3D space whose color represents the data value, a 3D surface that is a contour through all equal values showing their distribution through a volume, and so on. Last comes some kind of *data output* module that actually produces the picture on the display screen.

You construct networks with the AVS Network Editor. The usual procedure is to scroll through the columns of modules on the module "palette," seeking the ones which you want. You drag modules one-by-one into the large empty workspace, then connect the modules together, in the right order, into a flow network. As you interact with your data, you may decide that you need different, additional data representations. To do this, you go back to the palette for more modules, bring them into the workspace, then change the connections between modules to hook in the new ones, producing a new data flow network.

There is a learning curve associated with building AVS networks similar to that of any language. The vocabulary of modules is rich; there are over 140 modules supplied with AVS. These can be extended indefinitely by user-written modules. Their names (**bubbleviz**, **thresholded slice**, **ucd streamlines**, **field legend**, **hedgehog**, etc.), though descriptive once you understand them, may be obscure at first.

Also like a language, networks have a grammar that guides but does not exhaustively define what visualization constructs you can build. The grammar is much richer than the linear *input -> filter -> mapper -> output* outlined above. It is more like the production rules of human languages.

There can be multiple inputs, or input can be coming from a simulation that runs as an asynchronous process. Filters may work upon data in parallel, or serially. Additional inputs may feed into the middle of a network. Data may be broken out into pieces that are worked on by different, parallel branches of a network, then composited back together again for final viewing, or viewed in separate windows in different representations. Intermediate data can be siphoned off into files, or for intermediate visual representations.

As a new AVS user, you are rather in the position of a visitor to a foreign country whose language you do not yet speak. You are standing on a street corner; you have a dictionary in one hand (the *AVS Module Reference* manual), and a grammar book (the "Network Editor" chapter of the *AVS User's Guide*) in the other. You know what you want to say, and you know that you probably hold all the pieces in your hands that you need to be able to say it. The problem is putting them together.

The Data Viewer takes an alternative "phrasebook" approach to visualization network construction. Rather than building networks "manually" by selecting the individual modules and connecting them together, the Data Viewer provides a pulldown menu interface. There is a menu for each of the main module categories (input, filter, mapper, output). From each menu, you select which input data type you need to use (AVS field or unstructured cell data), then any filter operation (crop, downsize, extract scalar), then any combination of data mapping techniques (slice planes, contour surfaces, vector streamlines). Each of these choices represents a predefined subnetwork. Behind the scenes, the Data Viewer automatically selects the corresponding modules and constructs the network. All that you see on the screen and need to deal with is the final output of the network—the data visualization in a display window.

This "pick one from column A," "some from column B," "some from column C," "some from column D" approach to visualization preserves a large measure of the original flexibility and dynamics of visualization possible with manual network construction, while eliminating much of the detail knowledge of network structure, data types, and mouse button mechanics required to perform it. As a new AVS user, you can concentrate on learning the actual visualization techniques.

Should you be curious, it is possible to view the networks underlying the visualizations as the Data Viewer builds and re-builds them, thereby learning basic network construction approaches. As you watch the networks, be aware that, though the Data Viewer's built in network grammar is flexible, it implements only a subset of the network structures supported by the full Network Editor. Examples of these more sophisticated visualization structures can be found in the other AVS learning tool, the **AVS Demo Suite**.

Starting the Data Viewer

You can start the Data Viewer in one of three ways:

1. From the AVS Applications menu. On the main AVS menu, select **AVS Applications**. This brings up a secondary menu. (See Figure 3-1.) Select **Data Viewer**. This is the usual way to start the Data Viewer.

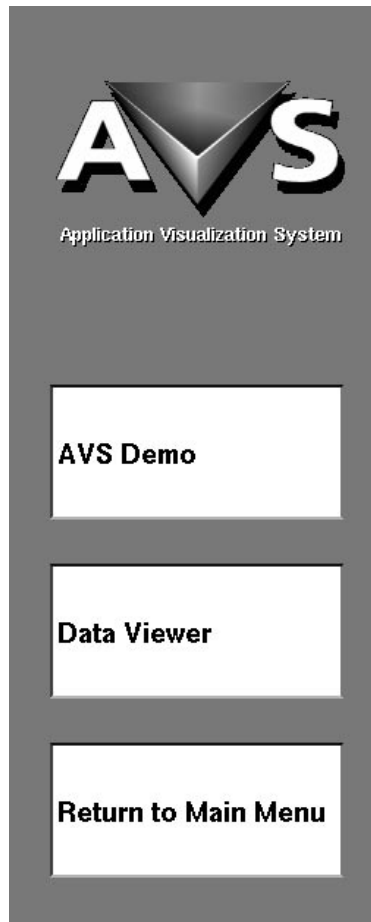


Figure 3-1 Starting the Data Viewer from the Applications Menu

2. From within the AVS Network Editor. The Data Output column in the module palette contains a **Data Viewer** module. Use the left mouse button to drag this module into the large blank workspace. You would probably use this method if you were intent upon watching the networks as the Data Viewer constructs them. (The Data Viewer provides its own switch that also lets you watch the networks as they are built and rebuilt.)
3. Directly from the system shell, as you start AVS. Type:

```
avs -network /usr/avs/networks/dv/data_viewer
```

Leaving the Data Viewer

Figure 3-2 shows the Data Viewer as it appears on the screen. It may take several seconds to initialize.

