AVS 5 UPDATE

Release 5
February, 1993

Advanced Visual Systems Inc.

Part Number: 320-0019-03 Rev A

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, 1993
Advanced Visual Systems Inc.
All Rights Reserved

AVS is a trademark of Advanced Visual Systems Inc.

STARDENT is a registered trademark of Advanced Visual Systems 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, ULTRIX, OpenVMS, VMS, DECwindows, DMS, VAX, ULTRIX Worksystem Software, and the DIGITAL logo are trademarks or 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, CRAY X-MP EA, and CRAY Y-MP are registered trademarks 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, Solaris, XDR, XGL, and SunVision are trademarks or registered 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, IRIS Crimson, and Personal IRIS are trademarks of Silicon Graphics, Inc.
DG/UX and AViiON are trademarks and registered trademarks of Data General Corporation.
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.
FLEXlm 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

CHAPTER 1

INTRODUCTION

AVS 5 Documentation

This *AVS 5 Update* manual documents AVS 5's new features. It describes improvements to the AVS interface, data types, Image Viewer, Geometry Viewer, and new facilities for module and application developers. It also provides summary tables that list new and upgraded modules.

The *AVS Module Reference* manual has been updated for AVS 5 to include the module man pages for the new and significantly-upgraded modules.

Almost all of the new features in AVS 5 represent new functionality that augments the information in the *AVS User's Guide*, *Developer's Guide*, *Tutorial Guide*, and *Applications Guide*, without contradicting or overriding the existing text. Thus, the remainder of the AVS documentation set is the same as that provided for AVS 4.

AVS 5 New Features

The new AVS 5 features documented in this *AVS 5 Update* manual and the updated *AVS 5 Module Reference* manual encompass:

- Over 65 new modules, including:
    - A set of 45 image processing modules that include support for Fourier transformations, morphological operations, warping, contouring, and arithmetic functions.
    - Five new volume rendering modules, including the **cube** and **edit substances** modules that will render based on user-defined substance classifications.
    - Eleven new unstructured cell data (UCD) modules, including support for vector curl, division, and gradient operations, and a **scatter to ucd** module that converts a scatter field to a UCD structure so that it can be visualized using UCD modules and techniques.
    - Five new modules for data presentation, including the **generate axes** module that gives the user full annotation control over geometry axes, the **color legend** module that places an annotated colormap in a Geometry Viewer window, and the **3D bar chart** module that creates three dimensional bar graphs.

- Interface and API support for interactive mouse-driven image drawing, picking, and mensuration operations in the Image Viewer.
- 8-bit and scalar images in the Image Viewer.
- Full support for a short (usually 16-bit) field datatype.
- Geometry Viewer support for:
    - Sending scene windows to a different display (for multi-headed systems).
    - Turning off image output to improve performance on hardware renderers.
    - Non-transformable objects.
    - New texture-mapping *libgeom.a* library calls.
- A simplified interface to creating macro modules in the Network Editor.
- Support for ANSI-C and C++ modules.
- Unmapped module control panels to suppress unwanted displays.
- The ability to overwrite the input data to modules in shared memory instead of allocating a new output data shared memory segment.
- A new, reorganized list of default module libraries.
- AVS can be installed anywhere in a directory hierarchy, without needing links from */usr*.

Demos of New Features

The **AVS Demo** application (located on the secondary AVS main menu accessed by pressing **AVS Applications**) has a new **AVS5 Features** button. **AVS5 Features** provides access to sample networks that illustrate AVS 5's new features in four visualization areas: Imaging, Volume Rendering, New UCD Features, and Miscellany. We encourage you to explore these networks to see this release's new and improved functionality in action.

CHAPTER 2

STARTING AVS

$Path—Installing and Finding AVS Anywhere in a Directory Tree

You can install AVS 5 anywhere in a directory tree. Symbolic links in the */usr* partition are no longer required to find it.

Prior Releases

Prior AVS releases assumed that either:

- AVS was installed directly in */usr/avs*. Three symbolic links (*/usr/bin/avs*, */usr/include/avs*, and */usr/bin/avs_dbx*) were necessary to find AVS binaries and include files.
- If AVS was installed elsewhere in a directory tree, then */usr* had to contain one additional symbolic link (*/usr/avs)* that pointed to AVS's actual location.

The **-path** command line option, **Path** *.avsrc* keyword, and **AVS_PATH** environment variable could be used to redirect AVS's attention to other directories in limited situations. However, significant AVS features did not take these into account.

This approach required **root** privilege to install AVS links in */usr*, if not to install the AVS directories themselves. This had several disadvantages:

- Many secure installations will not allow users to install software as **root.**
- In networked installations, a local **root** password does not necessarily confer root privileges on remote hosts.
- Moreover, in networks where many client workstations with local */usr* partitions are accessing AVS on a remote file server host, every client that wished to access AVS had to modify its local */usr* partition.
- If two versions of AVS were available, switching between them required changing the links.

AVS 5 Behavior

AVS 5 introduces some flexibility to this situation by completing the notion of an AVS **Path**. When AVS 5 starts, it checks for a definition of **Path** by various means. A user's *.avsrc* file can use this **$Path** symbol with any keyword that takes a file or directory specification argument. A parallel mechanism exists to find AVS's include files when compiling AVS modules.

Existing AVS 4 users should realize that if you install AVS 5 in */usr/avs*, then nothing is different from AVS 4.

Running AVS with $Path

Suppose you have installed a new release of AVS in your home directory named */users/me/avs*. No links exist in */usr*. To find AVS and all of its associated files, you would:

1. Add */users/me/avs/bin* to your directory search path:

   csh users would add a line like this to one of their startup files, usually either *.cshrc* or *.login*:

   ```
   set path=($path /users/me/avs/bin)
   ```

   sh or *ksh* users would add a line like this to one of their startup files, usually *.profile:*

   ```
   PATH=$PATH:/users/me/avs/bin
   ```

   This definition lets your shell find the AVS executable. The startup files must be re-executed for the new definitions to take effect.

2. Define **Path** for AVS. This may be done in three ways. They are listed in their order of precedence.

   1. Start AVS with the **-path** option:

      ```
      avs  -path  /users/me/avs
      ```

   2. Have the following line in your personal *.avsrc* file:

      ```
      Path  /users/me/avs
      ```

      This line may occur anywhere in the *.avsrc* file.

   3. Define the environment variable **AVS_PATH**.

      csh users:

      ```
      setenv AVS_PATH /users/me/avs
      ```

      sh or *ksh* users:

      ```
      AVS_PATH=/users/me/avs
      export AVS_PATH
      ```

   4. If none of these is explicitly defined, **Path** will default to */usr/avs*.

You should at least define the **AVS_PATH** environment variable. Then, whenever you encounter one of the many file references in AVS documentation that specify the old */usr/avs* path, you can just replace it with *$AVS_-PATH*. For example: */usr/avs/data/image/mandrill.x* would become *$AVS_PATH/data/image/mandrill.x*

The system's *.avsrc* file (*$AVS_PATH/runtime/avsrc*) has been modified to use the **$Path** symbol defined by one of the three methods listed above. For example:

```
ModuleLibraries    $Path/unsupp_mods/Unsupported \
                   $Path/chem_lib/Chemistry \
                   $Path/avs_library/Animation \
                   $Path/avs_library/UCD \
                   $Path/avs_library/Volume \
                   $Path/avs_library/Imaging \
                   $Path/avs_library/Supported
```

Therefore, when AVS begins to execute, the module libraries will be loaded from */users/me/avs/unsupp_mods/Unsupported*, */users/me/avs/chem_lib/Chemistry*, etc.

In prior releases, these module libraries could only be loaded using full absolute directory and file specfications.

You can use the **$Path** symbol in your personal *.avsrc* file.

Any *.avsrc* keyword that takes a file or directory specification can use the **$Path** symbol.

To run a completely different version of AVS, all you need do is change the value of **Path** using any of the methods described above.

Compiling Modules with AVS_PATH

To link with the AVS libraries, a module must be able to find the AVS include files and libraries. Formerly, this was accomplished by having the */usr/include/avs* link point to the real location of the include files. Library files were referenced relative to the symbol **AVS_LIBS** defined in */usr/avs/examples/Makefile*. **AVS_LIBS** was in turn defined relative to a **ROOT** symbol that defaulted to */usr/avs/libs*.

In AVS 5, the example Makefile in *$AVS_PATH/examples/Makefile* has been changed. The symbol **AVS_PATH** now prefixes all include file and library directory references. **AVS_PATH** defaults to */usr/avs*.

To redefine the symbol:

1. Define the environment variable **AVS_PATH**, and
2. Use the **-e** flag on the *make* command to override the definition of **AVS_PATH** (*AVS_PATH=/usr/avs*) at the beginning of the example Makefile. E.g., "make -e *target*".

The Module Generator has been modified to use **AVS_PATH**. If **Path** is anything other than */usr/avs*, the Module Generator will automatically use the **-e** flag when it compiles a file.

Debugging Modules with avs_dbx

Running the *avs_dbx* debugger is only a matter of adding the AVS binary directory (for example, */users/me/avs/bin*) to your default search path. *avs_dbx* is in the same directory as the main AVS binary.

Continuation Character in .avsrc File

The \ (backslash) character continues keyword argument lists across line boundaries for all *.avsrc* options. For example:

```
ModuleLibraries    $Path/unsupp_mods/Unsupported \
                   $Path/avs_library/Supported  \
                   $Path/chem_lib/Chemistry  \
                   $Path/avs_library/Animation
```

In previous AVS releases, argument lists had to be on one—perhaps very long—line.

New Startup Options

There are several new command line options and *.avsrc* startup file keywords.

-name command line option
Name *.avsrc* keyword

The **-name** command line option, and the **Name** *.avsrc* keyword cause the specified name to appear in window manager window title bars instead of "AVS". Names containing blanks or special characters should be enclosed in double quotes (""). For example:

```
     avs -name MyAvs     (command line)
```

Or

```
     Name "My AVS"       (.avsrc file)
```

Widget windows under the control of the Layout Editor will be named with the specified string followed by their corresponding module's designation (for example, **-name MyAVS** causes boolean parameter widget windows to appear as "MyAVS boolean.user.0"). If these names are too long, you can force truncation back to the simple string by appending the ! character to the string (for example, **-name** "**MyAVS!**"). Note that a ! requires surrounding double quotes.

-nodisplay command line option

This unsupported option is used to run AVS without displaying display widgets or output windows. It is useful for running compute intensive networks that do not require user interaction or graphic output. It is also useful in applications where an alternate user interface is provided by the developer.

Despite the option's name, AVS still requires that a valid X display device be given in order to initialize internal program resources such as pixmaps and fonts. Some supported AVS modules and dialog boxes do not recognize this option and still display their windows on the screen. Most of AVS will behave appropriately.

-nomenu command line option
NoMenu *.avsrc* keyword

The **-nomenu** command line option, and the **NoMenu 1** *.avsrc* keyword prevent the main AVS control panel from appearing when AVS is started. It is intended for application developers who use other means to control their application. You would use this as part of the string your application uses to invoke AVS.

-nomenu/NoMenu prevents the control panel from ever appearing. The previous CLI mechanism for doing this would still show the control panel briefly before taking it down.

OldMacros *.avsrc* keyword

The **OldMacros 1** *.avsrc* file keyword reenables AVS 4 macro module creation behavior. See the "Macro Modules: Improved Interface" section of the "Network Editor" chapter in this *AVS 5 Update* manual.

CHAPTER 3

DATA TYPES

Short Field Data Type

AVS 5 fully supports fields containing short data.

The short data type was added to the AVS 4 field data structure. However, no other parts of AVS, including AVS modules, made use of this feature.

In AVS 5:

- The AVS Data Interchange Application (ADIA) will read short data and store it in a short data type field.
- The Module Generator will create field input/output ports with the short data type.
- The main AVS library for manipulating fields, *libflow.a*, has routines and symbols that support the short data type. For example, to allocate a three dimensional field of single-valued shorts one can say:

```
field = (AVSfield_short * )
         AVSdata_alloc("field 3D scalar short", dims)
```

- The existing AVS modules in Table 3-1 have been upgraded to handle short fields. New modules (not listed below) with field inputs/outputs also handle short fields.

FORTRAN Short Accessor Routines

Two AVS FORTRAN accessor routines (**AVSload_byte** and **AVSstore_byte**, *AVS Developer's Guide,* p. A-74) already exist for FORTRAN compilers that do not support the byte (INTEGER*1) data type. The following two FORTRAN accessor routines, with similar functions, have been added to *avs.inc* for FORTRAN compilers that do not support the short (INTEGER*2) data type.

Portability Note: A short is not necessarily two bytes. It is eight bytes, for example, on a Cray Y-MP.

Table 3-1 Modules Upgraded to Support Short Field Data Type

Data Input	Filters	Mappers	Data Output
color range	clamp	arbitrary slicer	Module Generator
data dictionary	colorize geom	brick	compare field
file descriptor	colorizer	bubbleviz	graph viewer
read field	combine scalars	contour to geom	print field
samplers	compute shade	excavate brick	statistics
generate field	contrast	field legend	write field
	convolve	field to mesh	image viewer
	crop	isosurface	
	downsize	orthogonal slicer	
	extract graph	probe	
	extract scalar	thresholded slicer	
	extract vector	volume bounds	
	field math	volume render	
	field to byte, double, float, and int		
	generate histogram		
	interpolate		
	local area ops		
	mirror		
	sobel		
	transpose		

AVSload_short

include *'avs/avs.inc'*
INTEGER AVSLOAD_SHORT *(BASE, OFFSET)*
  INTEGER *BASE, OFFSET*

This function loads a short integer from memory. This is useful for FORTRAN compilers that do not have an INTEGER*2 data type.

Inputs

BASE	base address in an integer variable.
OFFSET	index into array. In the FORTRAN tradition, the first array element's index is 1.

Returns

Value of a signed short integer.

For example, this reads values from a scalar short field:

```
data_ptr = AVSfield_data_ptr (input_field)
do n = 1, field_size
   ... = AVSload_short (data_ptr, n)
enddo
```

AVSstore_short

include *'avs/avs.inc'*
AVSSTORE_SHORT *(BASE, OFFSET, VALUE)*
   INTEGER   *BASE, OFFSET, VALUE*

This subroutine stores a short integer into memory. This is useful for FORTAN compilers that do not have an INTEGER*2 data type.

Inputs

BASE	starting location in an integer.
OFFSET	index into array. In the FORTRAN tradition, the first array element's index is 1.
VALUE	new value for the short integer. The high order bits that do not fit are discarded.

Returns

Subroutine; does not return a value.

