AVS 5.5 ON UNIX PLATFORMS

CHAPTER THREE

Introduction

This chapter discusses information common to all platforms - new features and documentation; implementation information regarding the X Window environment and OpenGL graphics libraries; programming tips and topics; and defects fixed for this release.

AVS 5.5 NOTE: Several new features have been added (Geometry Camera module, new examples, Event masking) and the Animator source code is included; OpenGL stereo support has been broadened to include higher end SGI workstations, Compaq Tru64 Alpha, Solaris, and HP Workstations (IBM and Linux have not been tested); the AVS 5 documentation set is now online on the CD in PDF format; new documentation has been added on a number of topics; and a significant number of bugs have been fixed affecting all platforms. See the appropriate sections below.

NOTE: This document only covers platforms and operating systems that are currently supported for AVS 5.5. If you require versions of AVS 5 to run on older operating systems, please contact AVS Customer Support or your local distributor for appropriate media and release notes.

Following this chapter are platform specific chapters which document the differences, special considerations and known problems unique to that platform. Each chapter provides information on the following:

- hardware prerequisites (graphics hardware, memory, disk space and swap space)
- software prerequisites (operating system, graphics software, and other topics)
- general usage information (how AVS subsystems work)
- programming considerations for C, C++, and FORTRAN module writing.

New Features in AVS 5.5

AVS 5.5 includes the following new features:

- Geometry Camera module provides a better secondary camera to support the Geometry Viewer module
- Event Masking provides a means of masking out undesirable mouse or function key input to the Geometry Viewer
- Additional example modules provide a sample FORTRAN coroutine, a new test field data generator, and a GEOM data output/viewer module

Geometry Camera

A new builtin "Geometry Camera" module has been added to provide improved support for secondary cameras associated with a Geometry Viewer module. This wraps a camera window within a user interface like the Geometry Viewer module's main camera window, but does not provide image output and other Geometry Viewer features.

The Geometry Viewer module produces a main camera ("view") with the following characteristics:

- a title bar editable by the Layout Editor and recorded in networks
- an upper-left-dimple activated menu including zoom, unzoom, etc.
- resistance to being closed using ordinary window manager commands
- position and layout recorded in networks and scripts

In the Geometry Viewer (as opposed to the module with the same name), adding a new camera (Camera/New Camera) produces a camera without these characteristics.

The new Geometry Camera module takes as input a specific Geometry Viewer module output ("Trigger" - colored pink and now visible) to associate itself with that module and adds a new camera to this main view with all the characteristics noted above. A Geometry Camera can be detached from one viewer and reattached to another, restored from a network file properly, edited by the Layout Editor, and destroyed without adverse effects.

Demonstration:

- Start the Network Editor, then on "AVS Network Editor" panel, hit Help, then hit Demos. Close Help window.
- Select "..(demo)" at the top of the menu to go up one directory then go down into the "geom_viewer" directory.
- Run geom_camera script to create multiple Geometry Viewers with geom_camera modules attached to them.

Function Key, Arrow Key, Mouse Event Masking

A new feature has been added to "mask out" events coming from specific function keys, arrow keys or mouse events. This permits customers to prevent their end users from inadvertently performing various Geometry Viewer transformations linked to these keys. The developer encodes the key values in a "mask" in which each bit represents a function key, arrow key or mouse click. The combined mask is then set in the *.avsrc* file.

Function Key mask (KeyMask):

- Bits, 1-12: Function Key 1-12

Arrow Key mask (ArrowMask):

- Bits 1-4: Left/Right/Up/Down
- Bits 5-8: Shift Left/Right/Up/Down

Mouse key values: (MouseMask)

- Bits 1,2,3: Left/Middle/Right
- Bits 4,5,6: Control - Left/Middle/Right
- Bits 7,8,9: Shift - Left/Middle/Right

Sample *.avsrc* file: (Values may be in hex, decimal, octal, etc)

```
KeyMask         0x3   (Bits 1, 2 - masks out Function Key 1, 2)
ArrowMask       0x1   (Bits 1 - masks out Arrow key 1 (left)
DebugMask       0x1   (Bits 1 - prints out debug messages)
MouseMask       0x48  (Bits 4, 7 - mask out Control-Left and Shift-Left)
```

Demonstration:

- Create a *.avsrc* file in your home directory or the current directory with the sample shown above. See the *AVS User's Guide*, page 5-12 "Transforming Objects" which details what mouse key strokes should do.

- The DebugMask will show debug messages like "Mask Values:" and "Masking out xxx" as appropriate events occur and are masked out.

New Example modules

Several new example modules have been added in AVS 5.5.

- Corout f.f - a second FORTRAN coroutine example showing how basic data types are passed in and out

- Test field.c - a more advanced field data pattern generator for testing

- Write geom.c - a diagnostic output module for the GEOM data type

Corout f.f

This module is a more complete example of a FORTRAN coroutine that shows all basic data types being passed in as inputs and parameters and then copied out as outputs.

A test script is provided to show all inputs and outputs in use - *$AVS_PATH/demo/examples/corout_f.scr*

Test field

This module has been used extensively in internal testing at AVS and is being provided to help users produce a wider range of diagnostic test field data for testing their own modules or providing simple test cases back to AVS Customer Support to demonstrate problems they are having with supported modules. It also provides a more extensive example of a module that is actively managing and reconfiguring its user interface using CLI commands and the AVScommand function. **NOTE:** Test field's name is close to that of two earlier and simpler test field examples, used to demonstrate C and FORTRAN field creation.

Documentation is provided as an online module man page under *$AVS_PATH/runtime/help/modules/test_fld.txt* which should appear when you select module documentation for the module.

A test script is provided to show several different sample outputs - *$AVS_PATH/demo/examples/test_fld.scr.*

Write Geom

This module provides a variant on the "print field" diagnostic module that provides both binary and text output from GEOM data. It is useful to help diagnose GEOM data issues by displaying the data in a more readable form; often issues relate to extreme data values and invalid values such as "NaNs" (not-a-number). It also provides a simple example of how modules such as "print field" are able to display text files in a text window as part of an AVS 5 network.

Write Geom writes GEOM format data to a binary output file then uses an external filter program (*$AVS_PATH/bin/geom_to_text*) to convert the binary file to a text file. It then uses an AVS text browser widget to view the text file within the AVS user interface. The binary file is readable by "read geom"; the text file can be read using any text editor.

The binary file name must be set before any output is generated. Once this is set, the binary file is written out and the text file automatically produced and read into the browser. A very large GEOM file might generate enough data to cause the module to crash when read in. Unsetting the text file name in the module will prevent it being generated or read in; you can use *geom_to_text* externally as needed.

A test script is provided to show this module in use - *$AVS_PATH/demo/examples/write_geom.scr.*

AVS/Animator source code

The AVS/Animator module source code is now provided with the example modules, both to provide a more extensive example module and to offer users the opportunity to extend the Animator to provide alternate output formats not currently supported. For more information in general, see the associated end user documentation in the *Animating AVS Data Visualizations* manual included in the main doc set and in the online PDF files provided with AVS 5.5.