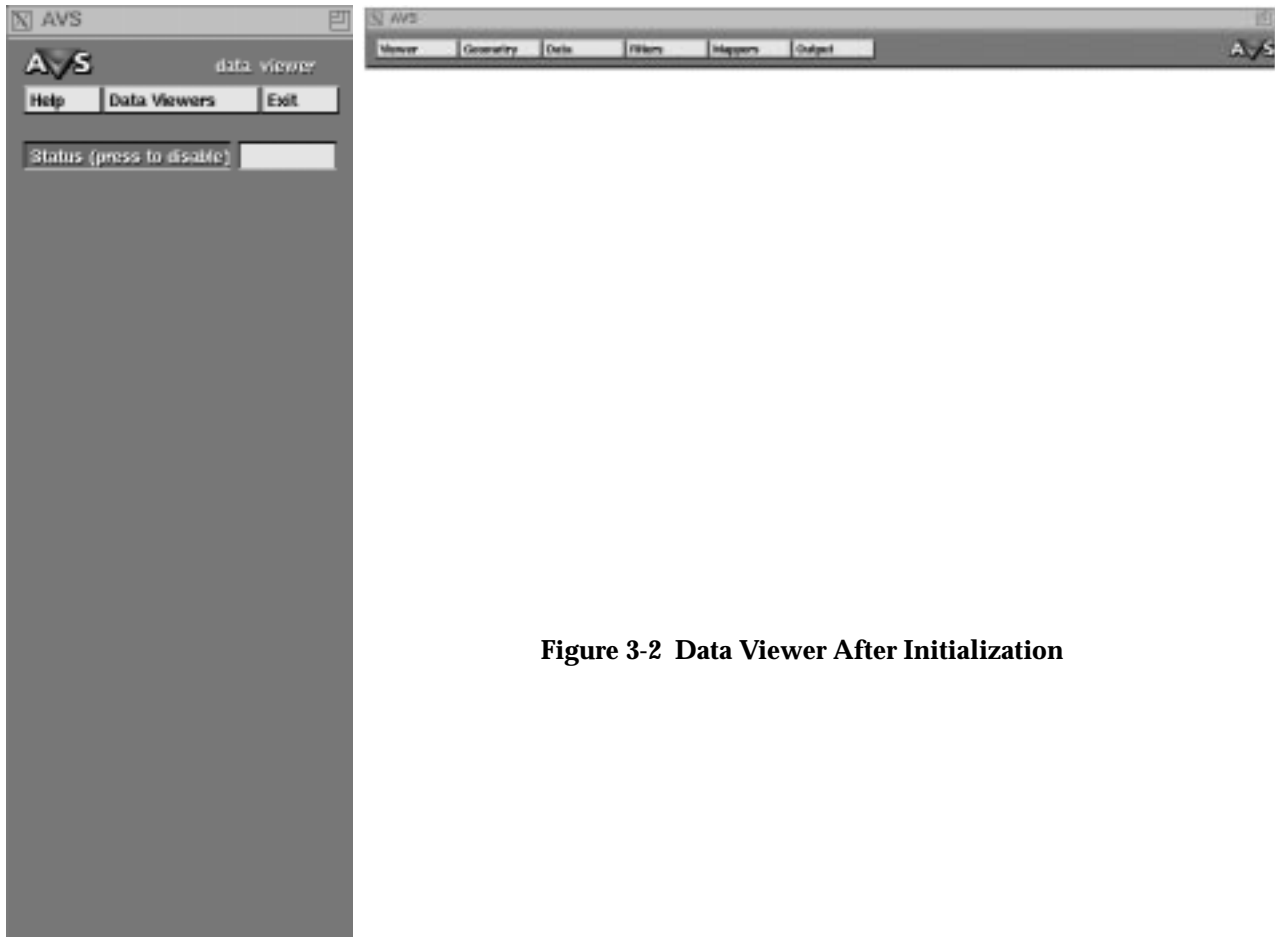


Figure 3-2 Data Viewer After Initialization

Leaving the Data Viewer

At the top of the Data Viewer's control panel is an **Exit** button. (See Figure 3-3.) When you press this button, a warning message appears as shown in Figure 3-4.



Figure 3-3 Exiting the Data Viewer

If there is work that you wish to save, click **Cancel** on the message panel. Press the **Viewer** button on the Menu Bar and select the **Save Viewer State** from the pulldown menu that appears. You are presented with a File Browser

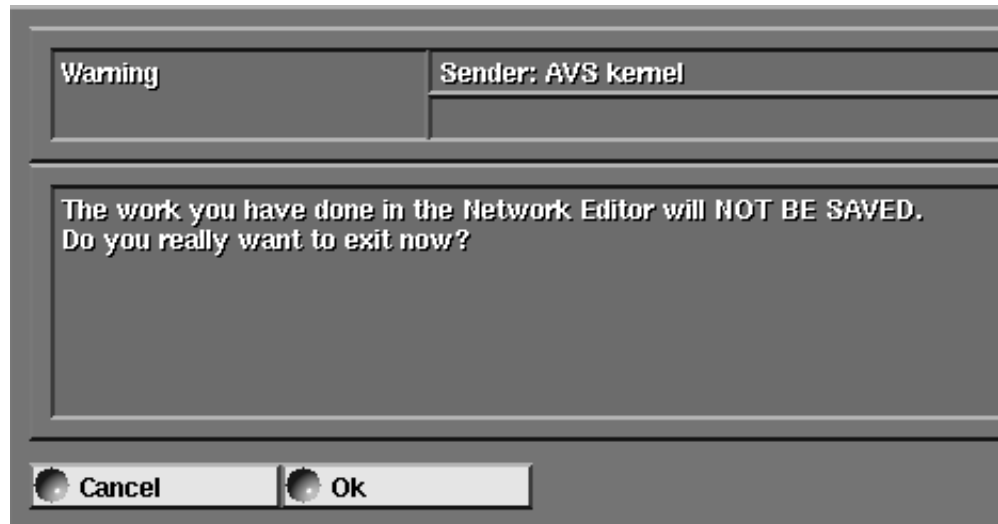


Figure 3-4 Work Will Not Be Saved Warning Panel

widget. Select either **New Dir** or **New File**. This raises a panel (Figure 3-6) into which you type a file specification and press **Enter**. The Data Viewer will save its current state. You can then exit the Data Viewer.



Figure 3-5 File Browser Widget

Otherwise, just click **OK** on the warning message panel. Exiting the Data Viewer returns you to the AVS Application menu.

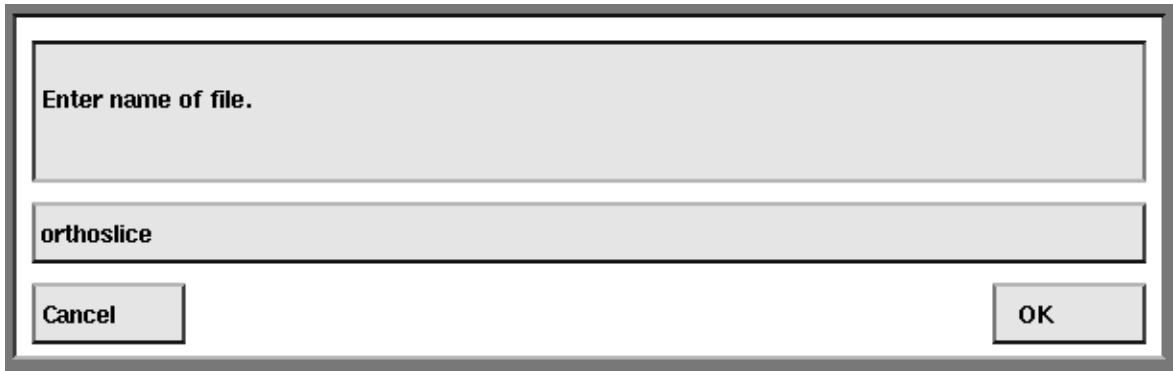


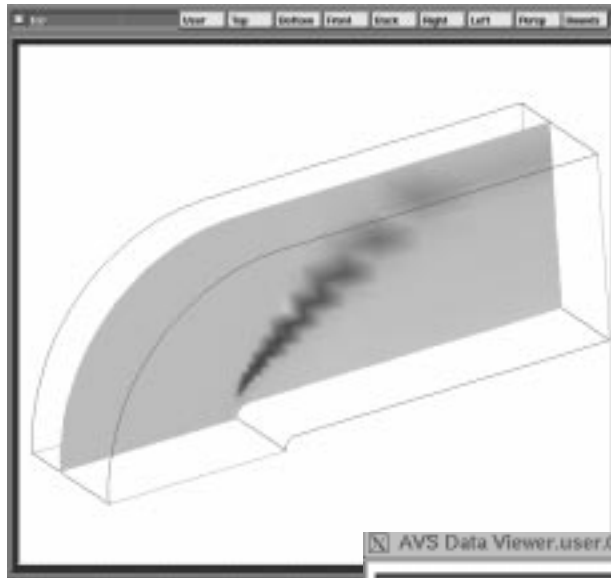
Figure 3-6 Save State File Typein Panel



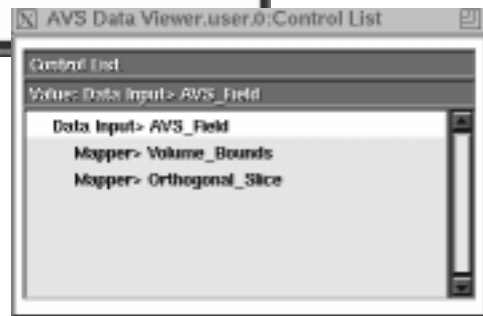
Control Panel



Menu Bar



Output Window



Control List

THE DATA VIEWER INTERFACE

Introduction

There are four main elements to the Data Viewer interface: the **Menu Bar**, the **Control List** window, the **Control Panel**, and the **Output** window. (See facing figure.)

Of these, the Menu Bar and the Control Panel are always visible, while the Control List window and the Output window appear only after you have selected an input data type from the Menu Bar.

