AVS on LINUX WORKSTATIONS INSTALLATION / RELEASE NOTES

Release 5.5 Final (50.86)
November, 1999

Advanced Visual Systems Inc.

Part Number: 330-0140-02 Rev B

NOTICE

Advanced Visual Systems Inc.
300 Fifth Ave.
Waltham, MA 02451

Printed in U.S.A.

CONTENTS

AVS 5.5 RELEASE NOTES

CHAPTER ONE

Release Highlights

This document describes Advanced Visual Systems Inc.'s Application Visualization System (version 5.5) as it runs on Linux workstations.

AVS5.5 NOTES: As a convenience to the reader, changes made for AVS 5.5 are summarized at the beginning of each chapter in "AVS 5.5 NOTES" blocks.

This release contains an operating system update to RedHat 6.0, platform specific improvements (AVSGraph for Linux, shared library kernel to support hardware graphics with shared OpenGL libs), new features and documentation (including an online AVS5 doc set and new debugging chapter) and bug fixes. The AVS Animator is now unlicensed and included in the standard AVS 5 package; its source code is provided in the *examples* directory and described further under Chapter 3, "AVS 5.5 on UNIX Platforms."

This release provides the following enhancements:

Operating System Updates

Support for the RedHat 6.0 Linux operating system on PC's. This distribution has been tested using the egcs compilers standard in that release.

OpenGL Graphics Library Support

Support for the OpenGL graphics renderer using the Mesa 3D graphics library, an OpenGL-like library that supports software rendering. A second kernel, *avs_shr*, is now provided which uses shared Mesa libraries that can be replaced by the user with OpenGL libraries to support hardware graphics adapters. **Note: Given the variety of cards and the rapidly changing market place, AVS only supports and tests software rendering using the Mesa libraries. AVS makes no representation as to the reliability or quality of the graphics output when using substitute shared libraries.**

Foundation Improvements

A number of product improvements are in this release. These include several new example modules, a new Geometry Camera module, and new documentation of existing features. See Chapter 3 for cross platform changes and Chapter 4 for Linux specific issues.

Documentation

Several AVS 5 documentation improvements have been made with this release.

The AVS 5 documentation set is now available online in Adobe PDF files providing the core set of books in a viewable, searchable, and printable format. The set can be installed (**AVS5_DOC** product archive) or read from the CD (*/cdrom/avs5_doc* on most platforms). For more information, see Chapter 2, "AVS 5.5 Installation" for installation options and Chapter 3, "AVS 5.5 on UNIX Platforms" for information on obtaining an Adobe PDF reader for UNIX or Windows platforms.

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 and prior releases. This chapter draws together information and techniques from throughout the AVS 5 product and is targeted towards both new and experienced module writers.

Licensing

The Linux port does not use any licensing software. The AVS/Animator is now completely unlicensed on all platforms and the full source code for it has been included into *$AVS_PATH/examples/animator*.

Organization

These *Installation and Release Notes* contain the following chapters.

Chapter 2: AVS 5.5 Linux Installation

Describes how to install AVS on your system from the media kit.

Chapter 3: AVS 5.5 on UNIX Platforms

Provides information common or important to all platforms, including how AVS works under X windows and common programming considerations. This chapter also includes a list of general product improvements for the AVS 5.5 release. All users should review this chapter in addition to the specific chapter for their platform.

Chapter 4: AVS 5.5 for Linux

Describes platform specific information for the Linux platform

including system prerequisites, release notes, and recent improvements and known limitations.

Chapter 5: Extended Features
Describes a number of extensions made to AVS 5 in recent releases, including the Cool CD, AVS/Voxel, Japanese Help and additional Demonstration networks.

Chapter 6: Debugging in AVS 5
Provides a summary of suggestions and tips on developing and debugging AVS modules; it draws together a number of techniques and features that are available in different areas of the product.

AVS 5.5 Media Kit

The AVS5.5 media kit contains the AVS5.5 Linux CD and release notes. New users will also receive the AVS5 Documentation Set and Cool CD.

The Cool CD provides a wealth of third party software from the IAC and other sources. The UCD Builder software on this CD does not currently support Linux.

For AVS 5.5, the Cool CD was not updated; for the latest public domain modules and information, users are invited to visit the International AVS Center web site at the University of Manchester at *http://www.iavsc.org*. For more information see Chapter 5, *Extended Features*.

Implementation Overview

This product is a full implementation of Advanced Visual Systems Inc.'s Application Visualization System (Release 5.5) on Linux workstations. With the addition of AVS/Graph and a shared library strategy for accessing hardware graphics accelerators, the Linux port is now comparable to other front line UNIX platforms.

Features introduced in recent releases are described in the *Extended Features* chapter below.

AVS is fully described in the following manuals that accompany this product in hardcopy and in the PDF archives on the CD:

- *AVS Technical Overview*
- *AVS User's Guide*
- *AVS Tutorial Guide*

- *AVS Module Reference Manual*
- *AVS Developer's Guide*
- *AVS Application's Guide*
- *AVS 5 Update*
- *AVS/Graph User's Guide*

The AVS Animator, which is now part of the standard AVS 5.5 release is documented in its own manual, *Animating AVS Data Visualizations.*

The *UCD Builder User's Guide* describes the UCD Builder, an unsupported product delivered on the Cool CD. Note: This product is *not* available for Linux at this time.

The Molecule Data Type, the *libchem* library that manipulates it, and the collection of chemistry and example modules that show how to program with *libchem* are documented in the *Chemistry Developer's Guide* manual. This manual is not included in hardcopy with the AVS release but is available in the online PDF archive. It can be requested separately from support@avs.com and will be provided *free of charge.*

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.