The source code is located under *$AVS_PATH/examples/animator* and consists of a number of subdirectories containing the Animator module, supporting libraries, and the associated modules that make up the Animator package. It consists of the following subdirectories:

- *anim_lib:* Animation operations needed by the Animator module.
- *asf:* Applications Support Functions library which is responsible for the parsing of the menu file and the generation of the user interface menus.
- *avs_library:* Empty directory that the modules will be installed into

- *bfa:* The Animator module source code, starting with *bfa_main.c* which defines the Animator user interface and module compute operations.

- *lib:* Empty directory that *anim_lib* and *asf* will copy libraries to; the Makefile will add links to standard AVS libraries here also.

- *modules/animator:* Source code for supporting modules such as **output ImageNode, output VideoCreator, prepare video, read frame seq** and **write frame seq.**

In order to make the animator from source code, do the following:

```
cd $AVS_PATH/examples/animator
make
```

This will create a few links from the animator source to the standard *$AVS_PATH* installation areas, then visit the subdirectories to make the supporting libraries and modules.

Documentation updates

AVS 5 Documentation set in PDF format

The AVS 5 documentation is now available online in Adobe PDF (Portable Document Format) files providing the core set of books in a viewable, searchable, and printable format. The set can be installed from the **AVS5_DOC** product archive or read directly from the CD (*/cdrom/avs5_doc* on most platforms). Advanced Visual Systems grants users permission to print and make copies of part or all of the documentation set for internal use at their site. **Note:** A few books were not available in PDF format at release time (*AVSGraph User's Guide, Technical Overview,* and *UCD Builder's Guide*); contact AVS Customer Support for possible future availability of these book files from our web site.

Installation

The AVS 5 Documentation package can be installed onto your hard disk if you use *install.avs* and select the **AVS5_DOC** package to install in *$AVS_PATH/avs5_doc* or a directory of your choice. The same files are available for direct access on the CD in the *avs5_doc* directory at the top level; the full pathname may be */cdrom/avs5_doc* or */CDROM/AVS5_DOC* depending on your platform's CD conventions.

You will need a copy of the Adobe Acrobat Reader in order to read or print the PDF files. If you don't have a copy you can download the software at no cost from the Adobe web site at *http://www.adobe.com/prodindex/acrobat/readstep.html*. Most UNIX platforms are supported as well as Windows 95 and NT.

Getting Started

Once you have obtained an Acrobat reader and have installed the package or mounted the CD for direct access, open up the introductory file, *$AVS_PATH/avs5_doc/intro.pdf*, which provides a general overview of the organization and evolution of the AVS 5 documentation set, information on where to get started, and direct links into the individual manual PDF files stored under the *books* subdirectory. Live links are highlighted with a red border and will take you directly to the appropriate section or manual file as appropriate; the Table of Contents of each manual also contains live links into the chapters and subsections of each manual for easier access.

If you should have any troubles using the external links in the *intro.pdf* file, you can directly access the individual manuals in the *books* subdirectory. **Note:** If you are printing out sections of the PDF files, you must set the Postscript level to "Level 1" in the print dialog; selecting "Level 2" will cause problems with some of the images.

New Chapter: Debugging in AVS 5

A new chapter has been added to this release note document that provides an overall summary of how to approach debugging modules in AVS 5.5. This chapter draws together information and techniques from throughout the AVS 5 product and is targeted towards both new and experienced module writers.

AVSfield_points_array_size

The AVSfield_points_array_size routine was not previously documented but has been available since AVS 5.0.

C:
AVSfield_points_array_size (*uniform, ndim, nspace, dimensions* **)**
  int *uniform, ndim, nspace, *dimensions;*

FORTRAN:
AVSFIELD_POINTS_ARRAY_SIZE (*UNIFORM, NDIM, NSPACE, DIMENSIONS* **)**
  INTEGER *UNIFORM, NDIM, NSPACE*
  INTEGER *DIMENSIONS(ndim)*

This routine calculates the number of items in a field points array depending on the type of field it is.

```
Inputs: uniform         - One of UNIFORM, RECTILINEAR, IRRREGULAR,
                          or UNIFORM_DONTCARE.
        ndim            - Number of dimensions in computational space.
        nspace          - Number of dimensions in model space.
```

```
dimensions      - Array of ndim values.

Returns: Number of floating point values required for the points array.
```

AVSoutput_string

The AVSoutput_string function is a utility to help FORTRAN modules allocate strings for use as function outputs. It has existed since at least AVS 5.3 but has not been formally documented.

FORTRAN:
INTEGER AVSOUTPUT_STRING (*OUTPTR, STRING* **)**
INTEGER *OUTPTR*
CHARACTER*(*) *STRING*

This routine converts a FORTRAN string value to a newly allocated copy that can be sent to output. It will automatically allocate/reallocate storage space as needed.

```
Inputs: outptr - pointer to string copy passed into the compute function
                 to hold the output string.
        string - FORTRAN string buffer to be copied
Returns: 0 if failed, 1 for success
```

AVSpath variable

AVS 5 modules can find out the path to the home AVS 5 directory by using the AVSpath variable. This is now included in the *$AVS_PATH/include/avs.h* header file but in releases prior to AVS 5.5 needs to be directly declared as shown in this example from the AVS-Graph module which is using AVSpath while looking for the *$AVS_PATH/runtime/AVSGraph directory:*

```
extern char *AVSpath;

if (AVSpath != NULL)
    putenv(strcat(strcat(tmpstr,AVSpath),"/runtime/AVSGraph"));
else
    putenv("UNIDIR=/usr/avs/runtime/AVSGraph");
```

Choice value output example

Some customers have had trouble in successfully sending a string value from one module to act as a choice value input or parameter value in another module downstream. The following is an example of of a generic choice output module description and compute function. The resulting string will be typed as a choice and passed to any module needing a choice parameter downstream. In this case it is up to the user to know valid choices that will be acceptable downstream.

```
MODchoice()
{
   int choice_compute(), param;

   AVSset_module_name("choice", MODULE_DATA);
   AVSset_module_flags( COOPERATIVE | REENTRANT );
   AVScreate_output_port("choice_value", "choice");
   param = AVSadd_parameter("choice_string", "string", 0, 0, "");

   AVSset_compute_proc(choice_compute);
}

static choice_compute(outvalue1,invalue)
char  *invalue, **outvalue1;
{
   if (!invalue)
     return(1);

   *outvalue1 = strdup(invalue);
   return(1);
}
```

Read Field module accepts short data

The current Read Field documentation does not mention it, but the module does accept short data and has done so since AVS 5.0 on all platforms.

Geometry Viewer object parameter

The man page does not provide information on this parameter. It is used to see what the current object is (used in the Data Viewer module for example). You can NOT change the current object by setting this, as it is for informational use only.

Unexposed command line options

A number of existing AVS command line options have previously not been exposed or documented but were used for internal testing purposes. Because some of these may be useful to customers they have been documented in the usage command line option and are described below:

- -mod_host <host>: default host to run modules on. This will result in nearly all supported modules being run as remote modules on the given machine. The machine must be present in the *.hosts* file.

- -mod_time: print out time spent in each module executed. This option can be useful for analysing where execution time is being spent.

- -no_display: run without visible windows. AVS always requires access to an open X display in order to allocate resources for the Geometry and Image Viewers. This option will prevent virtually all windows from being mapped to the display, running invisibly without disturbing existing users of the workstation.

- -swrender: Select software renderer as default instead of the hardware renderer. The hardware renderer may still be selected, unlike *-nohw*.