For example, this writes values into a scalar short field:

```
data_ptr = AVSfield_data_ptr (input_field)
do n = 1, field_size
   call AVSstore_short( data_ptr, n, ...)
enddo
```

CHAPTER 4

IMAGE VIEWER

Image Viewer: 8-Bit Scalar and Any-Data Images

The Image Viewer will now display 8-bit scalar (byte) images in addition to the 4-vector byte true color images it has always supported. Since scalar byte images no longer have to be converted to 4-vector byte prior to entering the Image Viewer, only one fourth the amount of memory is required to display them.

You can also input any data type (float, short, double, integer) into the Image Viewer. It will convert non-byte data to byte and normalize it to the range 0-255 before display. The field that contains the image must be 2D and uniform.

The scalar byte images can be displayed in two ways:

colormap
The **image viewer** module now has an optional colormap input port. Connect a colormap to this port (for example, from the **generate colormap** module) and the **image viewer** module will use the image's byte values as indices into the colormap.

grayscale
Without a colormap, the image will be displayed as grayscale. The byte values are used as indices into a 256-element gray ramp colormap.

Limitations

Scalar byte and any-data images can only enter the **image viewer** module through a network. You cannot read them in using the Image Viewer's **Read Image** button.

You cannot use the set of buttons/techniques found under the Image Viewer's **Image Processing** submenu on scalar byte images. If you select a processing technique, nothing happens. However, you can send the scalar byte field through any image processing network and have it displayed in the Image Viewer.

Image Viewer: Scalar Images and Colormaps on Pseudo Color Displays

8-bit pseudo color display heads often have only one 256-entry hardware colormap available at a time. All running applications, including AVS, must share this colormap.

Often, this restricts AVS's ability to represent colors as expected. When started on an 8-bit pseudo color X server, AVS tries to allocate 242 colormap cells to itself: 6 red x 6 blue x 6 green, plus 26 gray tone cells. If 242 cells are not available, it automatically falls back to more limited values (5 red x 5 blue x 5 green, plus 17 gray cells, etc.) until it succeeds. (You can control this manually with the **Colors** *.avsrc* keyword.)

The Image Viewer now contains a button, **colormap**, that will let AVS temporarily appropriate all 256 cells of the hardware colormap. You can use this button to improve the pseudo color and grayscale rendition of scalar byte images. (Its use is meaningless with 4-vector byte true color images).

The **colormap** button is found at the bottom of the Image Viewer's **Views** submenu. It (and the various dithering option buttons on the **Image** submenu) only appears when AVS is started on an 8-bit pseudo color display.

If you turn on **colormap**, then—whenever the mouse cursor is inside the Image Viewer scene window—AVS will use the entire 256 element hardware colormap.

- If no colormap is attached to the **image viewer** module's colormap input port, the system colormap is loaded with a 256 element grayscale colormap. This will make a substantial improvement in grayscale rendition (maximum of 26 shades increased to 256 shades).
- If a colormap is attached to the **image viewer**'s colormap input, then the system colormap is loaded with that colormap. Scalar images will use this 256 entry colormap instead of the more limited 6 6 6 case (216) or possibly 5 5 5 case (125).
- If you are using the Image Viewer from the main menu, rather than as the **image viewer** module, pressing **colormap** will load the 256 element grayscale colormap, but the colormap will have no effect. Scalar images can only enter the **image viewer** module through a network.

Note again that the colormap swapping only actually occurs when the mouse cursor enters the Image Viewer's view window. It is normal for the rest of the display to turn odd colors as the other applications lose their colormap. When the mouse cursor leaves the Image Viewer window, the original default hardware colormap is restored.

Image Viewer: Drawing in Image Viewer Windows

The **image viewer** module now supports interactive "sketching" operations. These sketching operations allow you to use the left mouse button to perform functions such as:

- Interactively draw an arbitrarily-shaped **region of interest (ROI)** on top of an image (Figure 4-1).
- Pick two points in an image and report the distance between them (mensuration).
- Draw a line on an image and display the values in the Graph Viewer.

The **image viewer** module does not do this work alone. It requires an upstream module(s) to interpret the mouse events captured in the **image viewer** window.

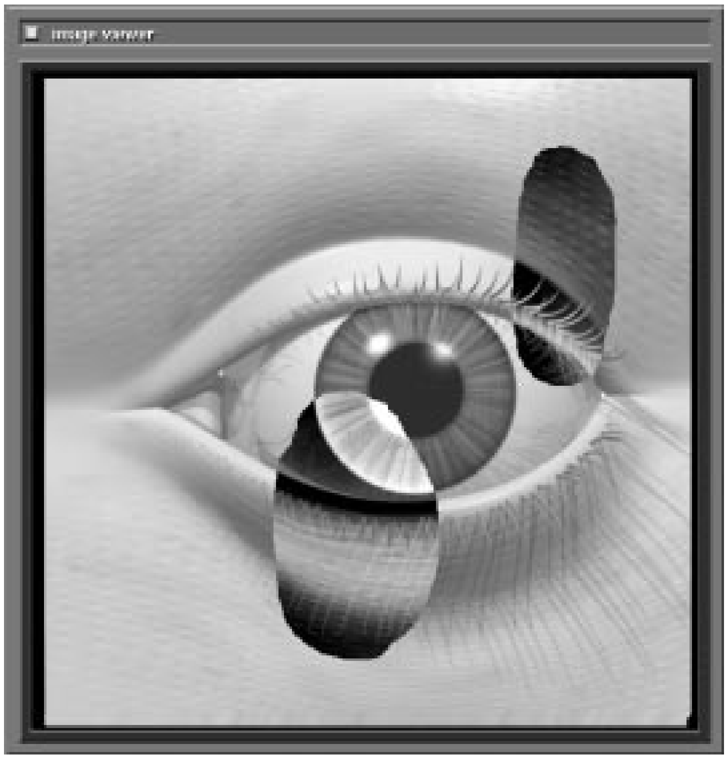

Figure 4-1 Image with Two Interactively Draw Regions of Interest

The sketching mechanism has three parts:

- There are two new upstream data structures (**struct image viewer** and **struct mouse info**). These structures supplement the AVS 4 **struct image pick**. The **image viewer** module outputs these structures. Upstream modules can define input ports to receive them.

- The **image viewer** now understands a new set of CLI commands. An upstream module can send these CLI commands to the **image viewer** module directing it to perform functions such as: inform the module of selected left mouse button events, draw a line on top of an image from point A to point B in image space, clear all lines drawn on an image, etc.
- Drawing modules usually have a **set pick mode** oneshot button on their contol panels. Pressing this button tells the Image Viewer that the user will be using the left mouse button to draw on the image. (The Image Viewer will not interpret the left mouse button as its usual "set current image" function.)

  Alternately, the Image Viewer has a set of **Left Mouse Button** controls at the top of the Image Viewer's control panel that let the user define what the left mouse button will be used for: picking current images, picking current labels, defining subimages, or drawing on an image.

The **image viewer** module's job is to capture the user's mouse events on an identified image and pass them to an upstream module via the new structures. The upstream module does most of the "sketching" work by sending CLI commands to the **image viewer** module.

The combined effect of these features provides the interactive sketching and picking capability. Five supplied modules, **sketch roi**, **image measure**, **image probe**, **calc warp coeffs**, and **ip read line** use the new upstream structures and CLI commands. Users can write their own modules that use these mechanisms to produce similar effects. The example module *$AVS_PATH/examples/doodle.c* illustrates how to do this.

The next section describes how to use the new interactive features of the **image viewer** module. The section following that describes the programmer's interface.

Drawing on Images

Left Mouse Button Controls

Figure 4-2 shows the new Left Mouse Button controls on the Image Viewer's control panel.

These buttons define what will happen when you push the left mouse button in the Image Viewer window.

Pick Image

Pushing the left mouse button will select the current image.

This is the default.

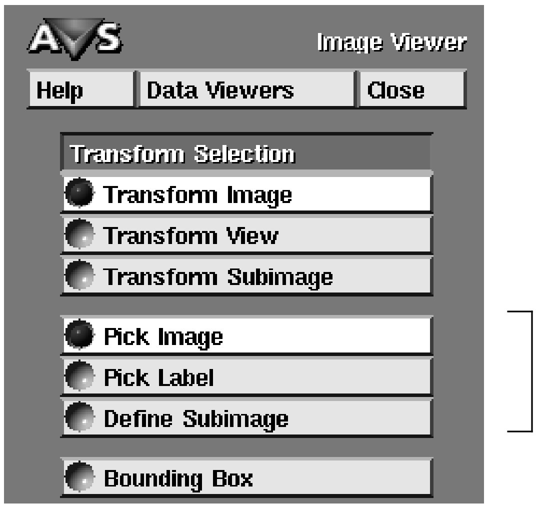

Figure 4-2 Left Mouse Button Controls

Pick Label
Pushing the left mouse button will select a current label. If you continue to hold down the left mouse button, you can move the label around the view. **Note:** In this release, **Pick Image** and **Pick Label** are identical.

Define Subimage
Pushing the left mouse button will define a rectangular subimage for image processing operations selected in the **Image Processing** submenu. Push the left mouse button, and, while still holding it down, move the cursor to define the rectangular subimage. Release the left mouse button to complete the subimage definition.

If **Define Subimage** is selected, you can still select a current image without switching back to **Pick Image**. Use shift-left mouse button to select a current image.

An Interactive "Sketching" Network

Figure 4-3 shows a network that includes the **sketch roi** module. **sketch roi** lets you draw arbitrarily-shaped and sized regions of interest (ROI) on top of an image. The region of interest so-defined is passed as a 2D uniform byte field to modules in the Imaging module library (**ip convolve**, **ip arithmetic**, **ip dilate**, etc.) that accept an optional ROI input. These modules use the ROI as a mask, restricting their operation to just the ROI rather than the whole image.

Notice these connections:

- The output of **ip arithmetic** is sent to both **image viewer** and to **sketch roi**. **sketch roi** must have **ip arithmetic**'s output image (as opposed to, say, **read image**'s output image) so that it can tell the **image viewer** which image it needs mouse events on, and which image it will draw on.

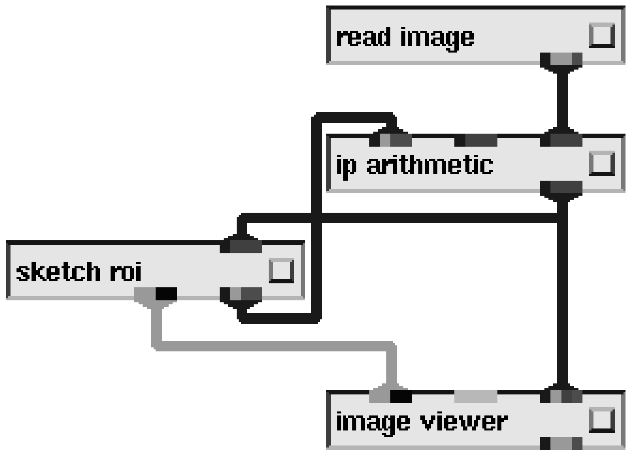

Figure 4-3 Network with Drawing Module

sketch roi also uses this image to make its ROI mask the same size as the image.

- **sketch roi**'s image draw structure output is connected to the **image viewer** module's corresponding input. This connection automatically establishes an upstream connection between **image viewer** and **sketch roi**.
- **sketch roi**'s ROI field output is connected to **ip arithmetic**.

There are two invisible connections going from **image viewer**'s structure outputs upstream to **sketch roi**'s structure inputs. These connections occur automatically—you don't need to connect them. If you were to make them visible with the Module Editor's Parameter Editor, they would look like Figure 4-4.

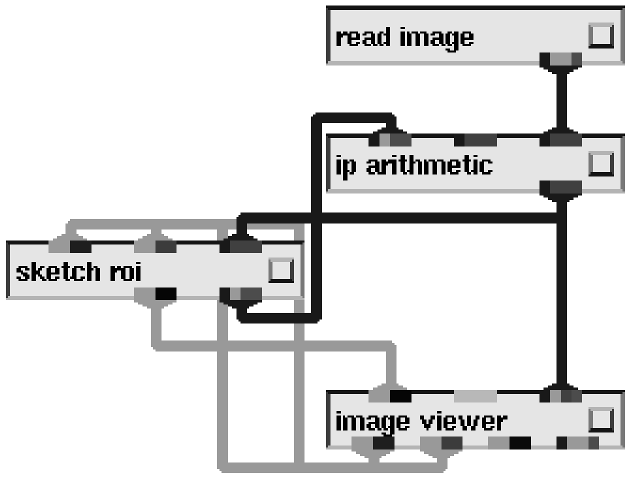

Figure 4-4 Network with Upstream Connections Visible

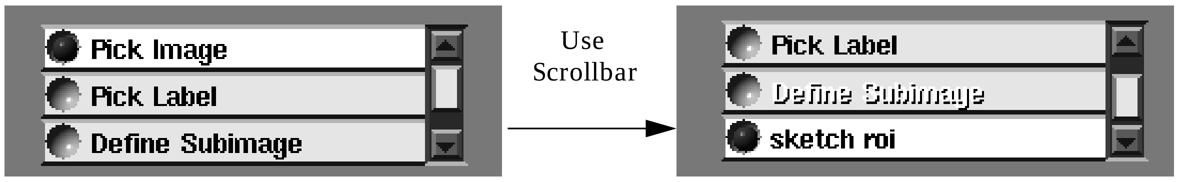

Figure 4-5 Left Mouse Button Controls with Drawing Module Attached

When you create a network that contains a "sketching" module (**sketch roi**, **image measure**, **image probe**, etc.), when data first passes through it, that sketching module is added to the Left Mouse Button control's list and is automatically selected. See Figure 4-5. As the figure shows, you may need to use the scrollbar to make the module's new button appear.

Having **sketch roi** selected on the Left Mouse Button control list means that, when you press and hold down the mouse's left button, you are drawing a ROI, not picking a current image or a label, or defining a rectangular subimage. See Figure 4-6.

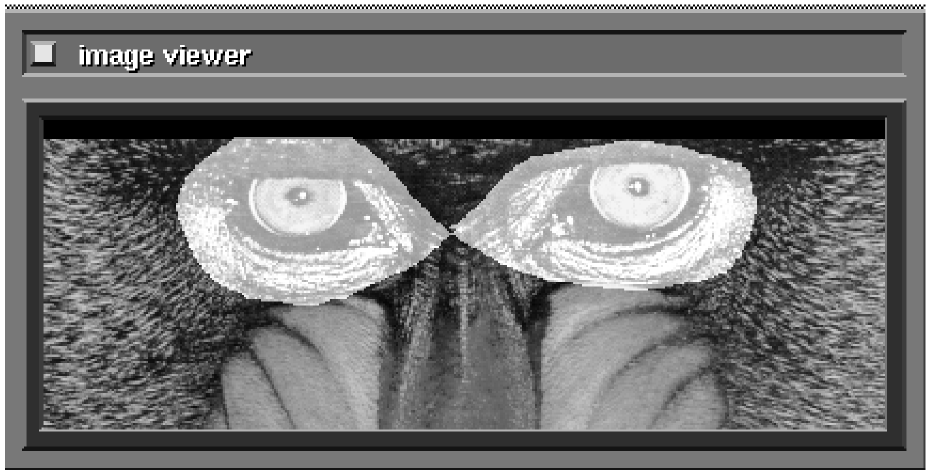

Figure 4-6 Result of Sketching a ROI

With **sketch roi** selected, you can still select a current image. Use Shift-left mouse button.