AVS 5.5 INSTALLATION

CHAPTER TWO

Introduction

AVS 5.5 is supplied on a multi-product compact disk (CD-ROM) providing AVS5 for Linux along with optional products including *AVS_DEMOS*, a collection of extended product demos, and *AVS_JHELP*, a Japanese version of AVS online help files.

There are five steps to an AVS5 installation:

Step 1
Decide where you want to install AVS5

Step 2
Mount the CD

Step 3
Install AVS5 from the CD, using either the RedHat RPM package manager or using the *install.avs* installation script to unpack tar archives

Step 4
Install the man pages if desired. Because the RPM packages are relocatable this can't be done reliably using RPM.

Step 5
Inform your users how to set the variables necessary to find AVS5 when they try to start it

AVS5.5 NOTE: No significant changes in installation have been made from the AVS 5.4 release, except for the addition of the AVS5_DOC archive containing the AVS 5 doc set in PDF format. If you are already familiar with this process proceed with mounting the CD, running *gnorpm*, and select which product(s) to install.

Remote Installations

The installation process assumes that the system reading the installation media is where you also want to install the software. There is no provision for "remote" data reads across the network. If the system with the CD-ROM drive can't access the file space where you want to install AVS5, then:

- Mount the CD as described in Step 2.
- Copy the necessary files from the CD to disk space that is accessible to the remote system. This requires either the RPMS directory or the *install.avs* script and the required <*product*>*.Z* files.
- Execute Step 3 to install the software on the remote system, using the disk space pathname as if it were a CD.

Step 1: Decide Where to Install AVS5

AVS5 assumes it will be installed under */usr/avs*, but it can be installed anywhere in a file system hierarchy.

The Linux port requires around 55 Megabytes of disk space, with more space required to compile the module examples or to install optional packages. This sort of space may not be readily available under */usr* so you may want to select other disk space.

Decide on a location that will have adequate space and access for your users. Once installed you can need to tell your users how to set up their path environment to use AVS. For more information, see "Step 5" instructions below.

Step 2: Mount the CD

Mount the CD-ROM to make its contents accessible. Login as the "root" user, insert the CD and use the appropriate mount command for your Linux distribution. The standard command for RedHat is

```
mount -r /dev/cdrom /mnt/cdrom
```

NOTE: If you experience difficulty mounting the CD, please confirm that it is functioning normally using other CD's you have, such as the RedHat distribution CD or by running */usr/bin/X11/xplaycd* with a music CD.

When you are finished, you will need to unmount the CD, which usually uses this command:

```
umount /mnt/cdrom
```

Step 3: Install AVS5 from the CD

There are two different ways to install AVS5 and the optional products from the CD. Users of the RedHat Linux distribution should use the RPM mechanism for greater convenience installing and managing the products. If RPM is not available on your system, you can use the traditional AVS5 installation tool, *install.avs*, which presents a list of available products, checks space requirements, and handles unpacking the selected tar archives. Either way you select, the package contents are the same.

RPM installation

RPM packages can be installed either using the GUI interface (*gnorpm*) or using the command line interface (*rpm*). The GUI interface is the most convenient but will only allow you to install AVS5 into the default location, which is */usr/avs*. The command line interface provides an option to install AVS5 into a different location, because the AVS5 RPMS are "relocatable".

RPM installation using gnorpm

The basic steps for installation are as follows:

1. Login as "root" and startup the *gnorpm* package manager, usually from the menu bar.

2. Hit the "Install" button, then hit the "Add" button, and select the directory */mnt/cdrom/RPMS/i386*.

3. Select one or more of the following packages (where * varies each release):

   - avs5-5-*: AVS5.5 for Linux, default pathname */usr/avs*, around 55 Megabytes.

   - avs5demos-5*: AVS5 extended demo package, default pathname */usr/avs_demos*, around 25 Megabytes. See Chapter 5 for more information on installing the demo package and setting it up after installation.

   - avs5jhelp-5-*: AVS5 help in Japanese */usr/avs_jhelp*, around 3 Megabytes initially, approximately 10 Megabytes once built.

   - avs5doc-5-*: AVS5 Doc set in PDF format for viewing, searching, and printing. Runs around 40 megabytes.

4. Once you have determined what to add, select "close" on the browser then hit the "Install" button. The packages will be installed into */usr*.

RPM installation using rpm

Using the *rpm* utility directly allows you to install the packages someplace other that */usr*. Once installed, you can use *gnorpm* to query, verify, or remove them later. NOTE: Make sure that *gnorpm* is not running when you try to use *rpm* - they can't both access the RPM database at the same time.

Login as "root" and then list out the actual package names you have to select from on the CD.

```
ls /mnt/cdrom/RPMS/i386
```

Install the desired package(s) where you want. For example, the following command would install the avs5 package under */home/me/avs*.

```
rpm -Uvh --prefix /home/me /mnt/cdrom/RPMS/i386/avs5-5-1.rpm
```

Once installed, you can either set up a soft link from */usr/avs* to the install area,

```
ln -s /home/me/avs /usr/avs
```

or access it from the install area, using one of the techniques described below under "Step 5" below.

When you are finished, you will need to unmount the CD, which usually uses this command:

```
umount /mnt/cdrom
```

Uninstalling RPMS

You can use *gnorpm* to uninstall any of the AVS5 packages.

First "clean up" the installed products.

- For avs_demo, go to the */usr/avs_demo* and run the "uninstall_demo" script to "detach" the demos from /usr/avs/demosuite. Then type "make port_mods_clean" to remove any built modules or data sets.

- For avs, go to */usr/avs/examples* and type "make clean". This will remove any built demos.

Now use *gnorpm* by selecting the Applications and Graphics folders, then selecting the package you want to delete, then selecting the "Uninstall" button. If there are complaints about directories that can't be removed because of leftover files (like built examples, etc) use "rm" to remove these manually.