- -test_path: Specify testing directory, exposed to test networks and scripts as the CLI variable $TestPath; used as a pathname reference during internal testing.

- -vistype: (New in AVS 5.5) Specify the visual type in the same way that the "VisualType" avsrc option works. Recognizes the first letter of the visual types for convenience and ignores the rest of the token. Recognized values include D(irectColor), P(seudoColor), T(rueColor), and V(isualID) <vid>. For example, the following would set avs to use VisualID 0x31 (use *xdpyinfo* to see a list of your available VisualIDs).

```
avs -vistype v 0x31
```

New Arbitrary Slice parameter Slice Size

The arbitrary slice module selects a plane size based on the XY extent of the object being viewed. Depending on the object, this plane may not be large enough as it is rotated around within the overall dimensions of the object - for example, an oblong brick object with the XY cutting across the small dimension of the brick would use a small square slice plane that won't extend across the length of the brick.

This has been fixed by adding a new choice parameter to the module called "Slice Size" with the following choices:

- XY slice: use the XY slice as the plane size

- YZ slice: use the YZ slice as the plane size

- ZX slice: use the ZX slice as the plane size

- Diagonal slice: use the diagonal of the extents (across opposite corners) to determine the size of the plane.

X Implementation

AVS runs as an X Window System client. It uses X protocol requests (via Xlib) to the X server to produce its user interface. The interface includes:

- the main AVS menu
- the Image, Geometry, and Graph Viewer control panels
- the Network Editor
- all control widgets such as file browsers, pop-up menus, dialog, message, and help panels
- the Image viewer's viewport windows, and the **display image** module's output window
- the Graph Viewer's plot windows

The images that appear in Geometry Viewer scenes windows can be produced by two different rendering mechanisms:

- A **software renderer** that implements a set of graphics primitives (polygons, lighting model, perspective, shading, color, etc.) in software, rendering the resulting image into an X Window System image. The image is transferred to the X server using the *xlib* XPutImage call.
- A **hardware renderer** that uses the hardware graphics library to create renderings that are displayed on the screen using the platforms's native hardware graphics subsystem. Depending on the platform this library will be GL, OpenGL, XGL, or PHIGS.

You can switch between hardware and software renderers while AVS is running using the rendering switches provided on the Geometry Viewer **Cameras** submenu. You can control which renderer will be the default active renderer when AVS first starts using the **-renderer** command line option or **Renderer** keyword in your personal *.avsrc* file. When multiple hardware renderers are available, you need to start different *avs* kernels to access them (*avs, avs.gl, avs.phigs*, etc.).

Z Buffer Required for Hardware Rendering: On IBM and SGI platforms, the hardware renderer requires the presence of a Z-buffer in order to function. A Z-buffer should be present on all SGI platforms except the Personal IRIS, where it is an option. On the IRIS Indigo, the hardware Z-buffer and GL graphics functions are emulated in software. On IBM or SGI systems without a Z-buffer, you must use the software renderer.

Both hardware and software renderers will function on 8-plane pseudocolor or 24-plane true color frame buffers. However, you will need to establish the correct X color visual, since the default X visual used by the X server when it starts is not always the most powerful that the system is capable of supporting. This is discussed in the "Starting AVS" section of the appropriate platform specific chapter.

All screen output that AVS performs on the display via X Window System calls is bounded by the capabilities of the X server implementation on the workstation.

All rendering AVS does via hardware graphics library calls is bounded by the capabilities of the particular library and its underlying hardware graphics adapter.

The software renderer implements its own 3D graphics model, and is bounded only by what graphics primitives it does or does not implement in software. However, the final color rendition on the screen is constrained by the number of color planes that the X server creating the window can support: 8 bits or 24 bits.

See the table and notes in the "Geometry Viewer" section in each platform specific chapter for further details on rendering capabilities.

Graphics Libraries: OpenGL

AVS 5.5 expands the use of OpenGL as a common graphics interface for 3D hardware rendering, providing OpenGL as the default renderer for all supported platforms, now including HP-UX 10.20.

OpenGL provides a more standard cross platform graphics layer for AVS 5.5 and future releases. It enhances remote display graphics support and centralizes AVS 5 cross platform graphics support, permitting better use of shared extensions in new releases.

Common features

A number of features are common to OpenGL renderers regardless of platform. On UNIX platforms (X Windows), GLX is the OpenGL extension. It embodies a programming interface and a protocol for client-server communication. GLX provides a mechanism for OpenGL to allocate and control display resources on the X server. It binds an OpenGL rendering context to a specific X visual that is selected from the list of visuals supported by the X server.

AVS 5.5 on UNIX Platforms

There are certain restrictions on the visuals that can be used with OpenGL. Only displays that use an RGB pixel format with a Z-buffer support the OpenGL renderer. To double buffer the view window under GLX, OpenGL must be supported in a double-buffered TrueColor or DirectColor X visual with an associated Z-buffer. The GLX specification requires that at least one such visual is supported.

Several UNIX tools enumerate the available X visuals and their support for OpenGL via the GLX extension. A standard X Window program, called *xdpyinfo*, lists X server parameters and all suported visuals with their X Window characteristics. A GLX inquiry utility, *xglinfo*, is usually supplied with OpenGL examples. *xglinfo* lists general GLX parameters as well as each X visual and its support for OpenGL. OpenGL installations can repeat X visuals, which differ only in the configuration of additional (non-X) buffers used by OpenGL (for example, alpha, depth, stencil). RGB visual formats can be 8-bit (3-3-2), 12 bit (4-4-4), or 24-bit (8-8-8). The OpenGL renderer does not use blending modes that require the framebuffer to store alpha values. **Note:** If you do not have *xglinfo* available on your SGI system, you can download it from *ftp://sgigate.sgi.com/pub/opengl/contrib/xglinfo.tar.Z*.

OpenGL Renderer Information

When AVS instances a view, a set of inquiries is performed when the OpenGL renderer is first instanced. The values returned from the inquiries contain information about the platform that is used as the OpenGL server (perhaps remotely), configuration of the OpenGL implementation, supported extensions, and characteristics of the view window. This information is used to configure the behavior of the renderer, and it is written to the command terminal if the environment variable AVS_OGL_INFO is set before running AVS.

```
setenv AVS_OGL_INFO 1
```

The following information is displayed when AVS_OGL_INFO is set:

- Confirmation that GLX is running on the server and the GLX version number.

- The OpenGL version number, vendor name, hardware name, and a list of supported extensions.

- OpenGL parameters are displayed. These are ARGB color buffer depths, Z-buffer depth, maximum number of lights, and maximum 2D texture image dimensions.

- The type of connection to the X server (Direct or Indirect), X visual id, and X visual class (TrueColor or DirectColor). The connection tells whether OpenGL and GLX have direct access to the display when client and server are on the same machine. If the type is Indirect, the client and server are remote or are they are on the same server but display requests pass through the X server. In general, direct connections are faster than indirect connections when the display is local.

Performance

OpenGL performance will vary widely depending on the level of support for the graphics acceleration hardware your system is using. Most newer graphics adapters will be well supported by OpenGL; older adapters will usually be supported but may run slower than in previous AVS 5 releases. Certain vendor specific graphics features may not be as well supported in OpenGL as in prior graphics libraries.