The sections below describe each interface element.

The Menu Bar

Across the top of the screen is the Data Viewer's main Menu Bar. There are six buttons, each of which calls up a pulldown menu.



Figure 4-1 The Menu Bar

You interact with the Menu Bar in the way that you would expect: it behaves like most button-across-the-top, pull-down-menu interfaces found in today's software products. Pressing any mouse button calls up the pull down menu; hold the mouse button down as you move the mouse cursor down the list; release the mouse button over your selection; or roll the mouse cursor off the list and release the mouse button to choose nothing. "Not applicable" items will be shaded out automatically according to context. For example, some techniques work only with field data, and will be shaded out if you are working with UCD data.

You can select among the buttons and their menus in random order throughout your session. Nothing—except choosing an input data type—has to come first. The Data Viewer's built-in network grammar takes care of

adding, inserting, deleting, and duplicating techniques in their proper place in the underlying network.

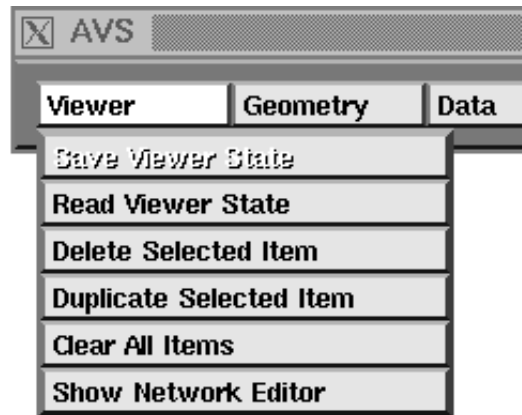


Figure 4-2 Viewer Pull Down Menu

Viewer

The Viewer menu contains general utility functions: **Read/Save Viewer State**, **Delete/Duplicate/Clear** visualization techniques from the Control List, and a **Show Network Editor** button that opens up the large AVS Network Editor panel so that you can watch the networks as they are being built.

If you watch the networks, it may be necessary to use the left mouse button to rearrange the module icons so that you can clearly see the connections.

Geometry

The Geometry menu contains buttons that affect the appearance of objects in the Output window. Its choices are really a "most-critical, most useful" subset of the many functions available in the full AVS Geometry Viewer. For example, you can temporarily **Hide/Show** objects in the output window, or change their rendering from a **Surface** representation to a wireframe **Lines** representation. Some techniques may shade out some of these choices.

The two functions that you will use most often are **Reset** and **Normalize**. **Reset** returns the an object to its original position in the window. It is a "start over" button. **Normalize** changes the size and position of an object so that it fills the Output window. **Center** does not center the object in the Output window. Rather, it sets the **Center** of rotation of objects to their physical centers.

There may be multiple geometry objects in the Output window. The Control List selection establishes which object will be affected by this Geometry menu. If a Mapper> technique is selected, then only the object representing that technique will be affected. If the top-level Data Input>

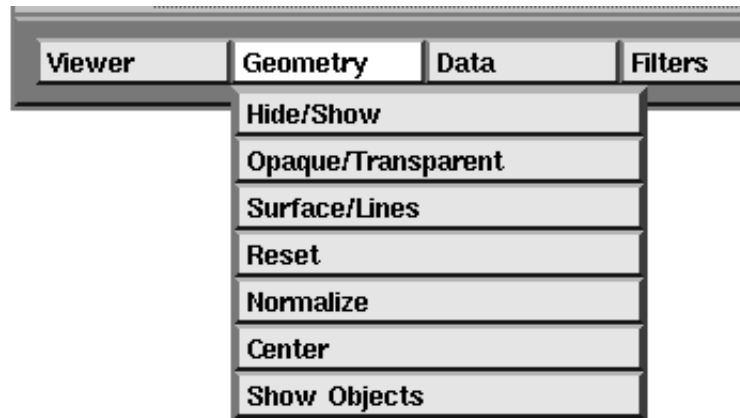


Figure 4-3 Geometry Pull Down Menu

technique is selected, then all objects in the Output window will be affected.

Data
Filters
Mappers
Output

The **Data**, **Filters**, **Mappers**, and **Output** buttons select the actual visualization techniques.

Data

As you begin a Data Viewer session, you will always use the Data menu to select the type of input data you will work on, either an AVS field or an AVS unstructured cell data (UCD) format input file. Once you have done this, the Control List and the Output window will appear. Unlike the complete AVS Network Editor, there can be only one input dataset per network.

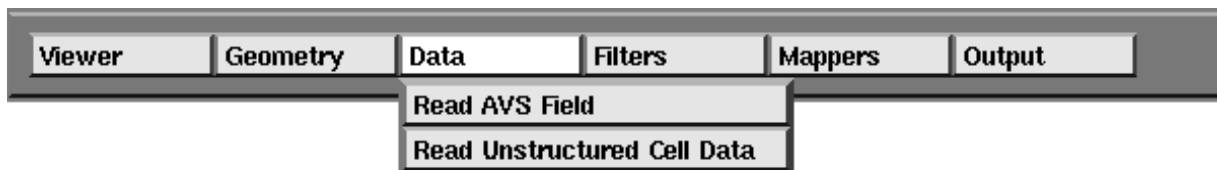


Figure 4-4 Data Pull Down Menu

Thereafter, your navigation through the Filters, Mappers, and Output menus is up to you.

Mappers

Most of your Menu Bar activity will involve selecting one or more visualization techniques from the Mappers column. This is where the core visualization techniques (**Orthogonal Slice**, **Isosurface**,

Colored Streamlines, etc.) that produce the actual pictures in the Output window reside.

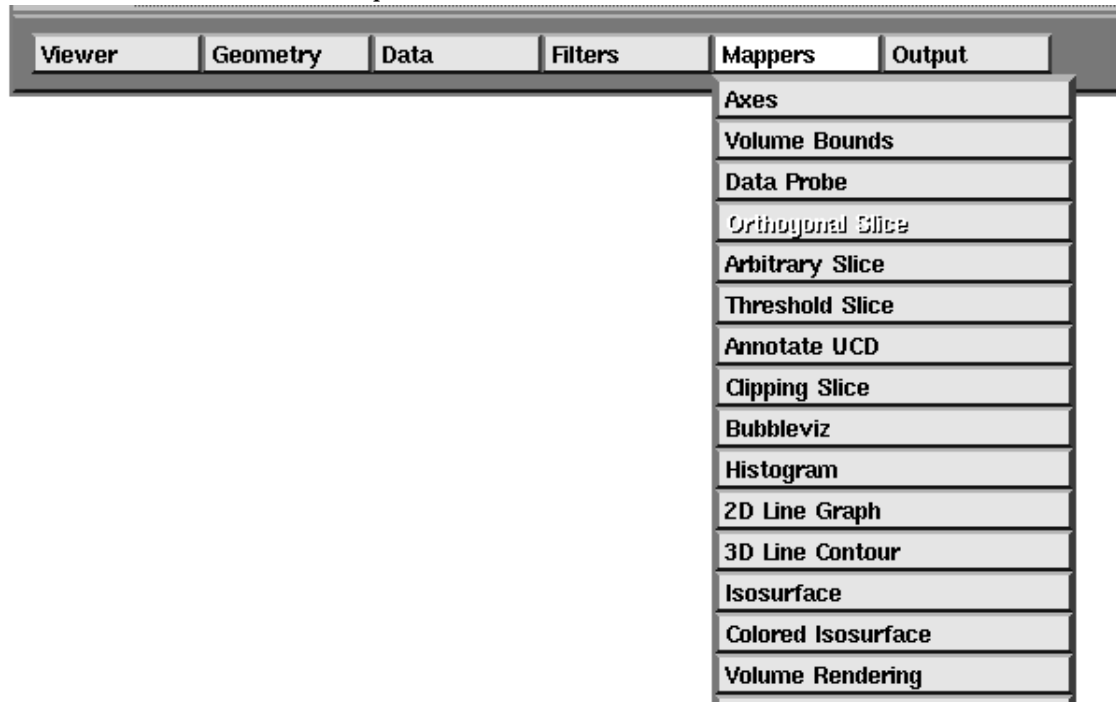


Figure 4-5 Mappers Pull Down Menu

Output

You may also, with less frequency, select from the Output menu. The network fragments represented by these buttons perform functions that will be of interest to you both while you are doing your visualization (for example, **Field Statistics** displays field or UCD minimum, maximum, and mean data values which you may need to define upper and lower bounds on dial control widgets); and before you leave the Data Viewer. (**Postscript**, for example, creates a gray-scale or color PostScript file of the image in the Output window that can be sent to a printer.)