See the module man pages for **sketch roi**, **image measure**, and **image probe** for specifics on using each one.

A More Complicated Network

The network in Figure 4-7 contains several "sketching" modules (**sketch roi** and **image measure**) and several image processing modules (**ip arithmetic** and **ip linremap**). Notice that they are working on *different* images.

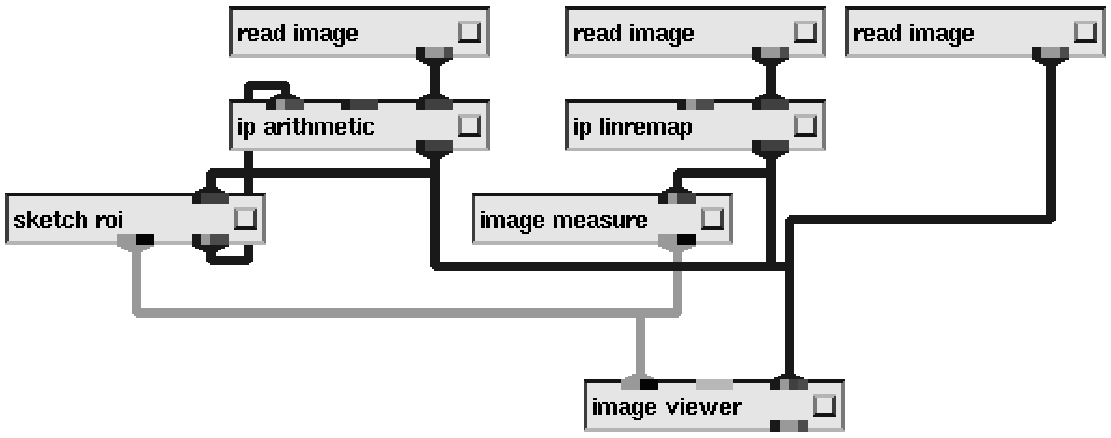

Figure 4-7 A More Complicated Network

Observe that:

- **image measure** will be added to the Left Mouse Button control list.
- the last drawing module connected will be the initial default selected in the Left Mouse Button control list.
- **sketch roi** is associated with the far left **read image** image. If this is not the current image, **sketch roi** is grayed out in the Left Mouse Button control list, since it has no meaning for the other images.
- The far right **read image** image has no "sketching" modules associated with it. If you make it the current image, **Pick Image** becomes the default in the Left Mouse Button control list.

ROIs and Subimages

The Image Viewer has always supported rectangular subimages. These subimages were defined using the shift-left mouse button. The subimages could be acted upon using the various techniques available under the **Image Processing** submenu.

There is no relationship between the existing rectangular subimage/**Image Processing** menu mechanism and the new region of interest mechanism created by **sketch roi**.

Programmer Interface to Image Viewer Upstream Data

The Image Viewer's upstream data ports can be used to interactively track mouse button press/release and motion events. These events can then be used by the upstream module to interactively draw on images via the CLI.

This section describes the programmer’s interface to this mechanism. It is best understood while referring to the *$AVS_PATH/examples/doodle.c* example module source code. *doodle.c* uses the facilities described in this section. It also demonstrates accessory skills, such as loading and flushing a CLI command buffer, that are necessary to make the mechanism work.

Establishing Connections

Data Structures

An upstream module that requires upstream data needs to define three additional I/O data ports.

struct image_viewer_id

This "Image Viewer ID" upstream input port identifies the Image Viewer **scene_id**. It has the following structure:

```
typedef struct _image_viewer_id {
   int  scene_id;
 } image_viewer_id;
```

struct mouse_info

This "Mouse Info" upstream input port contains most of the data that the module will use to identify the image in the Image Viewer, and interpret mouse button events. It has the following structure:

```
typedef struct _mouse_info {
   char          image_name[64];
   char          func_id[64];
   int           mesh_id;
   int           nEvents;
   float         image_x[32];
   float         image_y[32];
   int           buttonMask[32];
} mouse_info;
```

struct image_draw

This "Image Draw" output port is used to establish the automatic connections to the Image Viewer module, and to synchronize with the Image Viewer. It has the following structure:

```
typedef struct {
   int   integer;
} image_draw;
```

Include Files

The two upstream input data structures, **image_viewer_id**, and **mouse_info**, are described in the include file *$AVS_PATH/include/udata.h*. The module's source code will need to include this file.

The third data structure, which is the **image_draw** output port, is described in the include file *$AVS_PATH/include/image_draw.h*. The module writer will also need to include this file in the module source code.

In addition, there are various mouse button event mask values defined in the include file *$AVS_PATH/include/image_upstrm.h*.

Defining I/O Ports in the Module's Description Routine

The following code, placed in the module's description routine, defines the additional ports.

```
int port;

port = AVScreate_input_port("Image Viewer Id","struct image_viewer_id",
           INVISIBLE | OPTIONAL);
AVSset_input_class(port, "Image Draw:image_viewer_id");

port = AVScreate_input_port("Mouse Info","struct mouse_info",
           INVISIBLE | OPTIONAL);
AVSset_input_class(port, "Image Draw:mouse_info");

AVScreate_output_port("Image Draw", "struct image_draw");
```

The two input ports are generally made invisible, while the "Image Draw" output port is a visible port.

Image Draw Port

The **image_draw** structure is defined as follows:

```
typedef struct {
   int   integer;
} image_draw;
```

The integer data sent by this "Image Draw" output port is never actually used. However, it is required for two reasons. First, it is needed to make the automatic connections of the corresponding image viewer ports to the upstream input ports. And second, it is used for synchronization purposes in order for the Image Viewer to execute after an upstream module. This second reason not only requires that the upstream module's "Image Draw" port be connected to the Image Viewer's "Image Draw" port, but also that the module allocate data for the **image_draw** structure.

Allocating Data for the Image Draw Port

The **image_draw** structure is really user-defined data and so the module must load this type in its description routine. The module description routine should include the following line:

```
AVSload_user_data_types("<$AVS_PATH>/include/image_draw.h");
```

where <$AVS_PATH> is an automatically computed string that reflects the actual location of the AVS directory.

Image Viewer ID Port

The **image_viewer_id** structure is defined as follows:

```
typedef struct _image_viewer_id {
   int  scene_id;
 } image_viewer_id;
```

The first upstream input port is the "Image Viewer ID" port. The **scene_id** is an id that is used with CLI commands to identify to which Image Viewer instance the CLI command is being sent. All of the CLI commands that draw on images require a **scene_id**. The upstream module will not be able to draw on an image or request mouse events until this input port has valid data.

Typically, a module that requests mouse events on an image will have access to the AVS field that describes the image. Therefore, the image field that is an input to the Image Viewer must also be an input to the upstream module. If the upstream module creates its own output image field that is connected to the Image Viewer, then it will automatically have access to the field.

Mouse Info Port

The **mouse_info** structure is defined as follows:

```
typedef struct _mouse_info {
   char          image_name[64];
   char          func_id[64];
   int           mesh_id;
   int           nEvents;
   float         image_x[32];
   float         image_y[32];
   int           buttonMask[32];
} mouse_info;
```

Initialization: image_wakeup_when_defined CLI Command

Once the module has valid data on its "Image Viewer ID" port and has an image field for which it wishes to request mouse events, it then asks the Image Viewer to place data on its "Mouse Info" port indicating that the Image Viewer knows about the image. The CLI command to do this is

```
image_wakeup_when_defined -scene_id <val> -mesh_id <val> -module $Module.
```

The **scene_id** comes from the "Image Viewer Id" port. The **mesh_id** is an integer that is part of the field struct. And the **module** name is the name of the upstream module.

Once the Image Viewer knows about the field that contains the same **mesh_id**, it then sends a "Mouse Info" output to the upstream module stating that the image is defined. The **mesh_id** element of the **mouse_info** structure will contain the same **mesh_id** as the image_wakeup_when_defined CLI call. The **image_name** element of the **mouse_info** structure will contain the name of the image. All of the CLI commands that request events for an image require the image name.

Distinguishing Mouse Info Initialization from Mouse Events

As will be explained below, the **mouse_info** structure is also used to send mouse button press/release and motion events. The way the upstream module distinguishes whether the **mouse_info** structure is a response to an image_wakeup_when_defined CLI command or a list of mouse events is that in the former case the **nEvents** element of the **mouse_info** structure will always be zero, while in the latter case it will be greater than zero, indicating the number of mouse events.

Selecting Events: image_select_events CLI Command

After the upstream module receives input on its "Mouse Info" port as a reponse to the image_wakeup_when_defined CLI command, it can then use the name of the image to request mouse events on the image. The following command requests mouse events for a particular image.

```
image_select_events -scene_id <val> -image_name <name>
             -module $Module -func_id <func_name> -mask <val>
```

The **scene_id** is taken from the "Image Viewer ID" input port. The **image_-name** comes from the **mouse_info** structure that was a response to the image_wakeup_when_defined CLI command. The **module** name is the name of the upstream module. The **func_id** is a string that will be attached to a radio button on the Image Viewer panel. When the user selects this radio button all selected events will be sent to the upstream module through the "Mouse Info" port.

The **mask** is a bit mask that specifies what events the module requests. The following mask:

```
int mask = IMAGE_Button1Mask | IMAGE_PressMask |
           IMAGE_ReleaseMask | IMAGE_MotionMask;
```

requests press/release and motion events for button1. Button1 is the left button. Currently, events can only be selected for the left button. The above constants are defined in *$AVS_PATH/include/image_upstrm.h.*

The image_select_events CLI command only selects events for a particular image. Therefore, the appropriate mouse events are sent to the upstream module only when the currently picked image corresponds to the image for which events were selected, and the radio button corresponding to the **func_id** is selected. The user will not be able to select the radio button when another image is the currently picked image.

The upstream module may call image_select_events multiple times on the same image but with a different **func_id** and possibly a different mask. Each time image_select_events is called with a **func_id**, a new radio button is added to the Image Viewer panel that corresponds to the **func_id**. When mouse events are sent to the upstream module, the **func_id** element of the **mouse_info** structure will contain the corresponding **func_id** name.

Acquiring Left Mouse Button Events: image_set_pick_mode CLI Command

It can be inconvenient for the user to have to manually select the radio button on the Image Viewer panel that corresponds to the **func_id** of the module. The following CLI command will automatically select the left mouse button to the upstream module:

```
image_set_pick_mode  <mode>
```

where the <mode> is one of **Pick Image**, **Pick Label**, **Define Subimage**, or the **func_id** defined by the module that wants to gain control of the left mouse button.

Note: image_set_pick_mode should be placed after image_select_events in the initialization portion of the module code as it is in *doodle.c*. This establishes an initial condition for the left mouse button when the module is initialized and data first passes through it. The user may subsequently take action that moves the left mouse button focus, particularly if there is more than one image in the Image Viewer window or there are multiple drawing modules, each using image_set_pick_mode.

Stopping Events: image_stop_events CLI Command

The following CLI command is used to stop events that were previously selected with image_select_events.

```
image_stop_events -scene_id <val> -image_name <name>
             -module $Module -func_id <func_name>
```

This command will remove the radio button corresponding to the **func_id**.

mouse_info Event Structure

When a mouse_info structure is sent to an uptream module as a result of mouse events, the elements of the **mouse_info** structure will have the following meaning:

image_name
The name of the image for which events are being sent to the upstream module.

func_id
The corresponding **func_id** name.

mesh_id
Undefined.

nEvents
Number of mouse events listed in following array.

image_x[32]
x location of mouse relative to image.

image_y[32]
y location of mouse relative to image.

buttonMask[32]
Button mask specifying state of buttons and modifier keys.

image_x and Image_y Coordinate System

The coordinate system in which the location of the mouse is returned through **image_x** and **image_y** is relative to the image and is pixel based. The upper left corner of the image is (0.0, 0.0). The bottom right corner of the image is (image_width,image_height). The center of the top-left pixel, for example, is located at (0.5, 0.5). The upstream module can receive mouse events outside the image and at fractional pixel coordinates (e.g., 7.35, 25.91).

If the coordinate data ("points data") array for the image is defined to be different than its dimensions (real world spacings), then upstream modules can use this information. **image_probe** and **image_measure**, for example, make use of this.

buttonMask Mouse Events

The **buttonMask** specifies the type of mouse event: IMAGE_PressMask, IMAGE_ReleaseMask or IMAGE_MotionMask. An AND of the **buttonMask** with the above masks will indicate the type of the event. The mask also specifies the state of two of the modifier keys: the shift key and the control key. The masks IMAGE_ShiftMask and IMAGE_ControlMask are used to determine if the corresponding keys are pressed or not. The above constants are defined in *$AVS_PATH/include/image_upstrm.h*.

nEvents and Event Buffering

The **nEvents** element of the mouse_info structure indicates the number of mouse events that are included in the **image_x**, **image_y**, and **buttonMask** arrays. There can be between 1 and 32 events. If the upstream module can execute at interactive rates, there will generally be a few mouse events for each execution of the module. However, if the module takes a relatively long time to perform its computation, then there will most likely be a lot of events (<= 32) for each execution of the module. The Image Viewer buffers events, so all events will eventually be delivered to the module.

The CLI Interface for Drawing on Images

Once a module has a valid **scene_id** and an **image_name**, it can then send CLI commands to draw lines on the image.

Draw a Line: image_add_line CLI Command

The following CLI command is used to add a line to an image:

```
image_add_line -scene_id <val> -image_name <name>
            -module <module_name> <x1> <y1> <x2> <y2>
```

The line is drawn from (x1,y1) to (x2,y2). x1, y1, x2, and y2 are floating point numbers that are defined in the same coordinate system as the **image_x** and **image_y** elements of the **mouse_info** structure.

The Image Viewer keeps track of all the lines a particular module draws on an image. When that module's "Image Draw" port is disconnected from the module, the Image Viewer will delete all its lines.

Deleting Lines: image_clear_all_lines CLI Command

It is only possible to add lines to an image, but not to edit the list of lines that have been added to the image. However, it is possible to clear all the lines with the CLI command:

```
image_clear_all_lines -scene_id <val> -image_name <name>
             -module <module_name>
```

This command will delete all the lines that the corresponding module added to the image.