For this reason, AVS 5.5 provides alternate AVS executables on different platforms to continue support for the prior hardware rendering interfaces: XGL on Solaris, GL on IBM and PHIGS on HP-UX 10.20. These executables are provided in the *$AVS_PATH/bin* directory as *avs.xgl*, *avs.gl*, or *avs.phigs* respectively. It is hoped that for most users, the default OpenGL renderer will be most appropriate. When that is not the case, the user may wish to make the alternate renderer the default executable as follows:

```
cd $AVS_PATH/bin
mv avs avs.ogl
ln -s avs.gl avs     /* Or "avs.xgl" or "avs.phigs" as appropriate */
```

The renderer is always listed at the end of the version line to help minimize confusion for the user and AVS Customer Support when the need arises. For example

```
avs -version
```

might show "AVS version: 5.5 (50.85 SunOS5 ogl)".

Optimizing Performance

When using AVS's software renderer option on lower-end platforms, you will probably want to switch on **Bounding Box** on the Geometry Viewer's control panel to reduce the amount of rendering required when you transform objects.

If you will be rendering polygonal spheres such as those produced by the **bubbleviz/scatter dots** and **particle advector** modules, or molecular ball and stick models, you should use the **Subdivision** slider at the bottom of the Geometry Viewer's **Object** menu to reduce the number of polygons used to approximate a sphere. The 1 value represented by having the slider all the way at the left will render a sphere as an 8-

sided diamond. You can always increase the value after you have arranged the scene for high-quality output.

Texture Mapping Limitation

Texture map size is limited by the local implementation of OpenGL and varies across vendors and platforms, ranging from a limit of 64 by 64 pixels to 4096 by 4096 pixels or more. OpenGL also requires that texture maps be scaled to powers of two. When you provide a texture map that exceeds your local systems limit or is not a power of two, you will see warning messages (OGL_load_texture errors). If this causes problems, select the "Filter Texture" button, then reload the texture; this will scale your texture to fit the hardware limitations and smooth out the result.

OpenGL Errors

The following explains some of the more common errors you may encounter when running OpenGL.

Initialization Errors (GLX).

- *no GLX extension on this X server* OR *cannot get version of GLX:* The GLX extension that supports OpenGL is not available or was not configured in the X server when it started. On some platforms, you must start the X server with command line options to get OpenGL and double buffering. See the system prerequisites chapter for your platform.

- *cannot find OpenGL visual:* GLX and OpenGL are supported, but do not provide a satisfactory visual for use by AVS. The specification of GLX ensures that at least one single buffered visual is available. Try to re-instance the OpenGL renderer for a view in single-buffered mode.

- *glXCreateContext failed* OR *cannot create colormap:* GLX and OpenGL are available, and a satisfactory visual was found on the server. However, other resources were not found and the server cannot create an OpenGL renderer window or the associated X colormap.

General Errors.

- *no support for 3D texture:* The application tried to display an object with 3D texture on a system that does not support the OpenGL 3D texture extension. Note that 3D textures cannot be displayed from a remote client unless the client also supports the relevant extensions. If you do not have one of these high-end machines, use the Software Renderer to display 3D texture.

- *cannot create texture display list:* OR *texture dimensions (mxn) exceeds system limit (N)* The graphics display supports texture, but ran out of resources allocating texture memory from the host's main memory or from the graphics adapter. Reduce the size of the texture using filter modules (for example, downsize), or allocate more swap space, main memory, or hardware texture memory.

- *OpenGL error X in gluBuild2DMipmaps (glTexImage3DExt, glTexImage2D, gluScaleImage):* A standard OpenGL error occurred in one of these functions. The OpenGL error message should follow this error. Possibly, texture memory overflowed, in which case, reduce the size of the texture using filter modules (for example, downsize) or allocate more swap space, main memory, or hardware texture memory.

X Server Color

On 24-plane true color systems, each pixel can have one of 256 red x 256 green x 256 blue (16,777,216) color values. There are 8 bits to represent red tones, 8 bits for green tones, and 8 bits for blue tones. The red, green, and blue tones combine to create the actual pixel color.

On 8-plane pseudo color systems, each pixel can have one of 216 color values. There are 6 red tones, 6 green tones, and 6 blue tones. To display a true color image on an 8-plane pseudo color device, AVS takes the original red value for each pixel and finds the closest numeric value from among the 6 reds available. It does the same for green and blue.

AVS then takes the pixmap that is made up of these three best-matches and applies a dithering algorithm to the pixmap. Dithering uses the fact that the human eye will interpolate between dots of color, creating the impression of a color value between two actual color values. The dithering process corrects for information lost in the 256-to-6 reduction by comparing how far off each final pixel value was from the original value against a dithering matrix. Some pixel values have their red/green/blue values adjusted up or down to create a closer approximation to the original true color image. This might sound very limited, but the end result is surprisingly satisfactory.

The Image Viewer and **display image** module have an option that turns off dithering on 8-plane systems when no change to the original image appearance is desired.

Note: The IBM 24-bit High Performance Adapter only supports the 8-plane pseudo color X visual under some OS levels. See the IBM chapter below for more information.

AVS 5.5 on UNIX Platforms

Warning: reducing color usage

On some 8-plane systems, the applications already running on the workstation may have already allocated so many cells in the 256 entry colormap that there are not enough for AVS to use 6 tones for red, green, and blue. In this case, you will see a message like the following when AVS starts up:

```
Warning: reducing color usage:  R-5, G=5, B=5, Grey=17
```

indicating that AVS is reducing its colormap cell usage.

If you get this message on a truecolor system, it means you have the wrong default visual. See the "Visual Type" section later in this chapter.

Dark Display: Gamma Correction

On some workstations, the AVS interface (all non-graphics windows) may appear too dark. You can lighten the interface using the **-gamma** command line option, or the **Gamma** *.avsrc* startup file option. For example, in your personal *.avsrc* file:

```
Gamma 1.7
```

You will have to experiment to find a satisfactory value. Values between 1.7 and 2.2 are good starting points for experimentation. Higher real values produce a lighter display.

How to Check the Default Visual

You can check to see what the default visual is. Type the */usr/bin/X11/xdpyinfo* command and look for the "default screen number" line for your screen:

```
name of display:    :0.0
version number:    11.0
vendor string:    Silicon Graphics
     .
     .
     .
default screen number:    0        <--- the relevant line
number of screens:    1
```

Next, find the description information of the default screen and look for the "default visual id" line.

```
screen #0:
  dimensions:    1280x1024 pixels (340x270 millimeters)
  resolution:    96x96 dots per inch
  depths (6):    1, 2, 4, 8, 12, 24
  root window id:    0x2d
  depth of root window:    8 planes
  number of colormaps:    minimum 1, maximum 8
  default colormap:    0x2b
  default number of colormap cells:    256
     .
     .
     .
  number of visuals:    8
  default visual id:  0x20        <---- the relevant line
```

Now, look down the visual definitions until you find the hexadecimal visual id code that matches. This is the default visual that the X server and AVS will use.

```
  visual:
    visual id:    0x23
    class:    PseudoColor
    depth:    2 planes
    size of colormap:    4 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
     .
     .
     .
  visual:
    visual id:    0x20           <---- the relevant line
    class:    PseudoColor
    depth:    8 planes
    size of colormap:    256 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
     .
     .
     .
```

This is an 8-plane pseudo color visual, as specified by the "class" and "depth" lines.