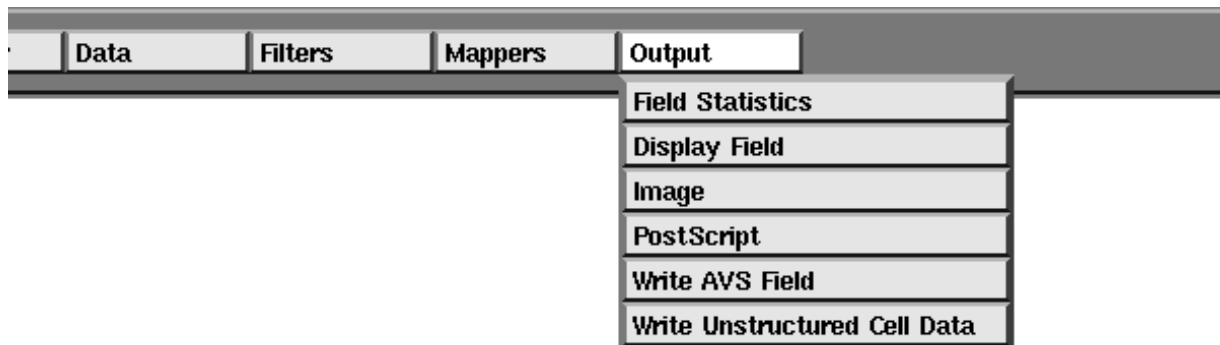


Figure 4-6 Output Pull Down Menu

Filters

The Filters menu contains buttons that call up network fragments that will pre-process the raw input data before it reaches the mapping technique(s). For example, **Crop** will pare down the size of the input data in any or all dimensions, like "cropping" a picture. **Downsize** will "thin out" the data, selecting perhaps only every other, or every fourth value in the input field to pass along to the mapper.

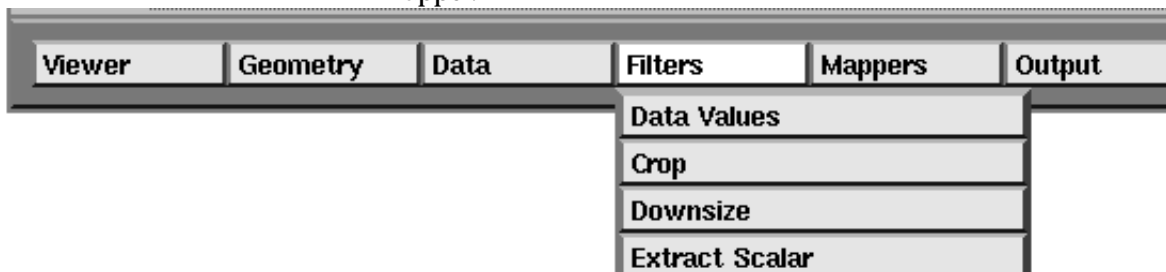


Figure 4-7 Filters Pull Down Menu

In addition to their function as visualization utilities, both **Crop** and **Downsize** are also tools to keep the sheer size of the data that the Data Viewer's network must process—and the 3D drawing facilities must render—within the realistic bounds of your workstation's capacity and rendering speed.

The Control List

As you select techniques from the Data, Filters, Mappers, and Output menus, the techniques are added to the Data Viewer's Control List window. (See Figure 4-8.)

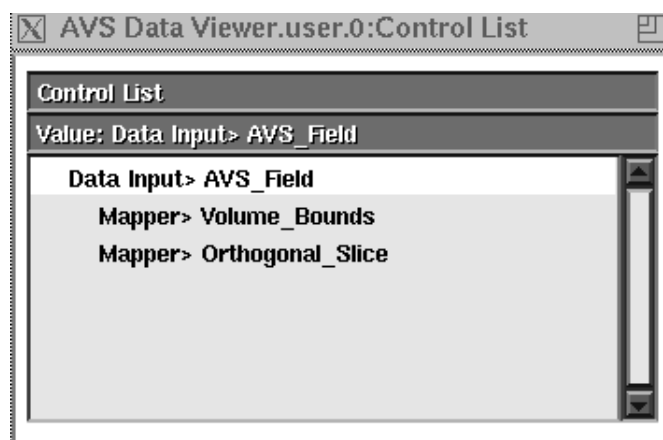


Figure 4-8 Simple Control List

The Control List has three related functions:

- The Control List represents the content and structure of the visualization that you are performing.
- The Control List selects which objects will be affected by choices made in the Menu bar. For example, if you wish to delete an item from the Control List, first select it. Then, use the Viewers menu's **Delete Selected Item** function. If you wish to reset an object back to its original position when it entered a geometry Output window, select its corresponding Mapper> in the Control List. Then, use the Geometry menu's **Reset** button to reset the object.
- The Control List selects which technique's control widgets are displayed in the Control Panel at the left of the screen.

The Control List is organized as an indented hierarchy. At the root of every visualization is a Data Input> technique. The second level of the hierarchy can be occupied by zero, one, or multiple Filter> techniques. These Filtering techniques will act *serially* upon the various Mapper techniques, e.g., first you **Crop** the data, then you **Downsize** it. The leafs of the hierarchy are always one or more Mapper> or Output> techniques.

Each visualization requires, at minimum, a Data Input> technique and a Mapper> technique. Filter> and Output> techniques are optional.

There can be only one Data Input> technique at a time. Although AVS's Network Editor easily allows multiple input datasets to feed into the same network and be composited together in one output window (for example, a dataset containing people's ages mapped onto a geographic coordinate grid, and a separate dataset containing people's income in the same geographic area), the Data Viewer's more restricted network grammar allows just one input dataset.

These simple rules allow a wide variety of visualization network constructions.

Figure 4-8 is the simplest of constructions. This visualization network reads in an AVS field, then uses one mapper technique, Orthogonal Slice, to display the contents of the field as a "slice" through the volume that is orthogonal to the data's I, J, or K axis. All visualization networks contain the utility mapper Volume Bounds that displays lines around the volume's extents (borders) in space. In UCD networks, Volume Bounds also controls the appearance of the UCD cells. For example, you can display only cell edges, only exterior cell faces or all faces, and you can shrink the cells to make their structure more obvious. Volume Bounds appears automatically when you select a Data technique.