The image_clear_all_lines CLI command also resets the line width and color to the defaults.

Setting Line Width: image_set_line_width CLI Command

It is possible to draw thick lines by using the following command:

```
image_set_line_width -scene_id <val> -image_name <name>
             -module <module_name> <width>
```

All lines that are drawn following a CLI command to set the line width will be drawn with the new width. The width of a line is specified as an integer number of pixels. The default width is zero. A zero width line is the thinest single-pixel line that can be drawn in an efficient manner.

Setting Line Color: image_set_line_color CLI Command

In the same way, the line color can be set by the following command:

```
image_set_line_color -scene_id <val> -image_name <name>
             -module <module_name> <red> <green> <blue>
```

The red, green and blue values specify the intensity of the corresponding component as a floating point number between 0.0 and 1.0. The default color is white (r,g,b) = (1.0, 1.0, 1.0).

Drawing XOR lines for Rubber-Banding: image_add_xor_line CLI Command

In addition to the image_add_line CLI command, it is possible to draw lines in xor mode.

```
image_add_xor_line -scene_id <val> -image_name <name>
             -module <module_name> <x1> <y1> <x2> <y2>
```

This command can be used to draw rubber-banded lines. Drawing an xor line on an image will produce a line on the image. Drawing the same xor line again will erase the line. The Image Viewer does not save information about xor lines as it does with image_add_line. It is the responsibility of the up-stream module to draw xor lines in pairs in order to erase them.

CLI Command Buffering for Performance

When an upstream module executes, it is possible that it will make multiple CLI calls. For performance and efficiency, it is important that the module buffer multiple CLI calls and send them as a single CLI command buffer.

The *doodle.c* example module contains a set of functions that create and initial-ize such a CLI command buffer, add CLI image drawing commands to that buffer, and periodically flush the buffer down to the Image Viewer.

image_refresh CLI Command

The Image Viewer now supports an image_refresh CLI command:

```
image_refresh
```

Like the Geometry Viewer's geom_refresh command, image_refresh redraws the current Image Viewer scene window. In the process, it updates the output image, if any, on the Image Viewer's image output port. This resolves the problem some users encountered when using the **image to postscript** module wherein old, out-of-date images were output instead of the current visible scene.

Image Output and Image Viewer Performance

The **image viewer** module's image output port can now be turned off to improve Image Viewer performance. The same change was made to the Geometry Viewer. Thus, this new feature is discussed in detail in the section titled "Image Output, Image Viewer and Geometry Viewer Performance" in Chapter 5.

CHAPTER 5

GEOMETRY VIEWER AND LIBGEOM LIBRARY

Outline Gouraud and Hardware Renderers

All platforms on which AVS uses hardware rendering now support the **Outline Gouraud** rendering technique. The AVS software renderer has always supported **Outline Gouraud**.

Sending Scene Windows to a Different Display

It is possible to send the output of each Geometry Viewer camera to a different X Window System display. This is useful, for example, if your workstation has multiple display heads, and you want the AVS interface on one screen and the Geometry Viewer's scene window on another screen.

To send the Geometry Viewer camera output to a different display:

1. Make one output window the current camera, either by clicking in its background with a mouse button, or with a CLI command. The current camera window is surrounded with a red border.
2. In most cases, you must to switch to the software renderer for the current camera. Under the **Cameras** submenu, select **Software Renderer**.

   Caution on Hardware Renderers: If you have **Hardware Renderer** toggled instead of **Software Renderer**, then when you press the Enter key in step 4 (below) you will get a message box that says:

```
The selected camera is not running the Software Renderer.
Please ensure that the remote device can handle the current
renderer before proceeding.
Shall we change the display to hydra:0.1?
```

   Two choices are given: **OK** and **Cancel**

   If you get this message, you should press **Cancel** until you verify that you will be able to do remote hardware rendering.

   If you press **OK**, then AVS will attempt to initialize a hardware renderer on the remote device of the same type as that running on the local device. Only a few systems (PEX to PEX, or DGL to DGL) are able to support remote hardware rendering. If you press **OK** and AVS is not able to initialize a hardware renderer of the same type on the remote device, AVS will abnormally exit.

A similar message, situation, and result will occur if you already have a remote display window created with the software renderer, and you attempt to toggle **Hardware Renderer**.

3. Under the **Cameras** submenu, near the bottom, there is a **Display** typein. It shows the X Window System display of the current camera. Edit this to be the X display you want the window to appear on.
    - The format is standard X (*host:Xserver.screen#*). For example: hydra:0.1.
    - The other display must allow you to create windows on it. Permission mechanisms may vary. Standard X would use an *xhost* command entered on the remote host that gives your host permission to create windows.
4. Press the Enter key to make the window appear on the other screen. It is removed from the original screen. It remains the current camera.

The remote display window is identical to a local window in all behaviors (mouse button input, lighting, etc.) except:

- Though you can click in the remote window to make it the current camera, it will not display the red border that usually signifies it is the current camera.
- The first camera / window created by a **geometry viewer** module is surrounded with an AVS interface "container." Subsequent cameras / windows are not containerized.

  A containerized window moved to a different display will lose its container. (The second display's window manager will likely parent the window, while the original container disappears on the original display.) The container is not restored if the window is moved back to the original display.

Colormap Messages

You may get messages related to colormap cell allocation when you try to create the remote window. The remote display is under all of the same colormap constraints as your primary display. The specifics of this issue on each platform is usually discussed in detail in the release notes that accompany AVS.

Two messages are possible:

Warning: using fewer colors on hydra:0.1 (R=3,G=3,B=3,Grey=5)
This message appears in the terminal emulator window from which AVS was started. It means that AVS was able to allocate enough colors on the remote display to create a picture, but fewer than the optimal R=6, G=6, B=6, Grey=26.

Display (hydra:0.1) does not have enough free colormap entries

This message appears in an error message box. It means that AVS was not able to allocate enough colors on the remote display to create a picture.

These messages tend to appear when:

- The remote display is 8-plane (pseudocolor) and there are other applications running on the remote display that are using too many colormap cells. For example, the file-finder applications on some systems use many colormap cells. The only "fix" is to remove the other processes that are competing for the colormap.
- The remote display is 24-plane (truecolor or directcolor) but its default X visual type is PseudoColor. (See the "AVS on Color X Servers" appendix in the *AVS User's Guide* for information on using the *xdpyinfo* command to determine the default X visual.) In this case, you can either:
    - Reconfigure the remote system's X server startup mechanism to start the truecolor display with the correct TrueColor or DirectColor X visual. Or,
    - Start AVS on the local device with the **VisualType** *.avsrc* keyword. **VisualType** would be set to **TrueColor** or **DirectColor**, depending on the platforms. For example:

      ```
      VisualType  TrueColor
      ```

      This will work if both devices are really truecolor, and it will also work in the case where the local device is pseudocolor and the remote device is truecolor. In this second case, AVS prints a warning for the local device and falls back on the correct pseudocolor, but still produces truecolor on the remote device.

geom_set_display CLI Command

This functionality is also available through the CLI. The command is:

```
geom_set_display  display  -camera n
```

where *display* is in X Window System display specification format, and *n* is the number of the camera to display remotely.

Image Output, Image Viewer and Geometry Viewer Performance

The **geometry viewer** and **image viewer** modules have an image output port. This port is used for a variety of purposes: to send the contents of the window to the **image to postscript** module for hardcopy; or to output a sequence of images that are part of an animation.

This image output port does affect the performance of the viewer. If a module such as **image to postscript** is connected to this image output port, then each time the scene in a window changes the **image** or **geometry viewer** converts the contents of the frame buffer to an AVS image and sends the output to the downstream module. This conversion and output transmission slows down the viewer. The performance difference is barely noticeable with the **geometry viewer** operating with the software renderer. It can be very noticeable when using the **image viewer** or the **geometry viewer**'s hardware renderer.

The AVS 5 **image** and **geometry viewer** modules have two new input parameters that let you control whether a new image is output each time a scene changes. Both parameter ports are invisible by default.

Update Always

This is a boolean switch. If it is **on**, then the viewer generates an output image each time a scene changes. If it is **off**, the viewer will only generate an image when its **Update Image** oneshot is fired.

For backward compatibility with previous releases, **Update Always** is **on** by default. That is, by default the viewers will generate an image whenever a scene changes.

Update Image

This is a oneshot. Any input received on this port causes the viewer to convert the frame buffer to an image and send it to the downstream module. It is active regardless of how **Update Always** is set.

Note that these switches only affect performance if a downstream module is connected to the **image** or **geometry viewer**'s image output port.

These two buttons do not appear on any Image or Geometry Viewer menu. Rather, the interaction is the same as any module with normally-invisible parameter ports:

1. Make the parameter port visible:
    1. With **image** or **geometry viewer** in the Workspace, click on its dimple with the center or right mouse button. This brings up the Module Editor.
    2. Under the Parameters list, click on **Update Always**. This brings up the Parameter Editor.
    3. On the Parameter Editor, click on both **Port Visible** and **toggle**.
2. Repeat for **Update Image**, but choose **oneshot** instead of **toggle**.
3. Press **Close** on the Module Editor to take down its panel.

This has two effects:

- The parameter ports become visible on the **image** or **geometry viewer** module icon.
- The **image** or **geometry viewer** acquires a traditional module control panel on the stack at the left of the screen. The viewer's page contains the two **Update Always** and **Update Image** buttons. Set these buttons however you wish. This setting will be saved if you perform a **Write Network** operation.

There are two ways to use these parameters:

- Just click on them in the module control panel.
- Or, connect the **oneshot** module to the **image** or **geometry viewer**'s **Update Image** parameter port, and the **boolean** module to the **Update Always** parameter port. (Both **oneshot** and **boolean** are in the Data Input column of the Module Palette.) If you choose this second method, it is not necessary to choose **toggle/oneshot** when making the parameter ports visible.

```
oneshot        boolean
     |             |
     |---| |------|
         | |
        geometry viewer
```

There is one subtle side-effect of this operation. Clicking on the **image** or **geometry viewer** module's dimple with the left mouse button will now raise the viewer's page in the module control panel stack—it will no longer raise the full Image or Geometry Viewer. To raise the full viewer, use the **Data Viewers** pull down menu at the top of the control panel.

The CLI **parm_set** command will work on these parameters regardless of their visibility.

Non-Transformable Objects

It is possible to use the *libgeom.a* library to mark an individual object as "non-transformable." Though the user can select the object as the current object, it will be unresponsive to any transformation instructions coming from an input device or a CLI transformation command.

The **color legend** module produces such a non-transformable object.

The non-transformable attribute is implemented as an additional mode argument called **locked** (all lowercase) on the **GEOMedit_transform_mode** subroutine call. It is in addition to the existing mode arguments **normal**, **parent**, **notify**, and **redirect**. Any **flag** option should be 0. The **locked** attribute will extend to the object (name) and all its child objects. For example:

```
GEOMedit_transform_mode(*output_geom,parent_name,"locked",0);
```

locked accomplishes its purpose by making the object part of *screen space*, extending from -1 to 1 in X, Y, and Z. Therefore, all positioning of the object within the view window should take this into account. Note that if a user resizes the geometry output window, the object will be rescaled accordingly.

Note: locked objects should *not* be placed at -1.0 or 1.0 in Z. Rather, a programmer who wants the locked object, for example, in front of all other objects should place it at .99 in screen space. Some hardware renderers do not render objects located at -1.0 or 1.0 in screen space.

The **locked** object may still be affected by lighting and color property editing on some renderers. It is therefore recommended that you set the object's render mode to **no light** with **GEOMedit_render_mode**.

If you also wish the object to be unpickable, use the **GEOMedit_selection_-mode** call with the "ignore" option:

```
GEOMedit_selection_mode(*output_geom,object_name,"ignore",0);
```

Texture Maps

AVS 5 has two new *libgeom.a* library calls that provide greater 2D and 3D texture mapping functionality. In AVS 5, a programmer can:

- Attach a separate texture map to each individual object in a Geometry Viewer scene window.
- Associate a colormap with each texture map.

These are independent, though related features. One can associate a texture map with an individual object with **GEOMedit_field_texture** without having to colorize that texture with **GEOMedit_texture_colormap**.

GEOMedit_field_texture

GEOMedit_field_texture(*list, name, fieldptr***)**
GEOMedit_list *list;*
char **name;*
AVSfield **fieldptr;*

This routine can be used instead of the existing **GEOMedit_texture** call. It attaches a texture-mapping "field" to the named object's edit list.

This routine takes a 2D or 3D field specified by the pointer *fieldptr*, creates *uv[w]* texture-mapping information for vertices of the object specified by *name*, and adds it to the edit list *list*. The field must be a uniform byte field. The field can be either scalar or 4-vector. If the field is scalar, then the texture map consists of 8-bit index values that will need to be associated with a colormap by a call to **GEOMedit_texture_colormap**. If the field is a 4-vector byte, then it is interpreted as alpha, red, green, blue color values.

The *fieldptr* attached to the edit list object is a reference to the field. Only the reference is passed to downstream modules.

GEOMedit_texture_colormap

GEOMedit_texture_colormap(*list, name, colormap***);**
GEOMedit_list *list;*
char **name;*
AVScolormap **colormap;*

This routine appends the AVS colormap pointed to by *colormap* to the edit list information of the object *name*, in the edit list *list*. It is used to assign color values to the vertices when performing colorized texture mapping.

It is assumed that *name* has previously had texture-mapping information assigned to its vertices with the **GEOMedit_field_texture** routine.

If *colormap* is 0, then this call removes any color information that may have already been assigned to *name*.

This call increases the size of the edit list by the size of *colormap*. This is 1024 bytes (256*4 bytes).

Platform dependence. This feature requires that the underlying renderer support 2D or 3D single component colorized texture maps. The software renderer supports this. No hardware graphics subsystems currently do. See the table of platform geometry rendering features in the "Release Notes" chapter of your platform’s *Installation and Release Notes*.

Network Example

A network containing a module that is coded to use both **GEOMedit_field_-texture** and **GEOMedit_texture_colormap** is not constructed in the same way that networks containing one of AVS’s existing texture-mapping modules are constructed.

This is the typical texture mapping network used by modules such as **brick** that use the older **GEOMedit_texture** call (assumes a uniform scalar field):

```
                    read field
                          |
generate colormap         |
             |            |
             |   |--------|-------------|
             |   |        |             |
            colorizer   brick      volume bounds
                 |        |             |
                 |----|   |-------------|
                      |   |
              geometry viewer
```

The key modules are **brick**, **colorizer**, and **geometry viewer**. **brick** produces a geometry of eight surfaces of the field. Which eight surfaces are determined by **brick**’s parameters. **colorizer** takes each of the original data values in the field and converts it into a 4-vector byte color value. **brick**’s geometry and **colorizer**’s 3D texture ("field of colors") are sent to the **geometry viewer**. The **geometry viewer** takes the 3D texture from its field input port, and the geometry from its geometry input port, and uses the 3D texture’s color values that intersect the geometry planes to produce a colorized geometry that is displayed as the texture mapped object.