Traditional AVS5 installation

NOTE: If you have already installed using RPMs, please skip ahead to section "Step 4: Install man pages".

The *install.avs* installation script uses the *tar* command to read AVS 5 off of the media. In almost all cases, *tar* will keep the AVS 5 file protections as they are on the media. The script will catch and warn you when this may not be true.

During installation *tar* will assign the files the user and group ownership of the user executing the installation script. Be conscious of this and execute the installation script as the entity—either root or otherwise—suitable for the pattern of use at your installation.

1. Run the *install.avs* installation script.

   ```
   /mnt/cdrom/INSTALL.AVS
   ```

2. Product Selection:

   You will see a table of possible product choices.

   ```
   This CD-ROM provides the following AVS products:

   Name          Platform    Product         Version       Size in Megabytes
   LINUX         LINUX       AVS             5.5 (50.79)   55  (55034 Kbytes)
   AVS_DEMOS     all         AVS_DEMOS       5.5 (1.4)     25  (25600 Kbytes)
   AVS_JHELP     all         AVS_JHELP       5.01 (1.4)     3  (3072 Kbytes)
   AVS5_DOC      all         AVS5_DOC        5.5 (1.0)     40  (40000 Kbytes)

   Please enter the Name of the product to install [default LINUX] or type quit:
   ```

   Enter the name of the product you want to install, for example:

   ```
   LINUX
   ```

   NOTE: See Chapter 5 for more information on completing the installation of the AVS_DEMOS demo package.

The script confirms the selection with a message like the following:

```
Selecting LINUX
```

3. The script asks to confirm the CD-ROM mount directory:

```
The default CD-ROM directory is /mnt/cdrom
If this is OK, hit return, otherwise enter the appropriate directory:
```

Either press return or enter a different CD-ROM directory.

4. Select Installation Directory

The script prompts for an installation directory. The default ("[/u2]:" for the example) is the current directory. **Note:** the script creates the *avs* directory automatically. Pressing return at this point will create */u2/avs*.

```
Enter directory in which to install AVS
[/u2]:
```

Since you've already changed to the directory in which you want to install AVS, press return.

5. If this directory contained an existing copy of a directory called *avs*, you will see the following:

```
There is an existing directory called /u2/avs.
Move it now or it will be removed.
[Hit return to continue]:
```

If you do not want the existing copy deleted, then use another window to move it or type Control-C to terminate the installation.

6. Next you will see:

```
mkdir /u2/avs; cd /u2/avs
```

followed by a message like the following. The exact format and size estimate varies from platform to platform.

```
Installing AVS in:  /u2/avs

AVS requires approximately:  60 Megabytes (61440 Kbytes) of data
df -k  yields:
File System            kbytes    used   avail capacity  Mounted on
/dev/sd0a               7735    5052    1909    73%    /
/dev/sd0g             151399  126178   10081    93%    /usr
/dev/sd1c             299621  185600   84058    69%    /u2
```

```
/dev/sd3c             189534  126055   44525    74%    /home2

Ensure that /u2 has enough space then hit return or type quit:
```

Note that the actual Kbyte size may vary from what is listed here.

7. Hit return and the installation will begin.

   Note: At the end of the installation the script tells you that you may optionally run "install.avs links to setup links from /usr". Read the discussion under "Step 5" before deciding to do this. The links from */usr* are no longer required.

```
Reading AVS from media
Reading LINUX archive
tar -xvof /dev/rmt/0m
x .version, 33 bytes, 1 blocks
x ./bin/avs_dbx, 2708 bytes, 6 blocks
x ./bin/avs, 2589380 bytes, 5058 blocks
  .
  .
122978 blocks
Read from media successfully
Installation of AVS is complete.
You may optionally run "./install.avs links" to setup links from /usr.
```

Note that the actual block count may vary from what is listed here.

8. **Unmounting the CD**

   If you mounted the CD as root, you will need to unmount as root also. This is usually done with the *umount* command with either the directory name or device name as argument:

```
# umount /mnt/cdrom
```

Step 4: Install man pages

The *avs* man page is supplied in two forms:

- an *avs.6 nroff/troff* source file. This file can be copied to one of the unformatted */usr/man/man* man page directories and renamed appropriately. This is what RedHat 6.0 expects.

```
cp <install_dir>/avs/runtime/help/modules/avs.6 /usr/man/man1/avs.1
```

- an *avs.txt* pre-formatted ASCII text file. This file can be copied to one of the pre-formatted */usr/man/cat* man page directories and renamed appropriately

```
cp <install_dir>/avs/runtime/help/modules/avs.txt /usr/man/cat/avs.1
```

Copy the file, in whichever form is suitable for your system, to its appropriate *man* page directory. See your platform's documentation for more information on how man pages are handled on your system.

You may need to manually update the *whatis* database with the one-line description at the top of the man page to make it findable with the *man -k* command. On Redhat, as the root user, run */usr/sbin/makewhatis* to update the database.

Step 5: Tell Users How to Find AVS

This section summarizes what users need to do to find AVS. The story is described in more detail in the "$Path—Installing and Finding AVS Anywhere in a Directory Tree" section of the *AVS5 Update* manual.

In order to find AVS, you and all of your users will have to define the AVS path or set up links to direct */usr/avs* to point to where avs is installed. Using the path approach is recommended.

Setting the AVS_PATH

Briefly, assuming AVS5 was installed in */u2/avs*, both you and your users would:

1. Add */u2/avs/bin* to your directory file search path.

2. Tell AVS5 where to find itself by one of three methods, in this order of precedence:

   - Start AVS5 with the **-path** option:

     ```
     avs -path /u2/avs
     ```

   - or, define **Path** in your personal *.avsrc* file:

     ```
     Path      /u2/avs
     ```

- or, set the **AVS_PATH** environment variable in one of the user's startup files, usually either *.login* or *.cshrc* (*csh*), or *.profile* (*sh/ksh*).

  ```
  csh:     setenv AVS_PATH /u2/avs

  sh/ksh:  AVS_PATH=/u2/avs
           export AVS_PATH
  ```

  It is convenient to define **AVS_PATH** even if the other mechanisms are used so that one can find AVS5 files by referring to $AVS_PATH/*filespec*.