You need to tell AVS to use a different visual. Continue to look down the list of visual definitions until you find one with a "class" of true color and a "depth" of 24 planes. Note its "visual id", in this case 0x29.

```
     .
     .
     .
visual:
  visual id:    0x29
  class:    TrueColor             <---- A 24-plane true color visual
```

```
depth:    24 planes
size of colormap:    256 entries
red, green, blue masks:    0xff, 0xff00, 0xff0000
significant bits in color specification:    8 bits
  .
  .
  .
```

You will see 12-plane true color visuals, and various others such as StaticColor. Do not use these visuals.

Changing the Default Visual for AVS

To start AVS using the correct visual, make the following addition to your personal *.avsrc* file:

`VisualType  VisualID` *n*

Replace *n* with the hexadecimal visual id of the correct true color visual.

```
VisualType  VisualID 0x29
```

With AVS 5.5, there is also a new command line option, *-vistype*, which provides the same functionality at runtime. See the section above on "Unexposed command line options".

Window Manager

AVS works with any of the standard X Window System window managers, including *mwm*, the Motif window manager; *dxwm*, the DECwindows window manager; *4Dwm*, the SGI IRIX window manager; HP VUE; and the Common Desktop Environment (CDE) window manager. No user modifications are necessary in order to use AVS with these window managers.

However, there are some modifications that you may want to make as a matter of personal preference. The following examples are more oriented towards Motif, but similar changes can be made to the other window managers.

Click to Type

By default Motif is a "click to type" window manager. It is not enough in all cases to simply move the mouse cursor into an input window such as an AVS file browser typein panel or the Transformation Options panel and begin to type. You may need to click on the left mouse button first to make the AVS window receive events.

To disable "click to type" in Motif, add or modify the following line to your personal *.Xdefaults* file to match what is shown here:

```
Mwm*keyboardFocusPolicy:   pointer
```

(On some systems, the "k" in keyboard may need to be an uppercase "K".) You may need to make additional changes to ensure that AVS gets all button events. Consult your Motif documentation.

In the HP VUE window manager, to disable "click to type" behavior, follow this selection sequence:

```
Style Manager
  Window
     FocusFollowsMouse
         OK
```

For CDE window managers, the sequence is this:

```
Style Manager
  Window
     Click in Window To Make Active
         OK
```

Intercepting Mouse Events

In a manner conforming to the interaction style of the older window managers under which AVS was originally developed, AVS makes heavy use of all mouse buttons. Under Motif, the window manager may try to intercept mouse button events, particularly the left mouse button, for its own purposes such as making a window the current window and automatically raising it. If you have any lines like the following in your *.mwmrc* file:

```
<Btn1Down>        frame|icon|window       f.raise
```

You may want to change them to:

```
<Btn1Down>        frame|icon              f.raise
```

to ensure that AVS gets the mouse button events that occur within its windows.

Close Function

Pop-ups menus include a **Close** function. However, selecting this **Close** function is actually a command to kill the window, not to just iconify (minimize) or unmap its window from the screen. Using the

Close function on many AVS windows will either not work, or will put AVS into an unuseable state. You should use the various **Close** buttons on AVS's windows instead.

Window Decoration

If you do not want all AVS windows to be surrounded with the Motif window border controls, put this line in your *.Xdefaults* file:

```
Mwm*avs*clientDecoration:none
```

Layout Editor

The Layout Editor relies on some knowledge of which window manager you are using to reliably determine window positions for some operations. If you have a problem in which windows move unexpectedly when selected in the Layout Editor, use the *window_mgr* CLI command to inform AVS which window manager you are using. For more information see Chapter 5 in the *AVS Developer's Guide*.

X Terminal and Remote Display Support

In most cases, you can run AVS as a remote X client on one UNIX workstation from another workstation with a color monitor and an X server that supports at least an 8-bit PseudoColor visual. You can also run AVS on a UNIX workstation from a color X Terminal. The X Terminal's X server must also support at least an 8-bit PseudoColor visual.

avs -nohw Required

Except as noted below for specific platforms, you must start AVS using the *-nohw* command line option or the **NoHW 1** keyword in your personal *.avsrc* file. This will force the use of AVS's software renderer. Without this option, AVS will attempt to initialize the hardware renderer when you enter the Geometry Viewer the first time and will fail.

The software renderer will perform 3D rendering into an X image and send the image to the X server for display. Color rendition will be up to the capabilities of the local X server: pseudo color on X servers with a PseudoColor visual, and true color on X server that support a TrueColor visual. See the "AVS on Color X Servers" appendix in the *AVS User's Guide* for more information.

Remote hardware rendering

In some cases when you are rendering from one hardware platform to display on another using the same hardware graphics library you may not need to rely on the software renderer.

- OpenGL: The OpenGL library is capable of running as a distributed graphics subsystem, even across different platforms, as long as the remote system is configured appropriately, providing GLX support in the X server.

- HP: When you are running between HP PHIGS workstations, you do not need to specify *-nohw*. The HP VMX (Virtual Memory X) facility will emulate remote hardware rendering transparently.

- SGI: The SGI OpenGL library is capable of running as a distributed graphics subsystem. This means that, in the special case where you are running AVS on an SGI workstation from *another SGI workstation*, you will still be able to use the hardware renderer and take advantage of its superior rendering speed. AVS on the remote system will send OpenGL library instructions to the local graphics subsystem to be executed. No special steps are needed to make this happen.

- IBM: If you get the following error message when you start AVS:

```
gl: gversion:  1345-072  The requested X Windows Server extension is not available
```

  then you need to set the AVS_GRAPHICS environment variable to **xterm**.

csh users would set this environment variable as follows:

```
setenv AVS_GRAPHICS xterm
```

ksh or *sh* users would set this environment variable as follows:

```
AVS_GRAPHICS=xterm
export AVS_GRAPHICS
```

- Compaq Tru64 UNIX: The **-nohw** command line option or the **NoHW 1** *.avsrc* keyword should not be necessary as AVS automatically detects the graphics adapter type and initializes the appropriate renderer(s).

Stereo Support

Support for stereo viewing is now provided for SGI (N32 and N64), Sun SunOS5 (Solaris) (OpenGL and XGL), Compaq Tru64 Alpha, and HP Workstations (Visualize-FX4 and -FX6 graphics adapters only). In this release, stereo support on SGI will support either "full screen" or "stereo-in-a-window" depending on the capabilities of your platform. Stereo on Solaris, Compaq Tru64 Alpha, and the HP workstations also supports the "stereo-in-a-window" approach. Note: IBM and Linux have not been tested but may work.

Stereo is supported using CrystalEyes stereo goggles from Stereo-Graphics. One source for these is Qualix Direct, found at *http://www.qualixdirect.com*. You must consult the vendor's documentation for information on how to connect this device to your workstation.

Stereo Display

Stereo display is controlled from the Geometry Viewer's **Cameras** submenu.

To turn on stereo:

- Toggle the **Stereo** button on the **Cameras** submenu. You may have to scroll to make this button visible.

- Type **Ctrl-left mouse button**.

The entire screen is taken over by the stereo view of the object when using "full screen" stereo; "quad-buffered" or "stereo-in-a-window" only affects the Geometry Viewer windows. You will not see stereo unless you have the CrystalEyes stereo goggles.