There are two problems with this approach:

geometry viewer

The **geometry viewer** module has only one "texture mapping" input port. Thus, only one texture map can be sent. This texture map will be applied either:

- automatically to the object that **brick** specifies on its **GEOMedit_texture** call ("GEOMedit_texture(*output, brick_object, dynamic)"),
- or, if a module does not make this call, then manually to whichever object is the current object when the user presses **Set Dynamic Texture** on the **Edit Texture** panel.

memory utilization

colorizer produces an output field that is either half the size of the input field (double input), equal to its size (int and float input), double its size (short input), or quadruple (byte input) its size. Since most uniform volumes are bytes or shorts, the typical effect is to create a field four times the size of the original field.

A module (for example, a hypothetical **new brick**) that uses **GEOMedit_-field_texture** and, optionally, **GEOMedit_texture_colormap** would be included in a network constructed as follows:

```
                    read field
                           |
generate colormap          |
              |            |
              |----------| |-------------|
                         | |             |
                      new brick      volume bounds
                           |             |
                           |-------------|
                           |
              geometry viewer
```

The differences are:

- There could be multiple **new brick** modules, multiple input fields, and multiple output geometries, each with its own texture map.
- **colorizer** and its space-consuming 3D texture has been eliminated. This **new brick** takes a colormap input and the input field and uses **GEOMed-it_field_texture** and **GEOMedit_texture_colormap** to create the colored geometry. This texture mapped geometry is sent to the **geometry viewer**, using the software renderer for display.

Note: **brick** (and the similar **excavate brick** and **volume render**) were not modified in this release because their prototypical old and new network structures would be incompatible. On some hardware renderers that supported 3D texture mapping, but not single component colorized textures, **brick**, **excavate brick,** and **volume render** would no longer work.

Pixmap Output with Window ID

The **geometry viewer** module now has an invisible AVS pixmap output, like the obsolete **render geometry** module. This is provided for those users who require the X window system id of **geometry viewer**'s output window. The pixmap data type is described in the "AVS Data Types" chapter of the *AVS Developer's Guide.*

Integer Synchronize Output

The **geometry viewer** module now has an invisible integer output port. This "signal" output is generated whenever the Geometry Viewer re-renders. It is provided so that the Geometry Viewer can synchronize with a module that might control a video camera or other device. Usually, this integer output is more efficient than attempting to synchronize off the image output.

New CLI Commands

The Geometry Viewer supports two new CLI commands:

geom_center

The geom_center CLI command lets you automatically set the center of rotation and scale operations for a Geometry Viewer object. The center is set to the XYZ location that is in the center of the object's extents. It is equivalent to pressing the **Center** button on the Geometry Viewer interface.

```
geom_center { -object <name> }
```

-object defaults to the current CLI object. This will be the current Geometry Viewer object unless it has been explicitly set with set_cur_cli_obj.

geom_get_position

The geom_get_position CLI command lets you find the current position of an object. In order to support the rotation / scale center function, the Geometry Viewer stores the object's position separate from the 4x4 transformation matrix. The XYZ position vector is essentially the location of the center of scale / rotation of the object in the object's coordinate system.

```
geom_get_position  { -object <name> }
```

-object defaults to the current CLI object. This will be the current Geometry Viewer object unless it has been explicitly set with set_cur_cli_obj.

CHAPTER 6

GRAPH VIEWER

Graph Viewer Geometry Output

In previous releases, the Graph Viewer would only output a geometry representation of its plot window when the plot was first drawn, or when a new dataset entered one of its input ports. There is now a button and a CLI command that forces the Graph Viewer to generate an output geometry.

Output Geometry

The **Output Geometry** button on the Graph Viewer's **Write Data** submenu forces the Graph Viewer to output a geometry version of its plot window on its geometry output port. You would use this button, for example, to write a new geometry when you make a change using the Graph Viewer's interface, such as changing the plot style from line to scatter.

The Graph Viewer will now only generate a geometry output when this button is pressed or the graph_output_geom CLI command is issued.

graph_output_geom CLI command

The graph_output_geom CLI command produces the same effect in a CLI script as pressing the **Output Geometry** button in the Graph Viewer's interface.

CHAPTER 7

NETWORK EDITOR

Macro Modules: Improved Interface

The AVS 4 Network Editor introduced macro modules: the ability to group sets of modules together so that they appear and behave as though they were one module. This functionality is accessed through the **Editing Tools** menu on the Network Editor's main panel. It is documented in the "Editing Tools: Macro Modules" section of the *AVS User's Guide*'s "Advanced Network Editor" chapter.

In AVS 5, creating and editing macro modules has been greatly improved. You can now create macro modules quickly "in place." The overall process is the same as in AVS 4, with these exceptions:

Existing modules and connections are not destroyed.

In AVS 4, when you started the macro creation / editing process by pressing **Create** or **Edit Macro Module**, the existing module instances and the connections between them and the rest of the network were saved and then destroyed. A Clear Network operation was performed, blanking the Workspace. You were placed in a mode where only the **IN->** and **OUT->** macro module I/O stub modules, and any selected modules, were present. When you finished creating or editing the macro module, you had to reconnect it to the restored network.

In AVS 5, you create and edit macro modules "in place" in a running network. There is no network buffer, and no Clear Network operation. Existing module instances, their parameter settings, and connections are untouched. The entire existing user interface context of the application is still available while you edit the macro module.

Macro modules are not saved or added to the Palette automatically.

In AVS 4, to complete creating or editing a macro module, you pressed the **Done Editing Level: *n*** button. At this point, you were always prompted for a filename in which to save the description of the macro module. Once the file was written, the macro module's new icon was added to the module Palette.

In AVS 5, macro modules are not saved to files nor added to the module Palette until you explicitly specify these operations. To save a macro module description to a file and add it to the module Palette, you must press **Write Network** under the Network Editor's **Network Tools** menu while you are editing the macro module. You will receive a prompt asking if you want to save the entire network or just the macro module.

Automatic IN-> and OUT-> Connections

If you have selected a group of modules (left mouse button/drag/release) to be part of a macro module, when you enter the editing mode any connections that existed between this block of modules and the rest of the network will be automatically added to the **IN->** and **OUT->** modules.

When you exit editing mode, these connections will automatically be made between the new macro module and the existing network. This makes it extremely fast and simple to make macro modules out of blocks of an existing network.

Keyboard shortcuts

There are two keyboard shortcuts:

Function	**Keyboard Shortcut (Left Mouse Button)**	**On**
Edit Macro Module	shift-click double-click	macro module icon
Done Editing	shift-click double-click	**IN->** or **OUT->** module icon

Option to CLI module command to change macro module name.

The -newname option to the CLI module command will change the name of macro module. For example:

```
module old.user.0 -newname new
```

will rename macro module **old** to **new**.

Scripts incompatible with AVS 4

There is one incompatibility between the AVS 5 and AVS 4 macro module mechanisms: CLI scripts that create and edit macro modules will not work under the new scheme. If you have such scripts, you can either:

- recreate them under AVS 5, or
- edit them to include the following CLI command:
  ```
  debug EDITORmacro_mode 0
  ```
  This will turn on AVS 4 macro module behavior.
- run AVS 5 with AVS 4 macro module behavior using the **OldMacros** option described below.

OldMacros option to renable AVS 4 behavior

If you want to create and edit macro modules in the AVS 4 fashion, add this line to your personal *.avsrc* file:

```
OldMacros  1
```

Dial Widget Behavior

Some subtle changes have been made to the behavior of dial widgets. The changes all relate to setting the resolution of a dial (how many units are traversed in one revolution of the dial's pointer), and the behavior of dials with respect to minimum and maximum boundary values. Some of the changes are "bug fixes"—corrections to either behavior or documentation. Some are new features that provide functionality not previously available.

This section updates/corrects the text on "Using Dial Controls" on pp. 6-24 through 6-27 of the *AVS User's Guide.*

Dial Resolution

Bounded Dials

These changes appear on the Dial Editor of bounded dials.

Min/Max

As in AVS 4, the **Min** and **Max** typeins set the resolution of the dial, where the resolution is defined as:

(Max - Min) / 270 = units per degree of needle rotation

Min and **Max** do *not* establish absolute minimum and maximum dial bounds. These are set by the module.

You use the **Min** and **Max** typeins to increase the resolution of the dial over a subrange of the module-defined absolute minimum and maximum dial bounds. For example, if the dial bounds were from 0 to 1000, the dial's needle would be too sensitive to use to set precise values ((**Max** - **Min**)/270 degrees = 3.7 units per degree of rotation). You can set **Min** to 100 and **Max** to 200 to make the resolution (200-100)/270 = .3 units per degree of rotation).

In AVS 4, a user could enter **Min** and **Max** resolution values that exceeded the module's defined dial bounds. For example, though the module had defined bounds of 0 to 64, the user could set **Min** and **Max** resolution values of 0 to 100. The dial would change appearance in a way that suggested that this had worked. However, AVS would just ignore the parameter value when it exceeded the module's defined bounds, and the needle would "stick" at the real minimum (0) or maximum (64) value.

In AVS 5, if a user types in **Min** or **Max** values that exceed the module's defined dial bound(s), the value is just reset to the module's defined bound(s).

Reset Min/Max

This new button will reset the **Min/Max** resolution values to the original module bound settings. Subsequent changes by the module will be automatically reflected by the **Min/Max**.

Saved Networks

In AVS 4, if a user had manually reset the dial **Min/Max** resolution, then it was impossible to save the network, read it in again, and restore the correct **Min/Max** resolution values. The values supplied were wrong. In AVS 5, this functionality works correctly.

Developer Issues

To save widgets to a network file, AVS uses the CLI manipulator command (*AVS Developer's Guide*, pp. 5-47 to 5-48). The clause that establishes the widget's parameter settings is the -P clause:

```
manipulator  ... { -P <prop name> <prop type> <prop value> }
```

The -P clause is the CLI equivalent of the **AVSadd_parameter_prop** library routine. Indeed, to make sense of -P's arguments, you have to look up the explanation of the **AVSadd_parameter_prop** call in the *AVS Developer's Guide*, pp A-10 to A-12.

In AVS 4, bounded dials used the **local_range** property name to save user-set Dial Editor **Min/Max** values. As noted above, this did not work.

In AVS 5, bounded dials use two new property names: **local_min** and **local_max**, to hold these values. Their corresponding property type, data type, and widget types are shown in Table 7-1. This is a shortened version of the table on page A-11 of the *AVS Developer's Guide*.

Table 7-1 Property Name, Data Type, and Widget Type Correspondence

Property Name	Property Type	C Data Type	FORTRAN Data Type	Widget Types
local_range	real	**float**	**real**	dial, idial
local_min	real	**float**	**real**	dial, idial
local_max	real	**float**	**real**	dial, idial

Thus, when a network containing bounded dial widgets whose **Min/Max** values have been changed by the user is written to a file, it contains manipulator CLI commands that look like this:

```
manipulator  ...  -P local_min real new-min-value
                  -P local_max real new-max-value
```

One can issue similar CLI commands interactively or in a script to set a bounded dial widget's resolution.

One can write a module that contains an **AVSadd_parameter_prop** call

like the following to set a bounded dial widget's resolution:

```
AVSadd_parameter_prop(param_num, "local_min", "real", new-min-value)
AVSadd_parameter_prop(param_num, "local_max", "real", new-max-value)
```

This would be used to override the parameter's absolute bounds, just as if the user had used the Dial Editor.

Unbounded Dials

These changes appear on the Dial Editor of unbounded dials.

Min typein not ignored

In AVS 4, the **Min** typein value was actually ignored. Instead, the **Max** typein value was taken as the sole numerator when calculating:

(Max - Min) / 360 = units per degree of needle rotation

In AVS 5, both values are used in the equation.

Range

A **Range** typein has replaced the **Min/Max** typeins on the unbounded dial Dial Editor to set dial resolution.

The default resolution of the unbounded dial’s face is "once around is 100." (Note that the *User’s Guide* says "once around is 200." This is incorrect.)

Setting this typein establishes the new "once around is *n*" dial resolution.

Reset Min/Max

This new button will have no effect on unbounded dials.

Saved Networks

As with bounded dials, in AVS 4, if a user had manually reset the dial resolution, then it was impossible to save the network, read it in again, and restore the correct resolution values. The values supplied were wrong. In AVS 5, this functionality works correctly.

Developer Issues

Unbounded dials are now the sole users of the **local_range** property name, as found in the CLI manipulator command’s -P clause, and the **AVSadd_parameter_prop** library routine.

CHAPTER 8

DEVELOPING MODULES

Introduction

This chapter describes major new AVS 5 features available to module developers. You may also wish to consult the "Documentation: Clarifications / Corrections" chapter for improved descriptions of several existing features already documented in the *AVS Developer's Guide.*

Module Generator

Reinitialize

In AVS 4, to clear all Module Generator settings you had to "hammer" the **Modle Generator** module and drag down a new instance of **Module Generato**r from the Palette.

In AVS 5, the Module Generator has a **Reinitialize** button at the top of its main control panel. Pressing this button zeroes all settings and starts creating a new module.

Support for Widgetless and Unbounded Parameters

The Module Generator's Parameter Editor panel has two new buttons: **No Widget** and **Unbounded**. (To make the Parameter Editor panel visible, press **Edit Parameters** on the Module Generator's main control panel.)

No Widget

Produces a parameter without an associated widget (widget type "none") by generating an **AVSconnect_widget(*param_num,* "none")** call. Widgetless parameters are used, for example, by the **AVS Animator** module. The Animator creates numerous parameters that it controls internally using CLI commands, and which have no user-visible or controllable widgets.

Unbounded

A module will often use unbounded parameters and widgets when its author cannot predict a parameter's range, or when the author wants the parameter range to be data-dependent, with the bounds only being established during execution in the compute function with an **AVSmodify_parameter** call.

Previously, the Module Generator had no interface to produce unbounded parameters. All parameters were bounded and the user had to hand edit the **AVSadd_parameter** call in the generated C or FORTRAN file to change a parameter to unbounded.

Now, an **Unbounded** button appears for all float parameters and "normal" integer parameters such as isliders, idials, and integer_typein.

When this button is on, the widgets created for a parameter are unbounded, and the **Min/Max** typeins at the bottom of the Module Generator's Parameter Editor are removed. The default is off.