Figure 4-9 is a slightly more complex construction:

This visualization network is identical to the previous, except that two mappers have been added: Histogram, which creates a graph Output window that contains an XY graph of the distribution of data values in the input data;

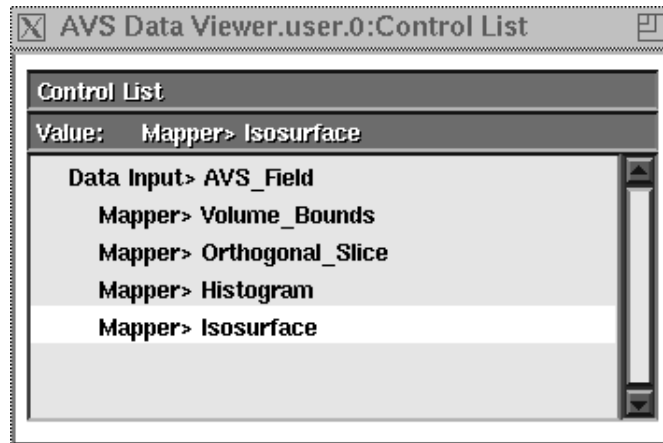


Figure 4-9 Control List with Multiple Mapping Techniques

and Isosurface, which creates a 3D surface through all data values in the volume that are equal. The Isosurface and Orthogonal Slice will appear composited together in a geometry Output window.

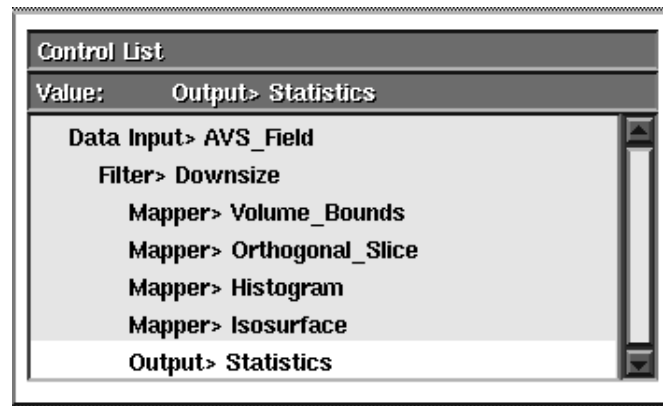


Figure 4-10 Control List with Filter and Output Techniques Added

The visualization network in Figure 4-10 is identical to the previous, except that a Filter> Downsize has been inserted before the mappers to thin out the data. This network also includes an Output> technique that will display some useful statistics about the data content of the field.

When adding items to the Control List with the Menu Bar, keep these rules in mind:

- You must select a Data input technique first.
- The Mapper> technique will appear *after* the current Control List selection. However, their position in the Control List does not really have any affect on their behavior because Mapper techniques are always leaves in the hierarchy and they are peers of each other. Multiple Mapper techniques create parallel branches in the underlying network, each producing a different picture from the same data. The pictures are composited

together in the geometry Output window, or appear in additional graph and/or image Output windows.

- Output techniques always appear at the end of the Control List as leaves in the hierarchy.
- Filter techniques are always inserted *before* a Mapper> or Output> technique selected in the Control List. If the selection is a Filter> or the Data Input> technique, the Filter is inserted immediately *after*.

Note: Filter techniques actually present a bit of a problem. It is possible to make an indented hierarchical representation in the Control List window using Filters that does not accurately reflect the structure of the underlying network. Some rules of thumb to avoid this are:

- Do not have Volume Bounds selected when you insert a Filter. This applies the Filter only to the Volume Bounds technique, not to the actual data.
- If you want to filter all Mappers (the most typical case), first select Data Input> in the Control List, then choose from the Filter menu.
- If you want to filter just an individual Mapper, put that Mapper at the end of the Control List, select it in the Control List, and then choose from the Filter menu:

```
Data Input> AVS_Field
  Mapper> Volume Bounds
  Mapper> Isosurface
  Filter> Downsize
    Mapper> Bubbleviz
```

This Downsizes just Bubbleviz. In this case, there can be only one Filter in the Control List. The second Filter added will cause all of the Filters to act serially upon all Mappers. For example, subsequently attempting to use a different filter on Isosurface:

```
Data Input> AVS_Field
  Mapper> Volume Bounds
  Filter> Crop
    Mapper> Isosurface
  Filter> Downsize
    Mapper> Bubbleviz
```

actually causes both Bubbleviz and Isosurface to be both Cropped and Downsized, even though the indentation does not suggest that this is true.

You can use the Viewer menu's **Show Network Editor** selection to see the structure of any network underlying a Control List.

The Control Panel

Each visualization technique in the Control List has a set of widgets that allow you to control its functions. (See Figure 4-11.)

Widget controls are organized onto pages. The pages are in a metaphorical "stack" on the **Control Panel** at the left of the screen.

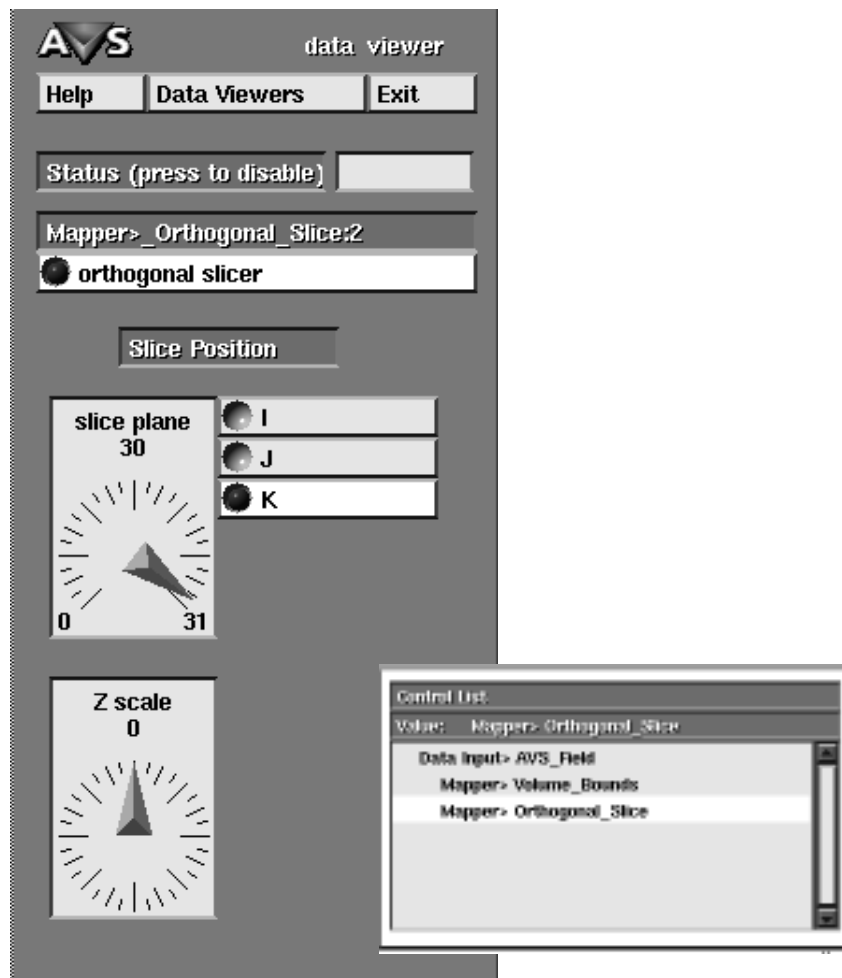


Figure 4-11 Orthogonal Slice Control Panel

By clicking on different techniques in the Control List, you bring up the different pages of widget controls on the Control Panel.

For example, the Orthogonal Slice technique has three buttons that control whether the slice plane is oriented in the I, J, or K (X, Y, or Z) plane. There is a dial that controls the Distance of the slice through the volume (plane 0, plane 2, plane 3,...plane n). The Z dial can be used to alter the appearance of the slice plane: higher values will be mapped as a distortion in the plane in the third dimension.

Some techniques have multiple pages of controls. For example, both of the Data techniques, (**Read AVS Field** and **Read Unstructured Cell Data**) have three pages. You switch between these multiple sub-pages by clicking on the choice buttons at the top of the Control Panel. (Figure 4-12.)