To exit stereo mode, type **Ctrl-left mouse button** again or use the Camera/Stereo toggle button. **Note:** You should exit stereo mode before leaving AVS or it may leave stereo enabled. If you do leave stereo on by accident, restart AVS and toggle stereo on/off to leave stereo mode; or use the "off" stereo command as shown in the stereo documentation below to exit stereo manually.

Enabling Stereo

Each platform has a different way of enabling and disabling stereo mode. You should see the appropriate platform chapter for more information.

Most platforms enable stereo in the X server configuration. For the SGI, you can see the current settings for on and off commands using two debug environment flags:

```
setenv AVS_STEREO_CMDS_ARE 1
setenv AVS_OGL_INFO 1
```

In order to override these values, there are two environment variables to set the "on" and "off" commands:

- **AVS_STEREO_ON_CMD** specifies the command that enables stereo mode
- **AVS_STEREO_OFF_CMD** specifies the command that disables stereo mode

Because AVS does NOT "remember" your previous mode before it entered stereo, you may want to at least set the AVS_STEREO_OFF_CMD variable.

Stereo Adjustment Parameters

In some cases, you may wish to make minor adjustments in the stereo control parameters controlling apparent eye separation between the two viewpoints or the apparent distance between the eyes and the focal point. These are controlled differently depending on which platform you are using.

On platforms supporting OpenGL stereo, the stereo parameters are accessible through a GEOM CLI command, *geom_set_stereo_params*.

```
geom_set_stereo_params  Sets the stereo viewing parameters
     Usage: geom_set_stereo_params -eye <V> -dist <V> -near <V> -far <V>
```

This command ultimately controls the values being used in setting up the stereo viewing matrix.

- eye: This is the "eye offset" controlling the apparent eye separation from the mid plane (nose). Used along with a fixed constant in setting the left and right vertical clipping planes. Initial value can also be set using the environment variable called AVS_STEREO_EYE_OFFSET; set by default to 0.025.
- dist: Distance of the eye from the center of the space being viewed. Initial value can be set using the environment variable AVS_STEREO_EYE_DIST; set by default to 2.0.
- near/far: Specify distances to the near and far depth clipping planes; both distances must be positive. Initial value can be set using the environment variables AVS_STEREO_NEAR and AVS_STEREO_FAR; set by default to 1.0 and 6.0 respectively.

 AVS 5.5 on UNIX Platforms

The source code for the internal function that uses these values shows exactly how they are used to make a *glFrustum* call in OpenGL to create the perspective viewing matrix.

```
void
stereoFrustum(GLfloat near, GLfloat far,
              GLfloat eyeDist, GLfloat eyeOffset)
{
    GLfloat eyeShift = (eyeDist - near) * (eyeOffset / eyeDist);
    GLfloat limit = 0.425; /* from Express routine to try to match stereo */

    glFrustum(eyeShift - limit, eyeShift + limit, -limit, limit, near, far);
    glTranslatef(-eyeShift, 0.0, 0.0);
}
```

With no parameters ("geom_set_stereo_params"), the current values will be displayed. Use of either or both parameters will modify the current values. Further information on CLI commands can be found in the *Command Line Interpreter* chapter of the *AVS Developer's Guide.*

Japanese, Greek, and Cyrillic Labels in Geometry Viewer

A font type called **Kanji** has been defined for labels in the Geometry Viewer. Kanji labels are supported in the software renderer using X window fonts. The X font pattern for Kanji is:

```
-jis-fixed-medium-r-normal-*
```

Use the **xlsfonts** program to discover if your X server has any matching fonts installed. Typically, you will find one or more point sizes for the JIS X 0208 1983 character set. Though named "Kanji," this character set can also represent all of the following fonts:

- Roman, Greek, and Cyrillic alphabets
- Hiragana and Katakana (Japanese syllabaries)
- JIS Level 1 and Level 2 Kanji (Chinese characters used in Japanese)

At present, there is no way to interactively type in or edit Kanji test in the **Labels** typein. However, Kanji strings will be displayed correctly in the **Labels** typein if the label is selected.

The font options **Bold** and **Italics** will not have any effect for these Kanji fonts.

Label strings can be specified in JIS, Shift-JIS, or EUC encodings. These are extended two-byte character codes.

The labels are typeset horizontally reading from left to right.

You input the text by coding the test into a user module with the GEOMadd_label function.

Here is a sample code fragment using an EUC coding for the three kanji characters. Each character is assigned two octal codes.

```
char kanji_string[] = { "306374313334270354" };
int font_number, label_flags;
GEOMobj *kanji_obj;

font_number = GEOMget_font_number( "Kanji", 0, 0 );
label_flags = GEOMcreate_label_flags( font_number, remaining params );
kanji_obj = GEOMcreate_label( GEOM_NULL, label_flags);
GEOMadd_label( kanji_obj, kanji_string, remaining params, -1 );
```

Consult your platform's release notes chapter to see whether Kanji labels are supported in your hardware renderer.

Programming Considerations

There are some issues that are important for developers to know for all platforms.

Use the Example Makefile as Template

You should use the makefile in *<installdir>/avs/examples/Makefile* as the template for your own FORTRAN and C module makefiles. This makefile in turn includes the file *<installdir>/avs/include/Makeinclude* that contains additional macro definitions appropriate to the platform you are using.

GEOMint_color

The GEOM library treats **integer** and **integer color** primitive data in a similar way, so these types should be the same length (32-bits). A new type has been defined for integer colors in the GEOM library which users should use for portability in both 32 bit and 64 bit platforms. For the C binding, **GEOMint_color** is defined in *geom.h* to be **unsigned int** on 64-bit platforms, such as Compaq Tru64 UNIX and SGI N64 and **unsigned long** on 32-bit plaforms, such as AIX and Solaris. FORTRAN should use INTEGER*4 in these cases. **Note:** Only programmers writing modules that use the *libgeom.a* library should be affected by this change; new compiler warnings may arise if function prototypes are used (C++ or Ansi-C compilers).

The programming interface to the following functions has been modified to accommodate this new type:

- GEOMcreate_mesh_with_data
- GEOMcreate_polyh_with_data
- GEOMcreate_sphere
- GEOMset_color
- GEOMset_pickable
- add_to_vlist_from_int
- create_vlist_from_int

C++ Support

Users writing C++ modules should use a special version of the subroutine module library called *$AVS_PATH/lib/libflow_C.a*. This library uses a *main()* function that has been compiled under C++, thereby including C++ initialization calls that were not invoked using *libflow_c.a*. Subroutine modules making use of C++ stream I/O functions must use this library to work properly, while other C++ modules may not require it.

The C++ examples in *$AVS_PATH/examples* should build directly in most environments. The *Makefile* and *Makeinclude* files were changed to use the most common names and paths for the C++ compilers and libraries. Read the Makefile for more specific information and examples.

FORTRAN Modules on 64-bit systems

On 32-bit platforms, integer (int) and pointer (int*, void*, etc) types are the same size and have historically been used as if they were the same. On most 64-bit platforms (Compaq Tru64 UNIX and SGI N64) pointers are 8 bytes and integers are 4 bytes; the two data types can NOT be used interchangeably without causing problems (bizaare values, 0's, etc.)