Modifying Fields in Place

The typical model for an AVS filter module is:

1. Free previous output field (if any) with **AVSfield_free**.
2. Create a new output field with **AVSdata_alloc**.
3. Copy data from input field to output field with some processing along the way.

There are many cases where the output field is the same type and size as the input field. In many of these cases it would be desirable to operate on the data in place rather than make an additional copy of it. This is possible in AVS 5 and later versions.

Rather than using **AVSdata_alloc** to allocate a new field, **AVSfield_equiv_-to_input** is used to create an additional reference to an existing field. For instance (from *$AVS_PATH/examples/threshold.c*):

```
static int thresh_compute_2(input, output, pmin, pmax)
AVSfield_float input,**output;
float *pmin,*pmax;
{
     if (*output != NULL)
         AVSfield_free (*output);

     *output = AVSfield_equiv_to_input(input);

     /*
      * Perform the calculations on input field here.
      */

     return(1);
}
```

By default, input fields in shared memory are Read Only so an attempt to modify the data will result in an error. To make them writable, use the **READ_WRITE_IN** option. Example:

```
in_port = AVScreate_input_port("Input Field",
    "3D uniform scalar float", REQUIRED | READ_WRITE_IN);
```

By using both the **READ_WRITE_IN** option and **AVSfield_equiv_to_inpu**t, a long chain of modules can access the same data and modify it in place, resulting in large memory savings.

This feature is also useful for building a switch module that simply passes an input field through to an output without making a copy. The sample module in *$AVS_PATH/examples/switch.c* has a few input ports and a single output port. One of the input ports is passed along to the output depending on a parameter. A related use would be a module with a single field input port and several outputs. It could send that input to zero, one, or more outputs. In these cases, the field is not modified so the **READ_WRITE_IN** option is not required for **AVScreate_input_port**.

Side Effects of Modifying Fields in Place

Users of modules using this feature should understand the consequences to avoid unwanted side effects. Let's look at a simple example:

```
+---------------+
| Module A      |
| Read Image    |
+---------------+
         |--------------------------|
         |                          |
+---------------+                   |
| Module B      |                   |
| Some Filter   |                   |
+---------------+                   |
         |                          |
         |                          |
+---------------+          +----------------+
| Module C      |          |  Module D      |
| Display Image |          |  Display Image |
+---------------+          +----------------+
```

Suppose Module B is some sort of image processing module that modifies the image in place. There is no guarantee whether Module D would fire with the original image from A or the result of B.

The AVS flow executive (version 5 and later) will try to run modules that have specified Read / Write access to their inputs after others connected to the same source. In the example above, when A produces new output, the flow executive will fire D before B. However it is possible for D to run again, after B has modified A's output, due to changes in another input port or parameter.

Note on Supplied AVS Modules

None of the modules supplied with AVS 5 modify their input data in place.

C++ and ANSI-C in AVS 5

Introduction

AVS 5 provides improved support for both ANSI-C and C++ modules. This support consists of:

- optional function prototype declarations for all AVS provided functions (required for both languages), and
- example modules that illustrate proper coding practices.

AVS 5 does not provide specific C++ base classes for immediate use by programmers. It does provide a C++ example using a sample base class that illustrates how subclasses can be derived.

Function Prototypes

The AVS header files now have definitions for all AVS functions in both Kernighan and Ritchie-style C declarations (i.e., "extern int AVSset_module_name();") and as the function prototypes needed for ANSI-C and C++.

All of these new function prototype declarations are defined using a macro, AVS_EXTERN. AVS_EXTERN itself is defined in *$AVS_PATH/include/avsargs.h*. *avsargs.h* is automatically included by all of the AVS header files.

In *avsargs.h*, a compiler definition called NeedFunctionPrototypes is automatically set to 1 when either __STDC__ or __cplusplus is defined.

- If NeedFunctionPrototypes is 1, then the AVS_EXTERN call turns into a function prototype declaration.
- If NeedFunctionPrototypes is set to 0, then AVS_EXTERN turns into a Kernighan and Ritchie-style C declaration without the function arguments being declared.

The user may override the automatic setting by predefining NeedFunctionPrototypes to the desired value before any header files are included.

To Ensure Module Compatibility

Since AVS functions were not previously declared, there may be some discrepencies between what is now declared and what users once had to declare in their own code. To turn off any function declarations from the header files, define the compiler option AVS_EXTERN_NO_DECLS to 1. This can be done by including the directive "-DAVS_EXTERN_NO_DECLS=1" on your compile line.

General Practices for ANSI-C and C++ Compatibility

Several example modules have been copied and modified so that they can be compiled by a C++ compiler: *polygon2.cpp*, *read_image2.cpp*, *read_ucd2.cpp*, and *widgets2.cpp*. These have been altered so that they are acceptable to either a C compiler or a C++ compiler. The changes required were not substantial but are worth enumerating. These changes *are not required* for normal C code modules. They are illustrations and tests of the ANSI-C / C++ interface.

flow.h

The <*avs/flow.h*> include file is not needed and should generally not be included unless there is a very specific reason to do so.

In the past *flow.h* has been included by module writers to get the FLOW_-MODULE structure declaration needed by **AVSmessage**. In fact, the **AVSmessage** function's third argument should have a NULL value. It should not be **AVSmodule** or any module pointer. The value will be figured out correctly inside the function. The *flow.h* header file is incompatible with C++ and has been removed from all example files.

AVSinit_modules()

In C++, this needs to be enclosed by a special "extern" declaration because this function is still coded in Kernighan and Ritchie-style C so that the *libflow* library can find it. For example (from *$AVS_PATH/examples/read_image2.cpp*):

```
/* C++ wrapper is required for AVS C functions to find
this function whenit is compiled by C++ compiler */

#if __cplusplus
extern "C" {
#endif

static int read_image();

void
AVSinit_modules()
{
    AVSmodule_from_desc(read_image);
}

#if __cplusplus
}
#endif
```

Compute function declaration

This must be a function prototype to match the later declaration of the actual function. This is also true of other local functions in the same file, depending upon the order of function declarations. Use AVS_STATIC (a static version of AVS_EXTERN) to declare it appropriately. When passing it to **AVSset_compute_proc**, cast it as an AVS_FNCP type value. For example (from *$AVS_PATH/examples/read_image2.cpp*):

```
/* In C++, the compute function must be declared as a prototype function */

AVS_STATIC( int read_image_compute, (AVSfield_char **data, char *filename));

static int
read_image()
{
    /* ... */

    /* Set the compute procedure */
    AVSset_compute_proc((AVS_FNCP)read_image_compute);

    /* Initialization and Destruction functions should also be cast
       when present:

       AVSset_init_proc((AVS_FNCP)read_image_init);
       AVSset_destrot_proc((AVS_FNCP)read_image_destroy);
    */

    /* ... */
}
```

return values

Functions that are implicitly declared to return an int value should return that int value in order to avoid warnings.

casting

Some function calls may need to cast their arguments to match the function prototype in order to avoid compiler warnings. The following examples shows both declaring return values and casting (from *$AVS_PATH/examples/read_image2.cpp*):

```
static int
read_image_compute(data, filename)
    AVSfield_char **data;
    char *filename;
{
    /* ... */

    /* AVSfield pointers should be cast as "generic" AVSfields to match
       the function prototype declarations */

    if (*data)
      AVSfield_free((AVSfield *) *data);

    /* AVSfield pointers return values may also need to be cast
       to specific subtypes */

    input = (AVSfield_char *)AVSdata_alloc("field 2D 4-vector 2-space byte", dims);

    /* Make sure that functions declared to return a value do so */
    return(1);
}
```

exit()

In C++, the **exit()** function is declared in <*stdlib.h*>. This file should be included conditionally if __cplusplus is defined and the module actually requires it.

C++ Example

The example Makefile in *$AVS_PATH/examples/Makefile* has been extended to include references to the local C++ compiler and its related libraries. CPP is used to name the C++ compiler. CPPLIBS is used to name the related libraries. You may need to redefine these environment variables on your target machine.

```
# In order to use C++, you will need to modify the CPP and CPPLIBS definitions
# to match your local system conventions.
CPPFLOWLIBS=$(FLOWLIBS) -lc -lC -lc
CPPSIMLIBS=$(CSIMLIBS) -lc -lC -lc
CPPHOME=/usr
CPPLIBS=-L$(CPPHOME)/lib
CPP=$(CPPHOME)/bin/CC
CPPFLAGS=$(CFLAGS) $(CPPLIBS)
```

To make the examples, type:

```
make -e cpp_examples
```

The C++ examples consist of:

- The modified C examples *polygon2.cpp*, *read_image2.cpp*, *read_ucd2.cpp*, and *widgets2.cpp*. These programs are "ANSI-C and C++ conforming" but are not object-oriented,
- A sample C++ module, *cpp_example.cpp,* that demonstrates a base class (in *cpp_base.cpp* and *cpp_base.h*) with a subclass built upon it.

All of these examples can be built using "make cpp_examples" after you have defined the CPP macros appropriately.

The *cpp_example.cpp* module builds on a base class called AVSModule defined in *cpp_base.h* that defines constructor and destructor functions. The constructor function, **AVSModule::AVSModule**, accepts a module name, a compute function pointer, and a module type value. It then calls the appropriate AVS *libflow* functions to declare the module. The header file declares a number of inline functions to call **AVSadd_parameter** and other commonly used AVS *libflow* functions.

The *cpp_example.cpp* file contains a subclass based on AVSModule called Derived. Derived sets up a constructor function, **Derived::Derived**, that uses the base class functions to call *libflow*. It also provides a compute function. Overall, this is a very simple example, but one that shows one way to employ C++ when writing an AVS module.

Compiler Warnings

You may receive compiler warnings when you compile the example modules. Warnings will be of two general types. The exact text may vary depending upon the compiler.

function declaration anachronism

In addition to function prototypes, ANSI-C has a similar function header where the parameter types are declared inline. ANSI-C compilers may complain about Kernighan and Ritchie-style C type declaration, but will usually accept them. For example:

```
"read_image2.cpp", line 83: warning: old style defini-
tion of read_image_compute() (anachronism)
```

variable not used

The examples, adapted from the existing *$AVS_PATH/examples* module examples, were not totally rewritten to make the ANSI-C and C++ correct. Thus, the compiler may warn of unused variables.

Unmapped Control Panels

Some developers may create an interface in which they do not want a module's control panel to appear on the screen. Two options, **-hide** and **-show** have been added to the CLI **module** command that makes this possible.

This is the new help text for the CLI **module** command:

```
avs> help module

The module is created if it does not already exist and its name is
printed out. If the module exists it is changed to match the given
arguments. If the name is given without a ".user.N" prefix it is assumed
to be a request to create a new module; the new name will be printed out.
The pend operations connect the module in between other  modules or remove
it from a previous pend operation. A tag is an added identifier that allows
a module to be referenced as part of a group of related modules.

Usage: module <module.user.N>
{-xy X,Y}         # location of module in network editor work space
{-host <host>}    # remote host name
{-alias <alias>}  # set a unique module alias name
{-ex <module_path>}# location of module executable/file
{-on/-off}        # enable or disable the module
{-hide/-show}     # hide or show module owned widgets             <--New options
{-parent <macro name>}    # this module is contained in the macro module
{-macro }         # this module is a macro module
{-unshared}       # prevents the module being mapped out during readnet
{-prepend <module:port>}  # insert module before module port
{-prepend_tag <tag>}      # insert module before tagged group
{-postpend <module:port>} # insert module after module port
{-postpend_tag <tag>}     # insert module after tagged group
{-tag <tag>}              # set or change the module tag ("" to clear)
{-type_tag <type_tag>}    # set or change the type tag ("" to clear)
{-unpend }                # remove module from pended connections
```

The **-hide** and **-show** options work both when the module is initially created, and later on as a modification operation. They cause both the main module panel and any other module-related panels to be hidden/shown. They only affect the top most panel in a hierarchy. Therefore, a developer does not need to do recursive **-show** operations later. Panels that are hidden can be shown using either the original option on the CLI **panel** command, or these new options on the **module** command.

When a module is in the "Top Level Stack", its button will still show even though the module panel itself is hidden. If the panel is within another panel, or if it is a root level window, this will not be an issue. However, most panels appear within the stack.

CHAPTER 9

MODULES AND MODULE LIBRARIES

Introduction

This section describes:

- The AVS 5 reorganization of module libraries.
- New AVS 5 modules.
- Existing modules that have significant behavior changes in AVS 5.

The module changes are summarized here. See the *AVS Module Reference* manual for complete new / updated module man pages.

Module Libraries

Default Libraries

AVS 5 loads all AVS module libraries by default at system startup. The module libraries appear as a scrolling list at the top of the Network Editor control panel.

This change was accomplished by changing the **ModuleLibraries** line in the system default *$AVS_PATH/runtime/avsrc* file to read as follows:

```
ModuleLibraries    $Path/unsupp_mods/Unsupported \
                   $Path/chem_lib/Chemistry \
                   $Path/avs_library/Animation \
                   $Path/avs_library/FiniteDiff
                   $Path/avs_library/UCD \
                   $Path/avs_library/Volume \
                   $Path/avs_library/Imaging \
                   $Path/avs_library/Supported
```

Your own local *.avsrc* file can override the system default. Edit its **ModuleLibraries** line to include just those module libraries you wish to have loaded.

New Module Library Classifications

There are now over 240 supported AVS modules. To make it easier to find the module that you want in the Network Editor's Module Palette, the set of AVS modules has been subdivided into eight module libraries. Seven module libraries contain supported modules. One contains unsupported modules. The groupings are designed to minimize the need to switch among module libraries.

The module library structure is depicted in Figure 9-1. The module libraries are:

Supported
Contains all supported AVS modules except the Chemistry and Animation module libraries.

UCD
Contains modules suitable for unstructured cell data networks. Note that all UCD modules are also in the Supported module library.

Finite Difference
Contains modules suitable for finite difference analysis. This includes most of the field modules. Note that all finite difference modules are also in the Supported module library.

Volume
Contains modules suitable for volume visualization. Note that all volume modules are also in the Supported module library.

Imaging
Contains modules suitable for imaging. It includes the 40+ **ip...** imaging modules. These imaging modules are also in the Supported module library.

Chemistry
Contains the chemistry example modules. These chemistry modules are supported although they are not found in the Supported module library.

Animation
Contains the modules that make up the AVS Animation Application. These modules are supported although they are not found in the Supported module library. On most systems, these modules require separate licensing.

Unsupported
Contains the unsupported AVS modules.

Many modules, such as **read field** and **geometry viewer**, are found in multiple module libraries. See the module library module man pages in the *AVS Module Reference* manual for complete lists of which modules are in which libraries.