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

Setting up links

To avoid the need to work with the path, you can set up links that point from */usr/avs* to the actual installation area. Additional links are needed to find the include directory, etc. The *install.avs* install script offers to set these up for you but it may be easier to set them up manually so you know explicitly what is involved.

Assuming that AVS is installed in */u2/avs*, you would create the following link as the root user:

```
ln -s /u2/avs /usr/avs
```

These additional optional links can be added to make it easier to run the standard AVS executables and to more readily reference the AVS include files:

```
ln -s /usr/avs/bin/avs /usr/bin/avs
ln -s /usr/avs/bin/avs_dbx /usr/bin/avs_dbx
ln -s /usr/avs/include /usr/include/avs
```

Distribution Contents

All files for this distribution are kept in the directory in which you installed AVS.

The following is a detailed listing of the contents of this release:

avs/LUI
: Contains LUI-specific runtime files, the pixmaps for the icons used in AVS.

avs/applications
Contains the command files for the canned applications.

avs/avs_library
Contains the supported module executables. Also contains the Animator executable and the animation module executables.

avs/bin
Contains most of the geometry conversion executables that were created in the *filter* directory and the *avs* and *avs_dbx* programs.

avs/chem_lib
Contains the Chemistry module executables.

avs/data
Contains sample datasets for use with AVS.

avs/demo
Contains a number of demonstration CLI scripts that allow you to more easily explore AVS5 functionality. You can access the scripts through the Help Demos button on the Help Browser.

avs/demosuite
Contains all the CLI scripts and help text associated with the AVS Demo Suite. You can access the scripts through the AVS5 Demo button under AVS5 Applications.

avs/examples
Contains source files for example AVS5 modules. It also contains a *Makefile* that you should use as a template for your own module make files.

avs/filter
Contains the source and some example data for building geometry tools and for template/example geometry converters.

avs/include
Contains the header files necessary for using the AVS5 libraries.

avs/lib
Contains the library files necessary for using the AVS5 libraries.

avs/math
Contains information and files to enable a direct interface between AVS5 and Mathematica™. (See the README file in that directory for more information.)

avs/networks
Contains links to the viewer subdirectories of *avs/applications*, which can be used to access sample networks that make up the

canned viewer applications.

avs/relnotes
Contains printable PostScript versions of these *Installation and Release Notes*. The printer must support the Palatino font.

avs/runtime
Contains AVS-specific files that must exist in order to run *bin/avs*, including the online *help* files.

avs/test
Contains an automated test procedure for verifying AVS5 operation.

avs/unsupp_mods
Contains the unsupported module executables.

Source Code

Source code provided with this release consists of templates and examples that are useful for creating new geometry converters (in *<installdir>/avs/filters*) and AVS modules (in *<installdir>/avs/examples*). The AVS/Animator module source code is now under the *examples/animator* directory as well. For more information see Chapter 3, "AVS 5.5 on UNIX Platforms".

Source code for AVS supported modules may be requested through AVS Customer Support. Please see Chapter 6, "Debugging in AVS5.5" for more information.

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.

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.

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.

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.

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.

```
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.

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

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.

AVS 5.5 FOR LINUX

CHAPTER FOUR

Introduction

This chapter describes AVS 5.5 as it runs on Linux workstations. It covers hardware and software prerequisites needed to run AVS; differences and special considerations for running AVS on this platform; and known problems.

AVS5.5 NOTE: The operating system support has been upgraded to RedHat 6.0 and the AVS/Graph module is now fully supported based on Toolmaster 7.1. A second version of the AVS kernel, (*avs_shr*), linked dynamically to the Mesa graphics libraries, permits users to substitute their own local versions of OpenGL drivers to support hardware graphics adapter drivers. A few additional bugs have been fixed.

NOTE: See the section below on Data and Module portability in "Programming Considerations" for important discussion such as "little-endian" data format compatibility. Additional information on using AVS under X windows, general programming considerations, and bugs fixed in AVS 5.5 can be found in the *AVS 5.5 on UNIX Platforms* chapter.

Hardware Prerequisites

Workstation Models

AVS will execute on any x86-based PC workstation meeting the requirements described below.

Graphics Hardware

This release of AVS does not directly support any specific graphics hardware due to the evolving Linux marketplace. AVS 5.5 does provide a dynamically linked version of the AVS kernel, *$AVS_PATH/bin/avs_shr*, which allows users to experiment with

substituting OpenGL driver libraries provided by graphics adapter vendors. The Mesa libraries *avs_shr* was linked against should provide basic OpenGL API compatibility. **Note: AVS makes no representation as to the reliability or quality of the graphics output when using substitute shared libraries.**

In order to use this shared library kernel, set the library path as follows:

```
setenv LD_LIBRARY_PATH $AVS_PATH/lib
```

to enable *avs_shr* to locate the default Mesa libraries, *libGL.so* and *libGLU.so*. To confirm where *avs_shr* is obtaining the shared files, type:

```
ldd $AVS_PATH/bin/avs_shr
```

If you have alternate vendor specific versions of *libGL.so* and *libGLU.so*, set the LD_LIBRARY_PATH variable to the appropriate directory where they may be found. **Note:** The Mesa *libGLU.so* library has been successfully mixed with at least one hardware vendor's *libGL.so* shared library when needed.

AVS 5 uses two different mechanisms to produce its screen renderings of Geometry Viewer objects and scenes however. The "**hardware**" **renderer** uses the Mesa 3D graphics library which performs software rendering unless it is replaced by the user with hardware drivers. The standard AVS5 **software renderer** 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.

See the "Geometry Viewer" section below for specific information on which rendering features are supported by the software and hardware renderers.

Memory Requirements

AVS requires that the system upon which it will execute be configured with a minimum of 64 megabytes of main memory. More real memory, such as 128 megabytes, is strongly recommended and will improve performance. In addition to real memory requirements, there is a minimum system swap space requirement (see below).

Disk Space

AVS 5.5 requires approximately 65 megabytes of disk storage to install AVS from the CD. More disk storage may be required for swap space (see below). The *df -k* command reports available disk space.

Determining Your Configuration

To interactively determine your workstation's configuration, use the following commands.

To determine your memory configuration,type:

```
cat /proc/meminfo
```

and you should see a display that looks like this:

```
        total:    used:    free:  shared: buffers:  cached:
Mem:  64675840 62750720  1925120 30797824 27463680 20041728
Swap: 133885952   151552 133734400
MemTotal:     63160 kB
MemFree:       1880 kB
MemShared:    30076 kB
Buffers:      26820 kB
Cached:       19572 kB
SwapTotal:   130748 kB
SwapFree:    130600 kB
```

where "MemTotal" is the amount of RAM and "SwapTotal" is the amount of swap space.

To find out more about your CPU, you can type

```
cat /proc/cpuinfo
```

which will describe your CPU model, clock speed, and vendor.

Software Prerequisites

Linux Operating System

There is only one version of AVS for Linux. The supported Linux distribution is RedHat 6.0 (kernel 2.2.5-15) for the x86 PC family. Compatible Linux distributions (based on glibc) will likely work but have not been tested.

The */bin/uname -a* command will list which version of the Linux kernel is running on the system.

Compilers

AVS5 was built and tested under RedHat 6.0 (kernel 2.2.5-15) using the standard egcs (2.91.66, release 1.1.2) compilers. It has not been tested under the older versions of RedHat 5.x - these are supported under AVS 5.4.

The Fortran compiler used was g77 (2.91.66), based on the egcs compiler.

Dependencies

The AVS5 packages depend on the following libraries to be installed:

- libm.so.6
- libc.so.6
- libXext.so.6
- libX11.so.6
- ld-linux.so.2

X Servers

Swap Space

Recommended Swap Space Size

We recommend that your system be configured with a minimum swap space size that is twice the real memory size or 128 megabytes, whichever is *greater*. Very large datasets flowing through networks with many modules and connections may require even larger swap spaces. As noted above typing "cat /proc/meminfo" will tell you how much swap space you have.

Running out of Swap Space: Message, Abort, or Hang

Depending upon the circumstances, when AVS runs out of swap space it either produces a message box, aborts, or hangs. In many cases, AVS itself receives the error during a *malloc* call. In this instance, either individual modules or the entire AVS system may abort with the following error message:

```
Failure allocating memory:
See Installation/Release Notes to increase swap space and/or
shared memory segment size
```

This message may be followed by many additional "tcp read" error messages as each module expires. Scroll to the top of the error messages to see this first message that indicates the actual cause of the failure.

In some cases, other parts of the system will generate the error and you will not see the above message. If you suspect that swap space may be the culprit, you can restart AVS, repeat the sequence of steps that led to the abort or hang, while monitoring AVS's consumption of swap space by entering successive *cat /proc/meminfo* commands.

How to Increase System Swap Space

See the man page for **mkswap** for more information on how to create and initialize more swap space for your system.

Using AVS on Linux Workstations

The following sections describe in detail the differences between using AVS on a Linux workstation and AVS as it is described in the standard documentation.

General

Dial Box

The Dial Box I/O device mentioned in the *AVS User's Guide* is not supported in this release of this product.

Spaceball

The Spaceball device driver has been linked in but has not been tested for this release.

Window Manager

AVS works with any of the standard X Window System window managers. See the "Window Manager" section in Chapter 3, *AVS 5.5 on UNIX Platforms* for more information on using different window managers.

Starting AVS

There is one difference to starting AVS on a Linux workstation from the account in the AVS *User's Guide* and the *avs* man page. You may need to change your display size if you have a small framebuffer size (1280x768).

Reconfiguring this involves making changes to your personal AVS startup file (*.avsrc*) in your HOME directory. If you don't already have an *.avsrc* file, copy the sample file from *<installdir>/avs/runtime/avsrc* to your home directory and edit it to include the necessary lines.

Display Size

AVS normally expects to execute on 1280x1024 resolution displays. If the display is smaller or larger than this, AVS will try to automatically rescale its interface to fit on the screen. If AVS's automatic efforts are not satisfactory (for example, the bottoms of menus are being clipped by the window manager or the Network Editor panel is overlapping other windows), you can explicitly set a virtual screen size and a position for the Network Editor window by adding lines such as the following to your personal *.avsrc* file:

```
NetworkWindow       650x750+250+20
ScreenSize          1024x768
```

You can also experiment with different display sizes using the *-size* command line option which does the same thing.