Portability issues and tools

Many AVS FORTRAN functions pass pointer values which have usually been declared as FORTRAN INTEGER variables on 32 bit platforms; FORTRAN does not have a standard "pointer" data type. On the Compaq Tru64 UNIX and SGI N64, these pointers MUST be declared as INTEGER*8 (8 byte) variables to avoid the module crashing with bad data. This primarily affects compute function arguments (input and output data pointers for complex data types like fields) and array offset values used in some data access functions such as

AVSfield_data_offset.

There is no standard data type to handle pointers on both architectures so truly portable FORTRAN code is not possible. One solution is to use C-like macros for the "pointer" data type and a macro preprocessor but this is not commonly used in FORTRAN and is not universally available on all platforms.

Instead, AVS 5 relies upon a set of "tags" to mark pointer variable declarations and a utility program to convert source code from 32-bit to 64-bit compliant. This allows portability to be managed but does not rely upon a compile-time macro preprocessor.

AVS 5.5 provides the source for the tools that have been used internally for converting files. These utility shell scripts are in the *$AVS_PATH/examples/f77_tools* directory in the Compaq Tru64 UNIX and SGI N64 platforms. They consist of a makefile, a generic converter program (*f77_converter.c*) and a shell script to apply the tool to a file tree (*convert_f77*). For historical reasons, the "tag" value used in AVS is "ALPHA_OSF" but other values can be used in the shell script. **NOTE:** These tools only provide one possible way you may address the portability problem - you are not required to use them, particularly if you do not plan on porting your code.

As an example, let's look at the sample FORTRAN file *CHEMelestf.f* in */usr/avs/examples/chemistry.*

You'll notice that all integer variables that are really pointers in disguise are declared as INTEGER*8.

```
C %IFDEF ALPHA_OSF
      integer*8 mol_input,ep_field
C %ELSE
C     integer mol_input,ep_field
C %ENDIF ALPHA_OSF
```

AVSfield function argument changes

IMPORTANT NOTE: Because of problems encountered with 64-bit pointers under SGI N64, several functions needed to be modified for 64 bit systems. In particular functions which pass "array offset" values had to be modified to use INTEGER*8 arguments to be able to hold large enough offset values. The following changes were made and affect the Compaq Tru64 UNIX and SGI N64 platforms ONLY.

AVS 5.5 on UNIX Platforms

```
AVSfield_data_offset( field, offset, basevec)
AVSfield_points_offset( field, offset, basevec)
AVSload_byte(base, offset)
AVSstore_byte(base, offset, value)
AVSload_short(base, offset)
AVSstore_short(base, offset, value)

   offset is INTEGER under 32-bit platforms, INTEGER*8 under 64-bit platforms
```

Compaq Tru64 UNIX data offsets

Due to the way the FORTRAN compiler generates code for subscripting, the trick with **AVSfield_data_offset** and **AVSfield_points_offset** doesn't work. The compiler does support %VAL() for function arguments, so the more straightforward **AVSfield_data_ptr** and **AVSfield_points_ptr** can be used.

```
C %IFDEF ALPHA_OSF
      call load_pnts(%val(AVSfield_points_ptr(ep_field)),
     1                            res,extents,xinc,yinc,zinc)
C %ELSE
C     iresult=AVSfield_points_offset(ep_field, coords, ocoords)
C     call load_pnts(coords(ocoords+1),res,extents,xinc,yinc,zinc)
C %ENDIF ALPHA_OSF
```

Fixed in AVS 5.5

The following bugs have been fixed in the AVS 5.5 release. Those affecting multiple platforms are covered here. Additional information may be found in the platform specific chapters.

User Interface Issues

7389/7517: AVS Message popup window should be a regular window

Problem Description:

The popup message window used to support the AVSmessage, AVSwarning, AVSerror, and AVSfatal information calls did not have window decorations or other conventional window manager access, preventing it from being iconified, moved, resized, or reordered amidst other windows. It has been changed to work like all the other standard windows.

7260: Typein widgets don't handle strings over 64 characters well

Problem Description:

Typein widgets will handle value strings of arbitrary length but do not scroll or resize dynamically with longer values.

Workaround: The typein widget will generally size itself to at least contain the initial value string being used. Setting the default to a long string initially will stretch it out. Using the width property is another way to control the size to make it large enough for the desired value.

17567: Demos: AVS5 Features/AVSGraph menu empty

Problem Description:
This submenu was still referencing an discontinued version of AVSGraph from an earlier release. It has been updated to reference the current AVSGraph module's sample scripts.

AVS Kernel, LibFlow or CLI issues

7824: image_list_image_names command returns error incorrectly

Problem Description:
The *image_list_image_names* CLI command added in AVS 5.4 always returned an error code at the end incorrectly. This has been fixed.

17213: Demo script information window clipping comments

Problem Description:
The demo viewer script controller window and the network editor script controller window were clipping long comments from view. This has been fixed.

17441: Demos leave some viewer windows behind after exit

Problem Description:
Many of the demos create geometry viewers and their associated windows while they run, and have only cleaned up these windows after a full run is completed and the clean up commands are run at the end of the script. This would tend to leave around unused windows after incomplete demos. This has been fixed so that starting a new script or leaving the AVS Demos application will cause a general cleanup of left over geometry and image viewer windows.

7728: AVSdata_alloc allocates fewer bytes than needed

Problem Description:
In some cases, the AVSdata_alloc routine allocated fewer bytes than were needed for user data structures, depending on alignment issues on different platforms. The allocation size has been padded to cover these differences.

8064: Some AVS modules ignore the AVS -path option

Problem Description:

AVS 5 uses the AVSpath value to determine its home location and this can be set in various ways - command line option, environment variable, etc. While the kernel itself worked with the AVSpath value and communicated it to the external module processes, many of the modules paid no attention to it, so they relied on finding AVS in the default */usr/avs* location.

This has been fixed by modifying all AVS 5 supported modules to use the value of the AVSpath variable when they need to know where AVS 5 is. Modules affected include the Geometry Viewer, the Data Viewer, vbuffer, and AVSGraph among others.

7322: UCD_TRIANGLE definition missing from FORTRAN include files

Problem Description:

The UCD_TRIANGLE definition was missing because of a bug in the FORTRAN interface program, *f77_binding*. This has been fixed and the definition is included in the *avs.inc* files for AVS 5.5.

17656: geom_save_postscript documentation missing argument

Problem Description:

The **geom_save_poscript** CLI command does not mention how or where it wants to get a filename argument, leaving the user to guess. It expects the final argument "filename" to come after the preceding -option phrases. The builtin help has been updated to reflect this.

Geometry Viewer issues

16962: Stereo picking inaccurate

Problem Description:

Picking in stereo was problematic because it was using the non-stereo transformation matrix to determine what was being picked. This has been improved.

16967: Stereo view different from original non-stereo view

Problem Description:

When switching to stereo, a new transformation matrix must be used for the new view. The matrix used substantially changed the appearance of the scene. This has been fixed by adjustments in the stereo transformation matrix that much more closely match the original non-stereo view.