One page contains a file browser that you use to pick the input dataset (**Read Field** in Figure 4-12). One page contains the colormap editor widget that controls the mapping from data values to colors (**Colormap** in Figure 4-12).

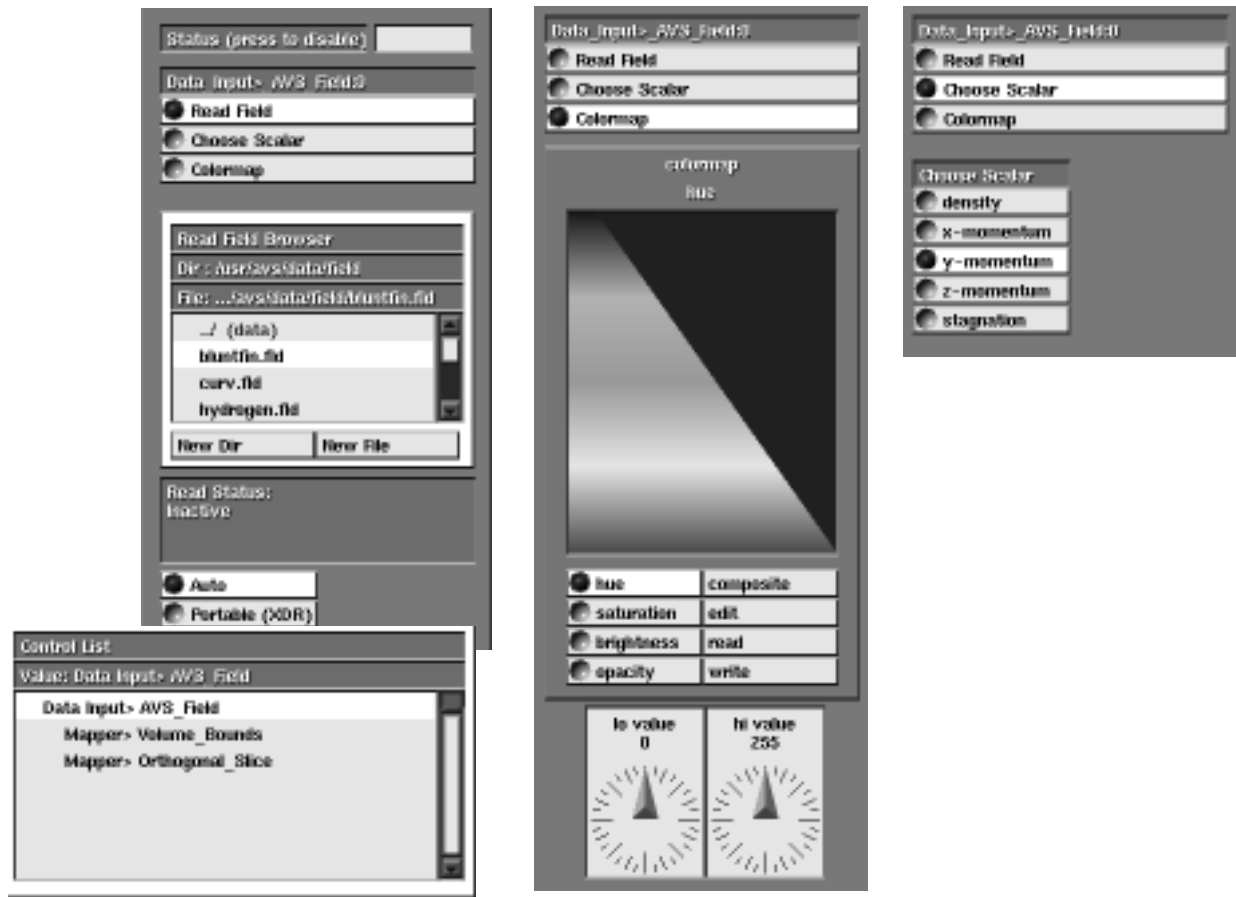


Figure 4-12 Read AVS Field Technique—Selecting Among Control Panel Pages

Fields and UCD data can contain multiple values (a vector) at each grid point. Though there are techniques that will draw objects that represent magnitude from three components (Hedgehog, Streamlines, Colored Streamlines, Particle Advector), most of the time you select just one of the vector elements and map its data values. Thus, the third page on the read technique (**Choose Scalar**) selects which of the vector values to send through the network to the Mapper.

Field Legend

When you first read in a dataset, the colorbar Field Legend will appear at the bottom of the screen, produced by the **field legend** module in the underlying network. Field Legend is a way of selecting data values by their representative color rather than by numeric values. However, the Field Legend control widget only works with these field Mapping techniques: 3D Line Contour, Isosurface, and Colored Isosurface; and these UCD Mapping techniques: Threshold Slice, 3D Line Contour, Isosurface, and Colored Isosurface. If you are not using one of these techniques, you should iconify and ignore the Field Legend widget.

Control Widgets

Use of the control widgets that appear in the Control Panel is fairly intuitive. There are browsers to select input and output files, radio buttons to select among options such as which vector element to send through the network, switch buttons to turn options on and off, and typeins to specify exact numeric values and file names (**Ctrl-u** clears a typein; **Backspace** erases single characters).

The Colormap Editor found under the Data Input> technique is described in detail in the "Using the Colormap Control" section of the *AVS User's Guide "Network Editor"* chapter, and again on the **generate colormap** page of the *AVS Module Reference* manual.

Dial widgets are largely intuitive, but do have some subtleties:

- Dragging the dial needle with the mouse button causes the network to be updated continuously. Click on the perimeter of the dial to jump the needle to a new value.
- To enter a specific value, click on the small circle at the base of the dial needle. The resulting Dial Editor has a typein for specific values.

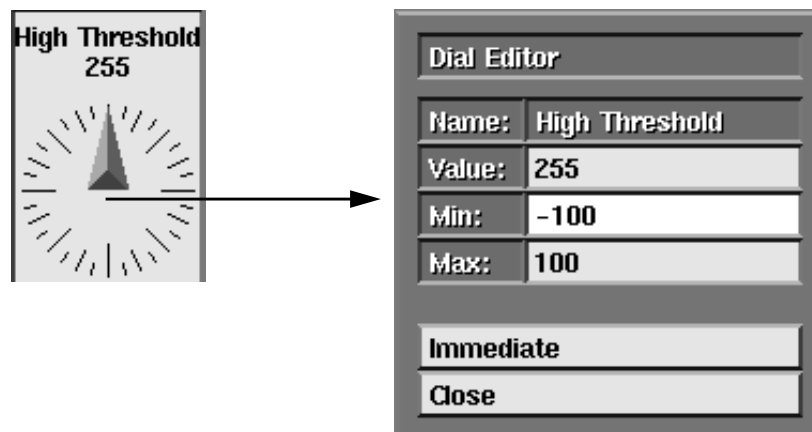


Figure 4-13 The Dial Editor

- The resolution of an unbounded dial is "once around is 200". Datasets with narrow data ranges (e.g., -0.4 to +2.5) may make the dials too sensitive to use effectively. Use the Dial Editor's **Max** and **Min** typeins to reset the dial's resolution. Both the Histogram Mapper technique, and the Field Statistics Output technique will show a dataset's data range.

Dial widgets are described in detail in the "Using Dial Controls" section of the *User's Guide "Network Editor"* chapter.

The Output Window

There are three kinds of Output windows:

- "Geometry" output windows that display the output from mappers as 3D objects. Most mappers produce geometry Output windows.
- "Graph" output windows that display the output from mappers as an XY 2D graph. Two field techniques, **Histogram** and **2D Line Graph** produce graph Output windows.
- "Image" output windows display the visualization as a 2D image. The **Volume Rendering** Mapper produces an image Output window that is really the **display tracker** module. The **Image** Output technique produces an image window that is the **image viewer** module.