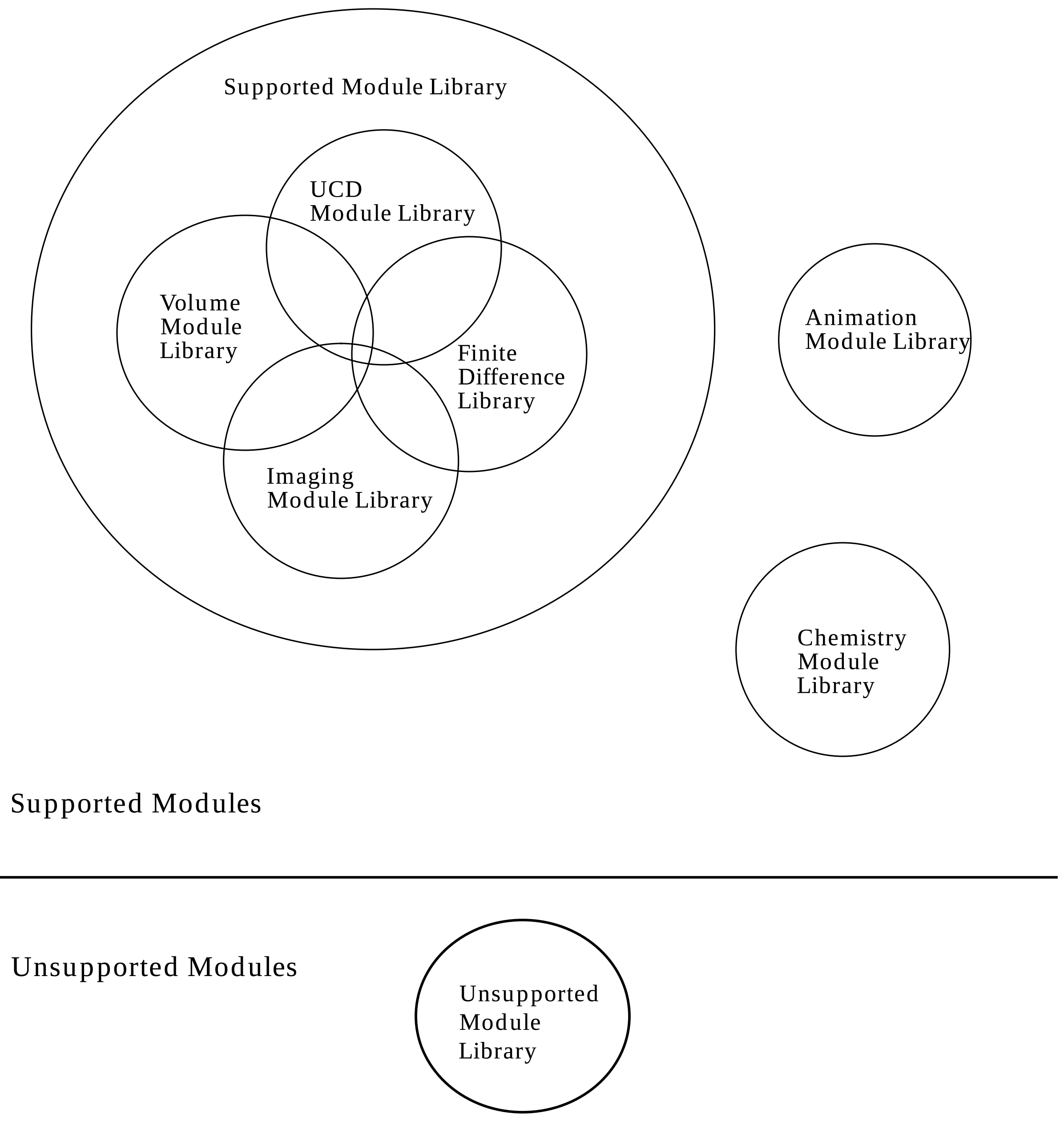

Figure 9-1 AVS 5 Module Library Organization

New and Enhanced Modules

See the *AVS Module Reference* manual for complete module descriptions.

Imaging Modules

AVS 5 contains greatly augmented support for image processing. Most of these modules are adaptations of the SunVision imaging processing library (*iplib*). AVS module "wrapper" code was written to surround the core *iplib* library function calls.

In general, the modules supply this functionality:

- 2D and 3D byte, short, and float images
- single- and multi-band images (maximum of 12 bands)
- support for arbitrarily-shaped regions of interest (ROI) operations, image probing, and image mensuration
- support for morphological, arithmetic, Fast Fourier Transformation, analytic, filtering, and geometric imaging operations

Table 9-1 lists the new imaging modules. The imaging modules are in the binary file *$AVS_PATH/avs_library/sv_multm*.

Table 9-1 New Imaging Modules

Name	Description	Type
calc warp coeffs	create / read tiepoints and calculate warp coefficients for ip warp module	data
ip read kernel	read convolution kernel for ip convolve	data
ip read mtable	read morphology table for ip morph	data
ip read sel	read structuring element for ip dilate, ip erode, ip median	data
ip read vff	convert .vff format image to AVS field	data
sketch roi	compose ROI image from Image Viewer sketched lines	data
draw grid	draw a grid on top of an image	filter
ip absolute	generate absolute values of an image’s pixels	filter
ip arithmetic	add, sub, mul, div, min, max, constant operations	filter
ip blend	alpha or compositing blend of two images	filter
ip contour	draw iso-level contours on images	filter
ip convolve	convolve an image with a float kernel	filter
ip dilate	dilate an image	filter
ip edge	image edge detection by various algorithms	filter
ip erode	erode an image	filter
ip fft	Fast Fourier transform of an image	filter
if fft display	calculate magnitude and phase of a packed FFT image	filter
if fft multiply	multiply two packed FFT images	filter

Table 9-1 New Imaging Modules

Name	Description	Type
ip fft pack	fold conjugate symmetric FFT representation	filter
ip fft unpack	unfold conjugate symmetric FFT representation	filter
ip float math	floating point operations: log, log10, sqrt, exp, recip, cos, sin, atan	filter
ip ifft	inverse Fourier transform for conjugate datasets	filter
ip lincomb	inter-band linear combination	filter
ip linremap	linearly remap an image	filter
ip logical	bitwise logical operations: and, nand, nor, not, or, xor	filter
ip lookup	pass image through a lookup table	filter
ip median	median image filter	filter
ip merge	merge two images	filter
ip morph	morphological operations	filter
ip reflect	reflect or transpose an image	filter
ip rescale	simple linear remap of pixels in an image	filter
ip rotate	rotate an image	filter
ip threshold	threshold an image against a float value	filter
ip twarp	arbitrary image warp from warp table data	filter
ip translate	translate an image	filter
ip warp	polynomial image warp	filter
ip zoom	zoom image with floating point pixel offset	filter
image measure	find distance between two pixels in an image	mapper
image probe	report pixel value at selected point	mapper
ip histogram	compute image histogram	mapper
ip read line	extract pixel values along an interactively drawn sampling line	mapper
ip compare	compare two images	data output
ip extrema	find pixel value extrema	data output
ip register	determine highest correlation offset between image pairs	data output
ip statistics	return number of pixels, mean, variance	data output
ip write vff	write AVS field in .vff raster image file format	data output

New UCD Modules

Table 9-2 is a list of the new unstructured cell data modules.

Table 9-2 New UCD Modules

Name	Description	Type
ucd cell color	color ucd structure based on cell or material id values	mapper
ucd curl	compute the curl of a vector UCD structure	filter
ucd div	compute the divergence of a vector UCD structure	filter
ucd grad	compute the vector gradient of a UCD structure	filter
ucd math	perform math operations between UCD structures	filter
ucd minmax	set min / max values of a UCD component; used with time series data to prevent widget resets; and with color legend module	filter
ucd plot	create a field to graph a linear sample through a UCD	mapper
ucd reverse cell	repair topology of imported UCD structures	filter
ucd rubber sheet	map values as a 3D surface with height proportional to value	mapper
scatter to ucd	convert scatter field to a tetrahedral UCD structure	filter
ucd vol integral	calculate volume of a UCD structure, and the volume integral of a scalar data component	data output

Updated UCD Modules

Table 9-3 is a list of the updated unstructured cell data modules.

Table 9-3 Updated UCD Modules

Name	Description	Type
ucd slice 2D	enhanced for animation: does not reset parameters after execution; has a continuous mode that doesn't require Do Slice parameter to be pressed to produce an output	mapper
ucd streamline	can create colored stream ribbons; will accept a sample point field	mapper
ucd to geom	will create cell-based color contours; has additional port to accept Color Field for this purpose	mapper

New Volume Rendering Modules

Table 9-4 is a summary list of new volume rendering modules.

Table 9-4 New Volume Rendering Modules

Name	Description	Type
compute shade	combines colorizer, compute gradient, and gradient shade module functionality into one memory-efficient module	filter
cube	derived from SunVision volume rendering library. Ray traces uniform volumes in these modes: texture mapping, maximum value, transparent surfaces (based on a substances table), and a Create Surface mode that stores intermediate results for rapid recomputation of surface opacities and colors.	mapper
edit substances	create the substance table for input to cube module	data
time sampler	supports sampling 3D volumes out of time-series data using linear or cubic interpolation.	filter
x-ray	provides orthographic projections of volumetric data. Can render very large datasets.	filter

Updated Volume Rendering Modules

Table 9-5 summarizes updates to volume rendering modules.

Table 9-5 Updated Volume Rendering Modules

Name	Description	Type
excavate	has been moved to the Supported module libraries	filter
tracer	will now render scalar byte fields with a colormap in addition to 4-vector byte fields. This saves considerable memory. Will now handle uniform fields with points / extents information.	mapper
vbuffer	this unsupported module has been removed from AVS	data output

New and Enhanced Data Presentation Modules

Table 9-6 is a summary list of new modules for data presentation.

Table 9-6 New and Enhanced Data Presentation Modules

Name	Description	Type
color legend	creates a color legend for a geometry window. Takes advantage of the new object locking mechanism in the Geometry Viewer.	mapper
generate axes	Creates axes with moveable origin, min / max extents, and independent control of X, Y, and Z axis labels, tick marks, tick length, label spacing, tick mark spacing, label font and height	data
image to CGM	converts an image to CGM format file	data output
label	provides fast support for titling in the Geometry Viewer	data
3D bar chart	converts a 2D floating point field into a group of 3D blocks or planes for viewing in the Geometry Viewer	mapper

Miscellaneous New Modules

Table 9-7 is a summary list of miscellaneous new modules.

Table 9-7 Miscellaneous New Modules

Name	Description	Type
average down	samples x, y, or z data values independently	filter
blend colormaps	interpolates colormaps for the AVS Animator	filter
dialog box	uses a dialog box to create a long string parameter	data
field to short	data conversion for new supported data type	filter
minmax	set field min / max value; used to prevent dial reset during animations	filter
ribbons	generates a ribbon representation from a geometry output of the stream lines module. Also inputs a 3D vector field to control the orientation of the ribbons.	filter
track ball	sends an object transformation to other modules	data

Miscellaneous Enhanced Modules

Table 9-8 summarizes enhancements to miscellaneous existing modules.

Table 9-8 Miscellaneous Enhanced Modules

Name	Description	Type
create geom	moved to Supported libraries	data
field math	handles byte, short, integer, float and double data values with data ranges that match system capabilities	filter
field to float field to int	have new, wider data ranges that match system capabilities	filters
generate grid	moved to Supported libraries	data
geometry viewer	supports optional colorized texture maps; pixmap output; it can be made to output an image only on user command to improve hardware renderer performance when downstream image module is connected; will now output an integer "signal" whenever it rerenders so that it can synchronize with video output modules.	data output
hedgehog	scale value parameter's minimum is now 0.0, not 0.01	mapper
image viewer	accepts scalar images; will color as grayscale or by optional colormap. Accepts any input data type, converting it to 0-255 normalized byte data before display. New upstream data support. Can be made to output an image only on user command to improve performance when downstream image module is connected.	data output

Table 9-8 Miscellaneous Enhanced Modules

Name	Description	Type
luminance	old "luminence" module, with fixed spelling. Old luminence has been moved to the Unsupported library.	filter
pixmap modules	all modules that took pixmaps as input/output have been moved to the Unsupported module library. Note that this includes render geometry and display pixmap. (Use geometry viewer instead.)	various
pdb to geom	moved to the Unsupported module library	data
set view	no longer resets center of rotation, no longer normalizes object	data

CHAPTER 10

AVS ANIMATION APPLICATION

Known Problems Fixed

Between AVS 4 and AVS 5, the **AVS Animator** module has been made more robust. Numerous known problems have been fixed such as:

- You can now animate cameras.
- Deleting and modifying keyframes now works all of the time. Modifications do not generate incorrect interpolation values.
- The **AVS Animator** no longer occasionally dies when modifying a keyframe, moving to a new keyframe, or reading in saved animation scripts.
- The Full Animator's Key Editors now always work, including moving a key. Parameters are now correctly classified.
- Backwards playback with Key Advance now works.
- Problems related to rotations, such as incorrect interpolation values, Z values in the Key Editor being incorrect, and saved scripts failing to restore the rotation values, have all been fixed.

The net effect of these changes is to make the **AVS Animator** a much more solid and predictable tool for generating animations.

Animator Interface Changes

There are two changes to the **AVS Animator**'s interface.

Green/Red Keyframe Change/Do Not Change Indicator

The current action icon at the top of the Animator's current status indicator now changes color between red and green.

green
The indicator is almost always green. When green, the user can add, delete, or modify a keyframe.

red
The icon changes briefly to red after a keyframe is added or modified. When red, the user *must not* interact with the system. During this time,

the **AVS Animator** is updating its state information from the other modules such as the **geometry viewer**. Changing keyframe values during this time leads to unpredictable animations.

This "do not modify" time is usually very short. It only becomes noticeable when the animation is very large, such as a complex **geometry viewer** scene.

Setting Keyframe Times in the Full Animator

To set a new keyframe in the Full Animator, you should:

1. Arrange the scene to the desired configuration.
2. Type the New Keyframe Time into the widget.
3. Press **Set Keyframe**.

Formerly, steps 1 and 2 had to be reversed. If you followed the procedure in the above order, then all parameter settings and Geometry Viewer manipulations accomplished in step 1 were lost in step 2 as the **AVS Animator** reset the scene to match the interpolated values that would already be present at the New Keyframe Time. In short, it interpreted it as a "go to time *n*".