Licensing Issues

7793/7812: Developer's AVS: AVS-Runtime encryption problems

Problem Description:

AVS 5.4 and earlier releases used a different encryption code for runtime licensing than what was used in the standard licenses. With the upgrade to FlexLM 4.x in the AVS 5.3 upgrade, support for this second encryption code was lost causing problems with existing licenses in the field and temporary difficulty making working licenses. This has been fixed for AVS 5.5 and offers the Developer customer two options:

1. AVS 5.4 and earlier: AVS-Runtime features use the alternate encryption code and must be run using the old FlexLM (2.4c) AVS lmgrd daemons provided with AVS5.02. AVS5.3 and AVS5.4 use these codes but their lmgrd daemons won't recognize them.

2. AVS 5.5 and AVS 5.4 patch: AVS-Runtimes are generated using AVS 5.5 or AVS 5.4 with a patch available from AVS Customer Support. They use the same encryption codes as standard AVS, and must run using FlexLM 4.x or higher daemons. AVS5.5 and beyond will make this the default code. Existing AVS 5.4 or earlier runtime licenses should be reissued by Customer Support.

If you are affected by this issue, please contact AVS Customer Support (support@avs.com) for more information or new licenses.

11139: Demo_license Y2K problem

Problem Description:

The demo_license program used for creating temporary licenses for users evaluating AVS 5.4 was using 2-digit years. The resulting licenses still appeared to work, but the program has been upgraded to handle 4-digit years. This issue did not affect the permanent licenses given to registered users.

17568: Animator and BTF Renderer need to be unlicensed

Problem Description:

Both the Animator module and its supporting modules, and the BTF renderer modules have been unlicensed and are now available to all users.

Module Issues

8529: Module Generator puts 'struct' in user data declarations

Problem Description:

The Module Generator erroneously was adding the "struct" keyword to user data declarations, causing compiler errors and requiring manual editing. This has been fixed.

7822: Arbitrary slice plane may use wrong field extents

Problem Description:

The arbitrary slice module selects a plane size based on the XY extent of the object being viewed. Depending on the object, this plane may not be large enough as it is rotated around within the overall dimensions of the object - for example, an oblong brick object with the XY cutting across the small dimension of the brick would use a small square slice plane that won't extend across the length of the brick.

This has been fixed by adding a new choice parameter to the module called "Slice Size" with the following choices:

- XY slice: use the XY slice as the plane size
- YZ slice: use the YZ slice as the plane size
- ZX slice: use the ZX slice as the plane size
- Diagonal slice: use the diagonal of the extents (across opposite corners) to determine the size of the plane.

3428: Set View module doesn't work properly on SGI platforms

Problem Description:

The set view module was producing a bad transformation matrix on the SGI N64 platform causing the geometry viewer to show nothing in some views. This has been fixed.

14838: read_ucd does not recognize tab delimited files

Problem Description:

The read_ucd module did not recognize the use of tabs as delimiters in data files. It has been fixed to recognize these as it does other white space characters such as space and end-of-line.

7426: Module Generator produces incorrect FORTRAN code for UCD calls

Problem Description:

The Module Generator was producing references to "UCDstructure_x" calls based on the C naming convention rather than "UCDstruct_x" as used by the FORTRAN API. It was also using the function call convention ("call UCDstructure_free") rather than the subroutine calling convention (result = UCDstructure_free()"). Both of these problems have been fixed.

7738/7825: AVSGraph needs AVS_PATH to be set

Problem Description:

The AVSGraph module needs to find some underlying Toolmaster definition files in the AVS home directory. When AVS was not installed in */usr/avs* it would fail to run at all. This has been fixed so that AVSGraph checks the definition of the AVSpath variable to find the home directory.

7745: AVSGraph does not remove temp file

Problem Description:

The AVSGraph module was always writing debugging information to a log file in the */tmp* directory that was named AVSGraph.log<processid> which it left around when it exited. AVSGraph has been modified to NOT write out any file unless the user indicates that the log file is desired by defining the AVS_AGX_DEBUG environment variable to any value.

7756: AVSGraph writes to unnecessary temp file

Problem Description:

The AVSGraph module was always writing debugging information to a log in the /tmp directory named AVSGraph.log. When multiple users tried to use AVSGraph, the file was written out owned by the first user, then subsequent users would fail because they couldn't gain write access to the same file. The file naming convention has been changed to AVSGraph.log<processid> to always create unique names to avoid this problem.

16167: UCD_Legend typein parameter override

Problem Description:

Difficulty was reported using typein parameters for 'value', 'lo value' or 'hi value'. Whenever a parameter or the radio dials change, the module compute routine ensures that the 'radio dials' are displayed correctly and are in sync with the parameter widget dials and legend scale.

The problem occured when the parameters were given a typein value instead of using the dials. The value was scaled to the radio dial, which was then scaled to the data, which then over rode the

AVS 5.5 on UNIX Platforms

typed in value, preventing the user from controlling the precision as desired.

This has been fixed by preventing this circular update when the user provides typed in data.

16417: Minmax module crash

Problem Description:

The minmax module was crashing with repeated use due to memory not being reallocated correctly. This has been fixed.

17422: Color legend: Text disappears when increasing thickness

Problem Description:

When modifying the thickness parameter, the spacing of the text below or to the side of the color legend bar was adjusted proportionately, often sending the text off the screen for a thick bar. The spacing calculation has been changed to not reflect the thickness but to make some adjustment for the font height instead.

13262: geom_set_camera_name has no effect on window decoration

Problem Description:

The **geom_set_camera_name** CLI command can be used to set the name of geometry viewer cameras, which are names like "Camera 1" by default. However these did not have any visible impact on the user interface and were of limited value.

AVS 5.5 has been modified to display the Camera name as part of the standard window decoration. Changes made using this command are immediately reflected in the window title and icon names, making it easier to identify cameras. Also see the new "Geometry Camera" module documented under "New Features" that provides a camera "wrapper" similar to the main view window associated with the "Geometry Viewer" module, providing Layout Editor capability, pull down menu, close-window resistence, etc.

Known Problems

8195: Inconsistent ambient light coeeficient

Problem Description:

The default coefficient for ambient light (hardware mode) on GL platforms is 0.2, and the default object coefficient (software mode) for ambient light is 0.3; this results in brighter objects when switching from hardware renderering to software rendering.

8549: Blank lines in CLI scripts cause problems

Problem Description:

Empty lines in CLI scripts may cause problems during script playback. Under some circumstances the CLI may interpret the previous line alright but then get stuck on the blank line. AVS is still interactive and the user can abort or interrupt the script. Remove empty lines or use the "#" leader to indicate the line is a comment line.

14421: Shared memory segment left after AVS exits

Problem Description:

AVS allocates an initial shared memory segment to act as an index of all the other shared memory segments used by the kernel and module for passing Field and UCD data around. On exit, this segment may be left in use as various modules are in the process of exiting and cleaning up the data memory segments. Sometimes this results in this index memory segment being left around after all other processes have completed, something you may see when using the *ipcs -m* command. It will be reused by the next AVS session by the same user so it is not a cumulative problem. If desired you can manually remove this segment using the *ipcrm* command.

17399: AVS Demos - Network Editor 'exit' button may cause crash

Problem Description:

When running the AVS Demos (main menu, Applications, Demos), the user can exit using the pull down menu Control/Return-Exit button or using the "exit" button that may appear on the left panel with module user interfaces. The second approach does not exit as cleanly as the first and in some animation demos, may cause AVS to crash. Please use the pull down menu approach instead.