Depending upon which techniques you have selected, you may have one, two, three or four output windows—one of each type—on the screen at once. In most cases, however, there will be just one geometry output window.

Geometry Output Windows

A geometry Output window is depicted in Figure 4-14. The window is *not* a "dead, for display only" output window. You can interactively manipulate the objects in geometry output windows: rotate the objects to get a better point of view; rescale them larger or smaller; or shift them up, down, left, and right—all using direct manipulation with the mouse buttons.

Across the top of the geometry output window are a series of buttons that also allow you to quickly change your point of view on the object. For example, you can look at it from its **Top**, **Front**, and **Right** views and their inverses. You can cause the object to be presented as a **Perspective** projection (instead of parallel) giving you a better sense of depth perception. **Bounds** will reduce the amount of drawing that your system has to perform as you move objects, speeding the interaction.

A geometry output window is actually being produced by the **geometry viewer** module in the underlying network. You can exact much finer, detailed control over the contents of the output window (lights, cameras, object properties, etc.) by using the AVS Geometry Viewer subsystem. This is accessible from the Data Viewer using the **Data Viewers** pulldown menu at the top of the Control Panel. In general, the simplified controls provided in the Data Viewer are sufficient. There are occasions when you may need to call up the full Geometry Viewer.

Mechanics

Three mechanisms control objects in a geometry Output window:

- Mouse buttons
- The buttons along the top of the geometry Output window
- The Data Viewer Menu Bar's Geometry menu.

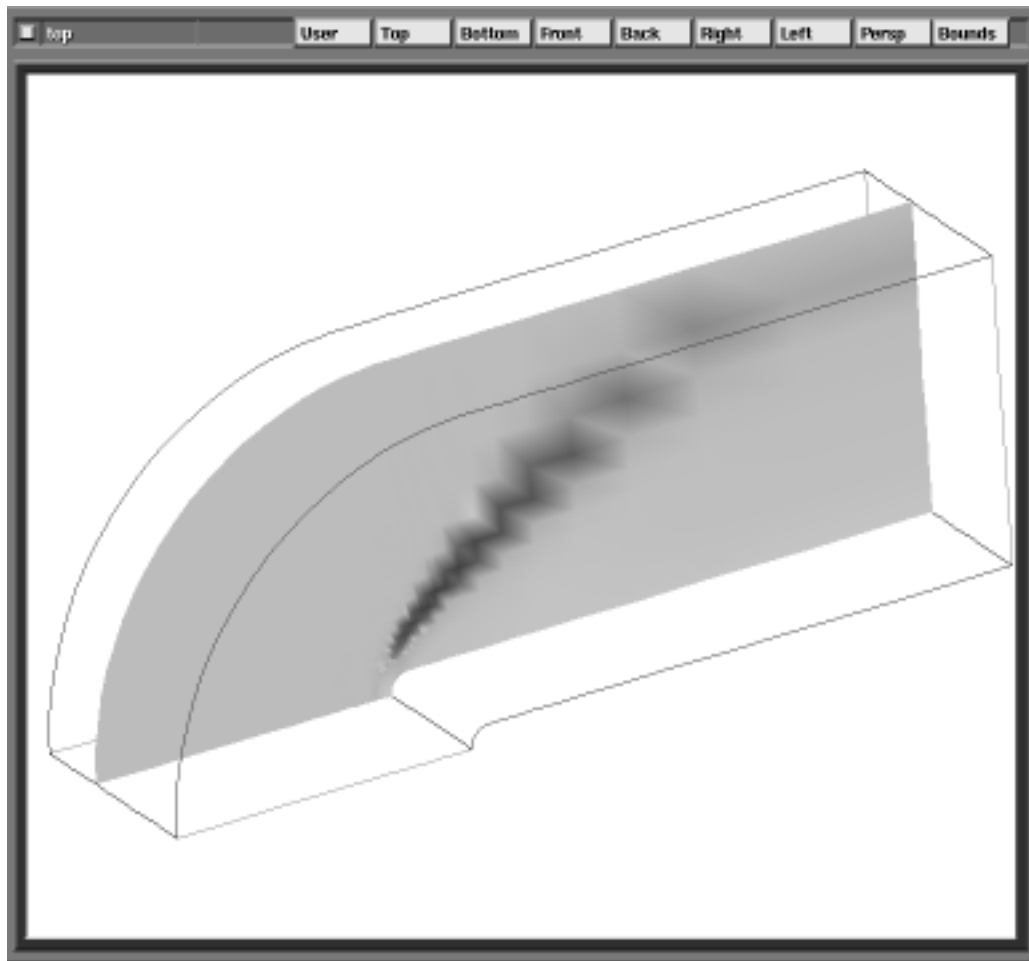


Figure 4-14 Geometry Output Window

When using mouse buttons and the buttons on the geometry Output window, you choose which object(s) will be affected with the mouse buttons.

When using the Geometry pulldown menu, you choose which object(s) will be affected with the Control List. Selecting Data Input> selects all objects.

You can move (translate), rotate, and scale objects in the output window. These operations are collectively called *transformations*.

Objects in the geometry output window are organized into a hierarchy. Top is the "top" of the hierarchy. All other objects are children of the Top object and peers of each other.

Mouse Buttons

The upper left corner of the geometry output window shows the *current object*. This is the object that will be transformed by the mouse buttons in Table 4-1.

To set the current object, point at it with the mouse cursor and press the left mouse button. To make Top the current object, point at the window background and press the left mouse button.

Many objects cannot be transformed independent of the Top object with the mouse buttons (Volume Bounds, Orthogonal Slices). These objects are either immobile for logical reasons (Volume Bounds), or are moved using dials on their respective control panels.

Table 4-1. Mouse Button Transformations^a

Action ^b	Mouse Button
pick current object	left
rotate	middle
scale	shift-middle
translate XY (move)	right
translate Z ^c	shift-right

a. Transformations occur in XYZ of the view volume, not object space.

b. Use **Bounds** for faster rendering.

c. Use **Persp** to see object change size.

Geometry Output Window Buttons



The geometry Output window buttons perform these functions:

User

Returns the objects in the window to their orientation prior to pressing **Top/Bottom**, **Front/Back**, **Left/Right**. Thus, you may use the mouse buttons to transform the objects in the window to any orientation. This particular view is always retrievable by pressing **User**.

Top/Bottom

Top views the objects from the positive Z axis. **Bottom** views the objects from the negative Z axis.

Front/Back

Front views the objects from the negative Y axis. **Back** views the objects from the positive Y axis.

Left/Right

Left views the objects from the negative X axis. **Right** views the objects from the positive X axis.

Persp

Turns on a perspective view of the objects. This makes it easier to detect depth and location.