New Modules

AVS 5 contains several new modules that facilitate animation. These modules are part of base AVS, not the AVS Animation Application. See their respective man pages in the *AVS Module Reference* manual for details and sample networks.

minmax and ucd minmax

Many modules, such as **isosurface**, use the minimum / maximum data values in a field or UCD structure to dynamically set the minimum / maximum bounds on their parameter dials. This behavior can interfere with animations of time-series data. Each new dataset in the time-series may have different minimum / maximum bounds. You can find yourself carefully setting up an animation of, for example, a particular **isosurface** value, only to discover when you start reading the time-series data that **isosurface** is resetting the parameter dial that you want to remain fixed.

Two new modules, **minmax** and **ucd minmax**, solve this problem. They allow you to set fixed minimum and maximum field / UCD values that are wide enough around the actual range of your data across the time-series so that no downstream module will reset its dials.

time sampler and blend colormaps

The new **time sampler** module extracts two sequential 3D fields from a 4D time-series field and interpolates between their computational data values by a choice of linear or cubic interpolation.

The new **blend colormaps** module interpolates linearly between two color-maps in HSVA space.

CHAPTER 11

DOCUMENTATION CLARIFICATIONS/ CORRECTIONS

Existing Documentation: Clarifications and Corrections

Except for the *AVS Module Reference* manual, AVS documentation is not being updated and reissued for AVS 5. This section contains clarifications, amplifications, and corrections to the existing documentation set.

Multiple Connections to an Input Port: AVSinput_changed

Some AVS modules, such as the **image viewer** and the **geometry viewer**, allow multiple connections to an input port. This is how, for example, you are able to feed multiple images or objects—each from a different module—into one **image** or **geometry viewer** scene window. Programmers have noticed a reference to this capability in the second "connection number" parameter of the **AVSinput_changed** call: **AVSinput_changed**(*port_name*, *i*) where *i* is the number of the connection.

The multiple connections to a single input port feature is available only to builtin modules: modules that are supplied with AVS as part of the main AVS binary—the AVS kernel. User modules cannot make multiple connections to a single input port This is a fundamental limitation of the kernel's Flow Executive.

Thus, the documentation for **AVSinput_changed** is correct as written for user modules: the second connection number parameter must always be 0 for C modules and 1 for FORTRAN.

Loading Modules with the CLI module Command: the -ex Option

The CLI module command creates or modifies an instance of a module in the Network Editor's Workspace. It is used as part of CLI script files and network files. Programmers debugging modules and people using remote modules may find that they are not getting the version of a module that they expected.

The module command has a -ex <module-path> option whose documented function is to load the executable from the specified binary file path. In fact, this is not what will always happen. The -ex option is often ignored.

Assuming a module command like the following:

```
module usermodule.user.0 -ex /userdir/modules/usermodule
```

This is the procedure that AVS follows to load the module into the Workspace:

1. If there is a module with that name ("usermodule") in the current module library, this module will be loaded. The -ex option is ignored.
2. If there isn't a module with that name in the current module library, then AVS looks in all of the other module libraries that are loaded. These appear in the module library menu bar at the top of the Network Editor's main control panel. AVS examines the libraries in the reverse order that they were loaded.

   When AVS encounters a module with the same name, it examines that module's binary path (as shown in the Path field of the Module Editor panel). If the Path field matches the -ex option on the CLI module command, then that binary is used. If not, the search continues.

   Note that if the module is linked together with other modules in a single binary, and any of the modules in the binary is already in the Workspace, then the binary's in-core image is used; the binary is not re-read from disk.
3. If the search in step 2 fails, then the first module encountered in any of the other module libraries whose name matches is loaded (Path doesn't have to match).
4. If no module matches have been made, then the module binary is read in using the -ex pathname.
5. If there is no -ex pathname, or there is no module match in the libraries, then an error message is generated.

Given these rules, here are some ways to get exactly the module binary that you want. Either:

- Give the module a name that you know is unique and will not be present in any loaded module library (force case 4 above).

Or:

- Make sure that the module in question is in the current module library (force case 1) by reading it into the current module library with the mod_read CLI command before you instantiate it:

```
mod_read /userdir/modules/usermodule
module usermodule.user.0 -ex /userdir/modules/usermodule
```

There are other techniques you could employ in different situations. For example, you can start AVS with the **-modules** command line option, which will load the binary into the default module library (last-loaded). This is equivalent to using the Network Editor's **Read Module** button. You can set the current module library with the mod_lib -select <libname>**.**

Unstructured Cell Data

ASCII UCD File Format: Specifying Cell Types

Though mentioned in passing in the "Unstructured Cell Data" appendix to the *AVS Developer's Guide*, the documentation does not make it clear what strings you need to use to specify the different UCD cell types in an ASCII UCD file (pp. E-9 to E-13).

The syntax of the cell specification line in an ASCII UCD file is:

<cell_id*n*> <material_id> <cell_type> <cell_vertex1> ... <cell_vertex*n*>

Replace <cell_type> with one of the strings in Table 11-1.

Table 11-1 Strings to Specify UCD Cell Types in ASCII UCD Files

Cell Type	Use String...
UCD_POINT	pt
UCD_LINE	line
UCD_TRIANGLE	tri
UCD_QUADRILATERAL	quad
UCD_TETRAHEDRON	tet
UCD_PYRAMID	pyr
UCD_PRISM	prism
UCD_HEXAHEDRON	hex

For example, a tetrahedral cell could be specified as follows:

```
1  1  tet  1  2  3  4
```

Cell Topology

It is sometimes necessary to know the "cell topology" of a cell. Although the numbering of nodes in the various cell types is documented, the order in which those nodes are combined to make up the face of a UCD cell (face connectivity) is not documented.

Here is the original cell types figure from page E-6 of the *AVS Developer's Guide*.

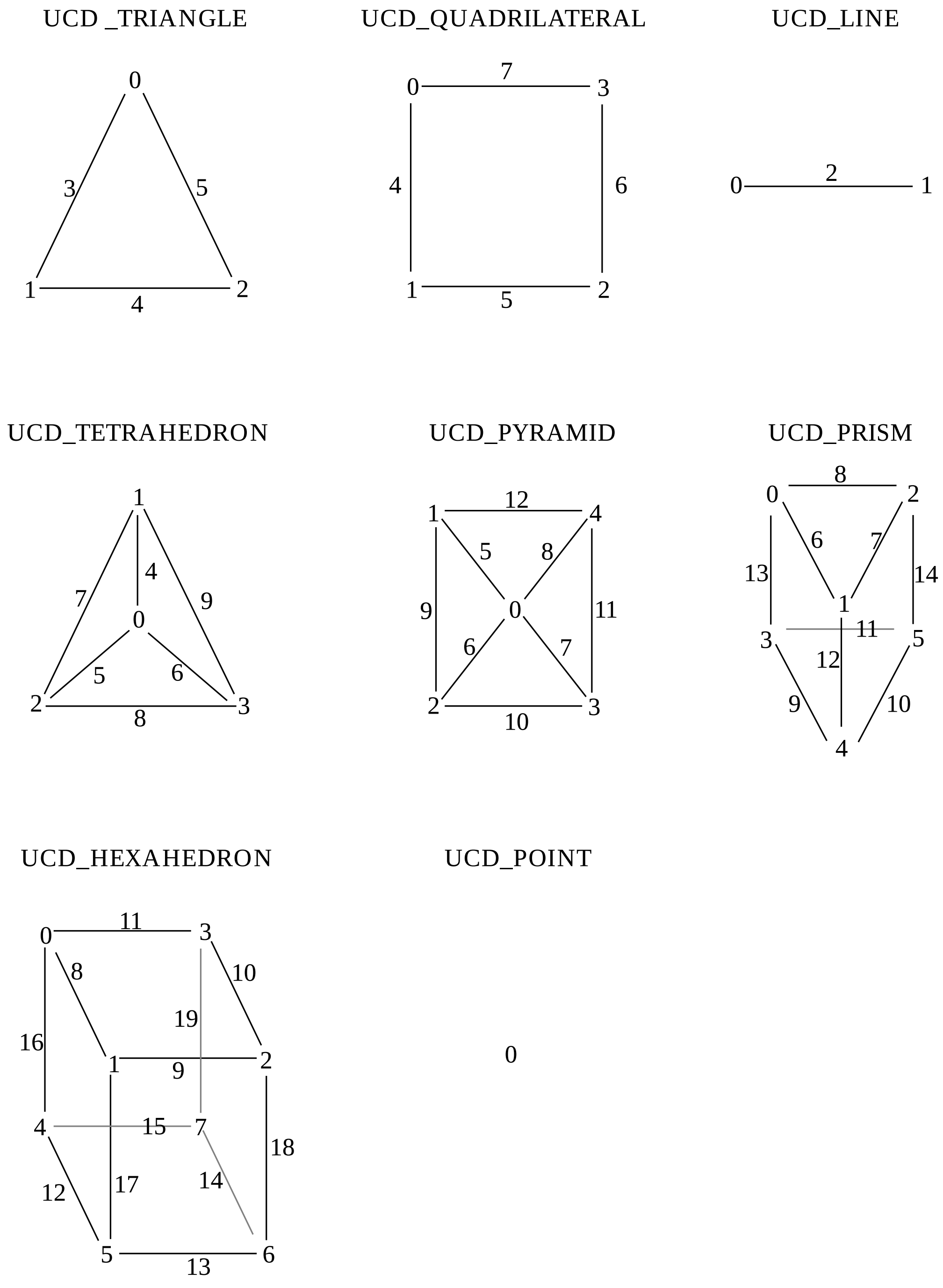

Figure 11-1 UCD Cell Types, Nodes, Mid-Edge Nodes, and Node Numbering

The cell topologies are show in Table 11-2.

Table 11-2 3D Cell Topologies

Cell Type	Cell Topology/Face Connectivity
UCD_TETRAHEDRON	1 2 0
	2 3 0
	3 1 0
	1 3 2
UCD_PYRAMID	0 1 2
	0 2 3
	0 3 4
	4 1 0
	1 4 3 2
UCD_PRISM	5 4 3
	0 1 2
	1 4 5 2
	1 0 3 4
	0 2 5 3
UCD_HEXAHEDRON	0 1 2 3
	1 5 6 2
	3 2 6 7
	0 3 7 4
	0 4 5 1
	4 7 6 5

For mid-edge nodes, use the connectivities in Table 11-2, but include each mid-edge node that you "pass over" going from node to node. For example, the face connectivity of a UCD_TETRAHEDRON with mid-edge nodes is: 1 7 2 5 0 4; 2 8 3 6 0 5; 3 9 1 4 0 6; 1 9 3 8 2 7.

The include file *avs/include/ucd_topo.h* contains a series of data arrays (*tet_topo[], pyr_topo[],* etc.) that use this connectivity. (The first number on each line is the number of nodes on that face.) This is what some AVS modules that manipulate cell faces, such as **ucd to geom**, use. You can use any data structure that suits your application.

FORTRAN Array Indexing and AVS Library Routines

All AVS library routines are written in C. FORTRAN bindings to these routines are provided so that FORTRAN modules can access AVS library functionality. However, there are places where the fundamental differences between the two languages become apparent. One such issue is how to reference the elements of parameters that are arrays.

FORTRAN uses 1-based indexing for array variables—a five element array is numbered from 1 through 5. C uses 0-based indexing—a five element array is numbered from 0 through 4.

AVS is not consistent in its treatment of array indexing.

The FORTRAN bindings for the following routines use 1-based indexing:

AVSinput_changed
AVSload_byte
AVSstore_byte

All other routines use 0-based indexing.

For example, when **AVSfield_get_label** is called from FORTRAN to acccess a label from a set of four vector element labels, the labels are numbered 0, 1, 2, and 3. This is true even though the normal FORTRAN convention would be to number them 1, 2, 3, and 4.

Error Message

AVS 5 sends a more descriptive error message when mis-indexing has occurred. In AVS 4, using FORTRAN **AVSfield_get_label** to reference the fourth label element with the 1-based index value of 4 instead of the correct 0-based index value of 3 resulted in this message:

```
AVSfield_get_label:  there are not 4 labels
```

In AVS 5, the same mistake produces this error message:

```
AVSfield_get_label:  there is no label 4
```

FORTRAN and AVSptr_alloc: Use AVStypesize for Portability

The **AVSptr_alloc** and **AVSptr_offset** routines (*AVS Developer's Guide*, p. A-72) provide one mechanism to allocate and reference data in FORTRAN. The definition of **AVSptr_alloc** is:

INTEGER AVSPTR_ALLOC(*NAME, NELEM, ELSIZE, CLEAN,*
        BASEVEC, ADDR, OFFSET)
    INTEGER *NELEM, ELSIZE, CLEAN, OFFSET, ADDR*
    DIMENSION *BASEVEC(1)*
    CHARACTER*(*) *NAME*

AVSprt_offset's parameters are similar.

The third parameter, *elsize*, defines the size in bytes of each element of the data array. If you specify a constant here, such as 4 to represent INTEGER*4, your code may not be portable across different hardware types. Instead, use the **AVStypesize** function:

AVStypesize(AVS_TYPE_BYTE)
AVStypesize(AVS_TYPE_INTEGER)
AVStypesize(AVS_TYPE_REAL)
AVStypesize(AVS_TYPE_DOUBLE)
AVStypesize(AVS_TYPE_SHORT)

For example:

```
 irselt = AVSptr_alloc( 'output field', condim, AVStypesize(AVS_TYPE_INTEGER),
*                        0, dimso, dimspo, doffset)
```

Geometry Library

GEOMcreate_label_flags: font_number

The documentation for **GEOMcreate_label_flags** call (*AVS Developer's Guide*, p. G-21) says that the *font_number* parameter is an integer from 0 through 21. This is incorrect. *font_number* is an integer from 0 to 20, specifying one of 21 different fonts.

GEOMcreate_mesh

The description of the **GEOMcreate_mesh** call on page G-22 of the *AVS Developer's Guide* reads:

> "...creates a mesh from a 2D array of vertices. The dimensions of the array are specified by the *m* and *n* parameters. The first ***n*** vertices constitute the first row of the mesh. There are ***m*** rows of vertices."

This is incorrect, the *m* and *n* are reversed in the third and fourth sentences. The description should read:

> "...creates a mesh from a 2D array of vertices. The dimensions of the array are specified by the *m* and *n* parameters. The first ***m*** vertices constitute the first row of the mesh, of which there are ***n*** rows."

Screen Space: +1.0 and -1.0 Not Always Visible

When placing objects (including titles) in screen space, which extends from -1.0 to +1.0 in X, Y, and Z, objects should not be located at -1.0 or +1.0 in Z. Some hardware renderers do not render objects at -1.0 or +1.0 in Z. Use -.99 or +.99 instead.

graph_set_line_style CLI Command

The documentation for the graph_set_line_style CLI command (*AVS Developer's Guide*, p. 5-41) lists the plot line style parameters as numbered from 0 to 3. The is incorrect. The correct line style parameters are:

1	Solid
2	Dash
3	Dot
4	Dot-Dash