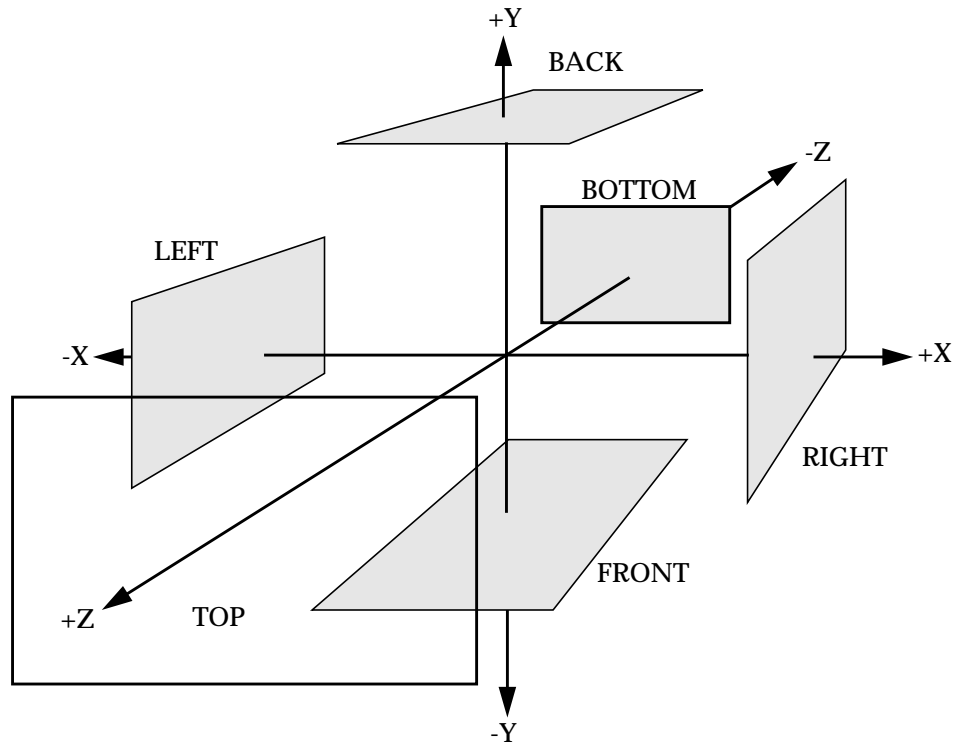


Figure 4-15 Views Produced by Geometry Output Window Buttons

Bounds

Reduces rendering overhead. Instead of moving objects directly, you move a bounding box that surrounds the objects. When the bounding box is correctly positioned, release the mouse button. The objects are re-drawn at their new location. The bounding box is also an orientation aid.

Geometry Menu

Different techniques may shade out one or more of these options.

Hide/Show

Temporarily "undisplay" the current object.

Opaque/Transparent

Makes the current object semi-transparent.

Note: Not all workstation hardware renderers support transparency. If this button has no effect, you will need to switch to the software renderer. At the top of the Control Panel at the left of the screen, press and hold down **Data Viewers**. When the pop-up appears, select **Geometry Viewer**. The full AVS Geometry Viewer's control panel will appear. Press **Cameras** on the menu below the miniature object window. Select **Software Renderer**. Press **Close** at the top of the panel to remove the Geome-

try Viewer. One can always access the main AVS subsystems from the Data Viewer in this way.

You may also need to switch to the software renderer if your hardware does not render spheres. The Bubbleviz Mapper technique produces spheres. If no spheres or dots appear when you use this technique, even after adjusting the various radius dials, switch to the software renderer.

Surface/Lines

Render the current object as solid, or as a wireframe of lines.

Reset

Return the current object to its original size and position.

Normalize

Resize the current object until it just fills the geometry Output window.

Center

Set the current object's center of rotation to the center of its extents in space. This will remedy the situation where objects appear to be rotating off-center.

Show Objects

Brings up a Current Object Browser listing all objects in the geometry Output window. Clicking on the names in the list is an alternate way to select the current object.

Graph Output Windows

A graph Output window is depicted in Figure 4-16. This window is not interactive, being for display only. The graph output window is actually being produced by a **graph viewer** module in the underlying network. You can control the display in this window (for example, the number of tic marks, whether new plots are added to the window or replace the existing plot) by using the AVS Graph Viewer subsystem, accessible from the Data Viewer using the **Data Viewers** pulldown menu at the top of the Control Panel.

Image Output Windows

The image Output window produced by the **display tracker** module when using the Volume Rendering technique is interactive. Table 4-1. shows the mouse buttons functions

The image Output window produced by the Image Output technique is interactive. The image Output window is actually being produced by an **image viewer** module in the underlying network. You can use the **image viewer** to perform image processing upon the results of a visualization. For example, you can increase its contrast so that it reproduces better in print. To access

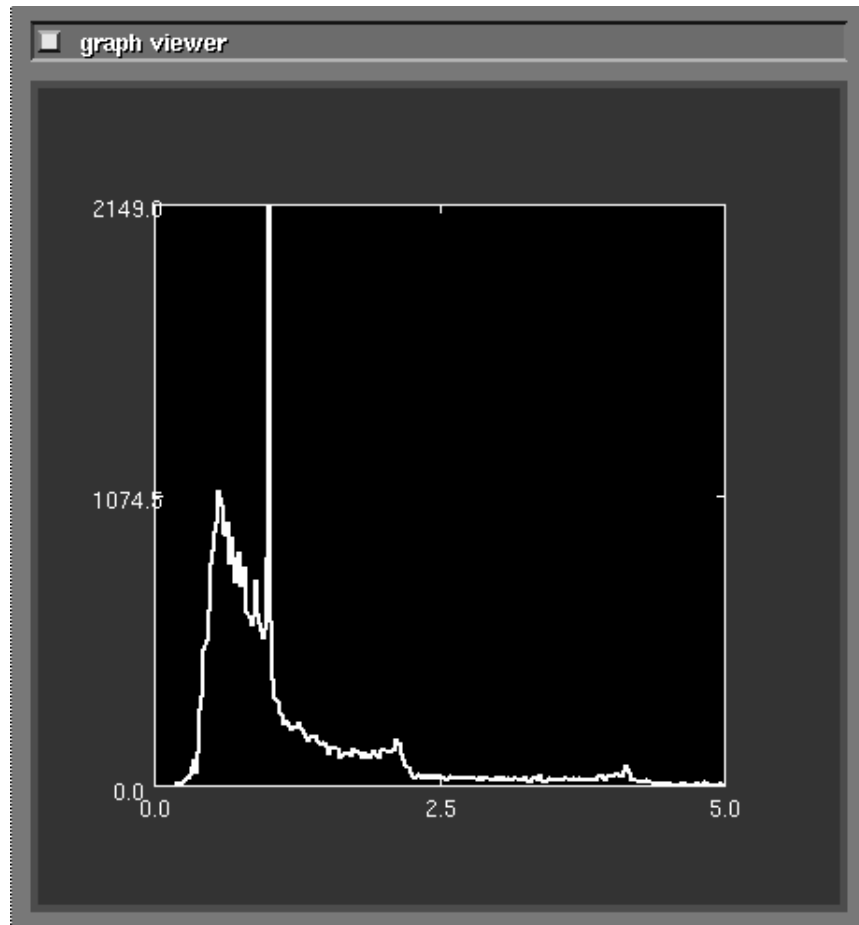


Figure 4-16 Graph Output Window

the full Image Viewer, use the **Data Viewers** menu on the Control Panel. Table 4-3 shows mouse button manipulations possible in this window..

Table 4-2. Display Tracker Mouse Button Transformations

Action	Mouse Button
reset object	left
rotate	middle
scale	shift-middle
translate XY (move)	right

Table 4-3. Image Viewer Mouse Button Transformations

Action	Mouse Button
pick current image	left
scale	shift-middle
translate XY (move)	right

The Techniques

To determine the purpose and use of the control widgets that appear on the left Control Panel with each technique, use the Viewers menu's **Show Network Editor** option to display the underlying network in the Network Editor. As techniques are added, their implementing modules will be included into the network.

Explanations of these modules' control widgets are found in two places:

- The *Module Reference* manual that accompanies each AVS release.
- In online help files. To access these help files, click on the module icon's small "dimple." A Module Editor Control panel will appear. Click on this panel's **Show Module Documentation** button. This brings up a text browser containing the "man page" for the module with a detailed explanation of the purpose and use of each control widget. Realize that the Data Viewer may be collecting the widgets of several modules together on one page.