```
avs -size 1024x720
```

Note that the font sizes selected for smaller display sizes may be lower quality than the larger fonts selected for the default size.

Image Viewer

All functions and behavior in the Image Viewer are as described in the AVS *User's Guide*. Color rendition will be up to the capabilities of the X server.

On pseudocolor systems, the Image Viewer's **Images** submenu will have a choice of dithering options.

Network Editor

The Network Editor functions as described in the AVS *User's Guide* and *Module Reference* manual.

Modules

The following modules documented in the *AVS Module Reference* manual do not appear in the AVS release for Linux workstations.

alpha blend
transform pixmap

Image Output

While using the hardware renderer, if you are using the **image to postscript** module to get output from the **geometry viewer** module, or any of the Animation Application output modules such as **write frame seq**, all parts of the Geometry Viewer scene window must be visible. No parts of the display window can be off the screen or obscured by other windows. If they are, any module downstream of **geometry viewer** will not receive the off-screen or obscured portion of the pixmap.

Graph Viewer

There are no significant differences between AVS Graph Viewer behavior on Linux workstations and the Graph Viewer descriptions in the AVS documentation. On pseudo color systems image data loaded as a background to a plot will be dithered to 8-bit pseudo color rather than appear in true color.

Geometry Viewer

The Geometry Viewer (or the **geometry viewer** module) will be using either the software renderer or the hardware renderer to produce the contents of its scene windows.

Stereo Support

AVS 5.5 for Linux does not include currently support for stereo viewing; the OpenGL stereo feature may work on Linux but this has not been tested.

Rendering Features

The following table lists the rendering features described in the "Geometry Viewer" chapter of the *AVS User's Guide*, including new AVS 5 features, down the left column, and the AVS software renderer and the AVS "hardware" renderer's (Mesa) implementation across the top. The table intersections show which features are present on each platform, and draw your attention to more detailed explanations of behavior later in this section.

Table 4-1. Geometry Viewer Behavior Across Platforms

Rendering Feature	Software Renderer	Hardware (Mesa)
Arbitrary Clip Planes	yes, 8 planes	no (note 1)
Geometric		
Volume Rendering	yes	yes
Vertex Transparency	no (note 2)	yes (note 2)
Vertex Colors	yes (note 3)	yes
Edit Property		
RGB/HSV Colors	yes (note 3)	yes
Ambient	yes	yes (note 4)
Diffuse Lighting	yes	yes (note 4)
Specular Highlights	yes	yes
Gloss	yes	yes
Transparency	yes (note 5)	yes (note 6)
Metallic	yes	yes
Edit Texture	yes	yes (note 6)
2D Texture Mapping	yes	yes
3D Texture Mapping	yes	no
Filtered Textures	no	yes
Alpha Textures	yes	no
Rendering Options		
Points	yes	yes
Lines	yes	yes
Smooth Lines	no	no
No Lighting	yes	yes
Flat Shading	yes	yes
Gouraud Shading	yes	yes
Outline Gouraud	yes	yes
Phong Shading	no	no
Backface Properties		
Cull front	no	no
Cull back	yes	yes
Flip normals	no	no
Lights		
Number of Sources	16/8 (note 7)	8
Ambient	yes	yes
Directional	yes	yes
Bi-Directional	yes	yes
Point	no	yes
Spot	no	no
Light Colors	yes	yes

Table 4-2. Geometry Viewer Behavior Across Platforms (continued)

Rendering Feature	Software Renderer	Hardware (Mesa)
Cameras		
Depth Cue	yes	yes
Perspective	yes	yes
Accelerate	no	no
Front/Back Clipping	yes	yes
Axes for Scene	yes	yes
Double Buffer	no	yes
Sorted Transparency	no	no
Shadows	no	no
Polygonal Spheres	yes (note 8)	no
Stereo	no	no
Head Tracking	no	no
Labels		
Drop Shadow	no	yes
Stroke Text	no	no
Kanji	yes	no

Notes

1. Arbitrary clipping planes are used by the **clip geom** module.

2. Vertex transparency is used by the **colorize geom** module.

3. When using the software renderer, the number of colors that will appear on the screen (216 pseudo color, 16,777,216 true color, or some number in between) is dependent upon the X server visual support present on the *display* hardware. Internally, 24-bit true color is always used and will be output on the **geometry viewer** module's image output port.

4. The **Ambient** and **Diffuse** sliders have no effect on objects with vertex colors.

5. The software renderer supports single-pass transparency. Single-pass transparency correctly renders a transparent surface that is over an opaque surface. Multi-pass sorted transparency is required to correctly render multiple overlapping transparent surfaces.

6. Enabling the feature has no effect.

7. When, at AVS startup, you initialize the software renderer only (*-nohw* or **NoHW**), then there will be 16 light sources in the software renderer. If, by default, both hardware and software

renderers are initialized, then the minimum common number of light sources will be used. Thus, both renderers will have eight light sources.

8. A feature in the software renderer is the ability to render spheres directly instead of subdividing them into polygons. This saves a tremendous amount of memory when rendering spheres. The algorithm's execution speed is bound by the size of the spheres it has to render. This feature may be disabled by toggling the **Polygonal Spheres** button under the Cameras menu. Spheres will now be rendered by subdividing them into polygons.

Programming Considerations

There are a few platform-specific issues to take into account when developing AVS modules on Linux workstations.

Compiling and Linking Modules

Be sure that you have installed the most current versions of the FORTRAN (g77) and C compilers and their associated libraries that accompany the RedHat 6.0 system.

Data portability issues

The Linux platform uses a data-storage format that is "little-endian" in contrast to most other AVS5 UNIX platforms which are "big-endian" (Compaq Tru64 UNIX is one other little-endian platform). This means that binary data generated by another platform may not be read correctly on Linux because the data format is incompatible. There are a number of ways that this may cause problems:

- **Geometry Viewer crashes:** Incompatible data may be read in and flow through a number of modules until it reaches the Geometry Viewer. The Mesa graphics library is fairly intolerant of bad data in vertex information and will often crash during renderering. You can enable verification of geometry integrity as follows:

  ```
  setenv AVS_GEOM_VERIFY 1
  ```

  When this flag is on, it will double check all the geometry elements for reasonable data. If it complains about "NANs" (invalid numeric values) in the incoming geometry, look back upstream to look for sources of binary data.

- **Unexpected results:** Field values may be read in as extremely small or large values resulting in unusual results down stream. Use the **Print Field** module to examine data values on the Linux machine and compare the output against a different UNIX platform.

There are several ways to address these issues:

- **Convert field data to XDR format:** On the original source machine, use the **Write Field** module (or equivalent for other data types) and write out the data using the "XDR" portable format. XDR will be read correctly on all platforms.

- **Modify reader modules to handle byte swapping:** Modules that read binary data directly should be coded to handle byte swapping. The AVS include file, *avs/include/port.h* has either "AVS_BIGENDIAN" or "AVS_LITTLEENDIAN" defined depending on the platform. If all the source data is "big endian", then the code can be modified to do byte swapping when run on a system that has "AVS_LITTLEENDIAN" defined.

- **Use remote modules to read the data:** Use your reader module on a remote system that can read the data properly. Remote modules then communicate to the local system using the XDR portable format.

Use the Example Makefile as Template

You should use *<installdir>/avs/examples/Makefile* as the template for your own FORTRAN and C module make files. This make file in turn includes the file *<installdir>/avs/include/Makeinclude* that contains additional macro definitions appropriate to the Linux platform, its compilers, compiler options, include files, and libraries.

As an examination of the *Makeinclude* file would show, when compiling modules written in C you must use the *egcs* compiler or modify or override the *CC* macro to use the *gcc* compiler.

Module portability issues

If you are using the same conventions that the AVS examples use (Makefiles including Makeinclude, etc) you should have fewer problems. The following issues were found during porting and beta testing:

- **LEX/YACC problems:** The Linux version of these utilities wasn't always as robust as other platforms used for AVS5. The C code output (lex.yy.c, etc) sometimes needed minor modifications.

- **Gmake - Trailing white space:** All trailing white space must be deleted in variable declarations, otherwise gmake takes the white space as part of the variable, i.e.

Programming Considerations
(continued)

```
on the sun:

SC=/source     #this is a comment (note the white space before the comment)

on linux:

SC=/source#this is a comment (note the lack of white space before the comment)
```

- **Gmake - $$@ doesn't work in dependency line:** gmake does not recognize $$@ on a dependency line. A new makefile rule may need to be written to work around this limitation.
- **Gmake - $$@F doesn't work:** this can't be used to get the filename portion of the make line.
- **-DBSD needs to be defined:** Without this flag (or -DUSE_BSD) defined, some include files like <sys/types.h> will leave undefined data types such as caddr_t, etc. This flag is included in the Makeinclude file as part of the ACFLAGS macro.
- **Porting flags may be needed:** The *avs/include/port.h* file is not included by most customer modules but does provide some helpful cross platform parameters. It is used internally to define various platform specific characteristics, such as endian-ness, BSD or not, data sizes of LONGS, etc. Using it may improve the portability of your module.

FORTRAN Modules

On 32-bit platforms, integer (int) and pointer (int*, void*, etc) data 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 64-bit platforms) pointers are 8 bytes and integers are 4 bytes; the two data types can NOT be used interchangeably. See the section on "FORTRAN Modules on 64-bit systems" in the *UNIX Platforms* chapter for critical information on this topic.

AVS on Linux Workstations: Known Problems

This section lists the known problems with this release of AVS 5.5 that are unique to Linux platforms.

Warning: reducing color usage: R=8,G=8,B=8,Grey=256:

Problem Description:

This message is sometimes received depending on the X server configuration.

X11201: Geometry Viewer - scale reduced to zero not recoverable

Problem Description:

Once objects are scaled down to 0 in the Geometry Viewer, they may not scale back up again. Some other UNIX platforms seem to recover from this situation more readily and will scale back up from 0. This is under investigation.

AVS on Linux Workstations: Fixed Problems

This is the list of fixed problems for AVS 5.5 on the Linux platforms.

17565: Set_view module crashes

Problem Description:

The set_view module was crashing under various circumstances. This has been fixed.

16418: Translate_molecule module crashing on Linux

Problem Description:

The translate_molecule module was crashing on exit and leaving a core file behind. This has been fixed to eliminate the crash.

17440: AVS crashing when exiting some demos

Problem Description:

It was found that after exiting the AVS Demos application, then attempting to reenter the demos, caused AVS to crash. This has been fixed.

AVS on Linux Workstations: Fixed Problems
(continued)

EXTENDED FEATURES

CHAPTER FIVE

Overview

AVS 5.5 is primarily a platform maintenance release; its main purpose is to update AVS to run on major new operating system releases and provide critical bug fixes. For an overview of what is specifically in the AVS 5.5 release, please see the "Release Highlights" in Chapter 1.

AVS5.5 NOTE: The AVS_DEMOS have been revived and retested on all platforms and a number of bugs have been fixed. The BTF renderer (AVS/Voxel) has been unlicensed to help simplify AVS 5 licensing. AVS/Graph has been ported to Linux using Toolmaster 7.1 for its underlying graphics support.

This chapter also describes the Cool CD and UCD Builder materials provided with AVS 5.3 and other features added in earlier releases of AVS 5, such as AVS/Graph, AVS/Voxel, Japanese Online Help, and the AVS Demos. Fixes for known problems are discussed in the introductory UNIX chapter and the chapters for the specific platforms.

Cool CD and UCD Builder

The Cool CD is an extensive collection of public domain and 3rd party modules and data sets, conference proceedings, and other valuable material to expand your use of AVS 5 products.

Much of the Cool CD comes from the International AVS Centre (IAC) through the contribution of users like yourselves who have developed AVS 5 modules and data sets and offered them to the AVS user community. The IAC, now based at the University of Manchester in England, makes these modules available in source form through its World Wide Web site. The Cool CD delivers them to you for convenience but make sure to visit the new IAC Web site for ongoing contributions and updates at *http://www.iavsc.org*.

Additional 3rd party modules are also on the Cool CD, including modules that help you -

- Read and write VRML (Virtual Reality Markup Language), an emerging 3D modeling language for the World Wide Web

- Read input from FEA and CFD applications to create UCD data structures (UCD Builder) (Note: This is not available for Linux)

- Read I-DEAS FEM data (FEMbridge) or write data to a V-LAN based video recorder (VIDEOkit), available for evaluation or purchase through KGT, Inc.

Additional information on other third party AVS5 and AVS/Express modules and applications available through other companies is also included.

The Cool CD also included a snapshot of the corporate website for Advanced Visual Systems; this has become substantially outdated since the CD was last updated. Please visit our online website at at *http://www.avs.com*.

The Cool CD is specially formatted to allow you to browse the contents with your favorite Web browser without having to install anything. Just mount, browse, and choose what you want to use; in some cases, the software can be run off the CD with no installation.

Support

Because the Cool CD contents are primarily from third parties and are not part of the AVS 5 product line, AVS Customer Support will not support them or answer questions about them. The contents of the Cool CD are provided "AS IS", with no warranty or support of any kind. We are making them available to you as a service to make it easier for you to explore and experiment with the wide world of public domain AVS 5 modules available.

If you do have problems, you should contact the appropriate third party for help or suggestions. For modules from the International AVS Centre, start by contacting the IAC directly or the original module contributor if they have provided contact information.

For most of the third Party modules (FEMbridge, VIDEOkit, VRML reader and writer), follow instructions on the Cool CD to contact KGT, Inc. for further information.

The UCD Builder product was developed by Scientific Visualization Associates Inc. Working with Advanced Visual Systems, the company is making its product available to you in binary form for free as a contribution to the AVS 5 community. This is an unsupported product and Advanced Visual Systems assumes no responsibility for it; if you do have problems or questions about it, see the related Web page for instructions. **Note:** It is not currently supported for Linux.

AVS/Graph

AVS 5.5 NOTE: The AVS/Graph module has been ported to Linux using the latest version of Toolmaster 7.1.

AVS/Graph is a graphing tool which allows you to plot AVS field data in a variety of ways (curve, scatter, bar, area, polar, pie, images), control the appearance of the graph (titles, axes, logarithmic scales, legends, tick marks) and generate hardcopy output in a variety of formats (PostScript EPS, PostScript Color EPS, CGM Binary, CGM Clear Text, etc.) AVS/Graph consists of the Data Output module named **AVS/Graph** with its own extensive user interface, a set of demonstration scripts, and a set of example networks.

The **AVS/Graph** module is a part of the Supported AVS Module Library. The module is a synchronous coroutine module (i.e., it runs as an independent process that is triggered when one of its inputs changes).

AVS/Graph is implemented using the Advanced Visual Systems product Toolmaster-agX. The source to the module is included with the release. If you also purchase Toolmaster-agX, you can use the module source as a guide to extending AVS/Graph functionality to perform additional tasks such as contouring, data interpolations and smoothing, 3D graphs, and note and arrow annotations.

For complete AVS/Graph documentation, see the *AVS/Graph User's Guide* included in the standard doc set.

Japanese Online Help

This release contains a complete set of AVS 5 online help in Japanese. All of the material in *runtime/help* has been translated, including the module man pages and the help files for the various subsystems. In addition, users can create their own Japanese language online help files to support their applications.

The k14 and a14 X fonts must be present on your system.

To install Japanese Online help:

1. Use either the RPM (avs5jhelp-*-*.rpm) or the *install.avs* script. When the *install.avs* script presents its list of installable products, select AVS_JHELP. This one help directory supports all platforms and takes approximately 3 megabytes of disk space.

The installation creates an *<install-dir>/avs_jhelp* directory. This directory can be anyway in the file system hierarchy, both within the *avs* installation directory or outside it.

2. With the AVS_PATH environment variable set to point to the AVS directory, execute the script *<install-dir>/avs_jhelp/test/SETUP*. If unset, AVS_PATH defaults to */usr/avs*.

This script adds the correct fonts to the *runtime/avs.Xdefaults* file and sets the AVS_HELP_PATH environment variable to point to the Japanese help files.

3. You can check that the installation went correctly by executing the <install-dir>/avs_jhelp/test/RUNME script.

For information on creating your own Japanese online help files, see:

- *test/kanji.eucf*: Man page that describes how to make a module with a Japanese help file. (This file is in Japanese).
- *test/kanji.scr*: AVS script for making a module with a Japanese help file.
- *test/kanjiall.txt*: The table of EUC codes.

AVS/Voxel

AVS/Voxel is a Back-to-Front (BTF) direct volume renderer consisting of a set of modules provided in the Unsupported library for customers to use and experiment with. **NOTE:** The BTF renderer is not provided on Compaq Tru64 UNIX.

The BTF renderer renders volumes one slice at a time, beginning from the back of the volume, and composites each new slice as it renders the volume. It is based on a high performance algorithm that achieves its speed by exploiting memory coherence and by optimizing cases involving transparency. In addition, BTF has support for a binary occupancy volume that increases rendering performance for cases that involve clipping, region growing, and segmentation. This volume renderer is extremely well suited for data sets where much of the data in the volume will be transparent (e.g. seismic interpretation and medical imaging).

The BTF renderer is a set of modules that are currently placed in the Unsupported library. It consists of the following modules:

btf shade
: Performs back-to-front (BTF) volume rendering. This module takes a volume, which can be visualized as a block of cubic "voxels" (volume elements), and generates a 2D image using a back-to-front (btf) direct volume rendering technique.

btf anim
: Performs back-to-front (BTF) volume rendering like **btf shade**

except that it allows the user to select a region of the volume and show intermediate rendering results, layer by layer to more clearly see the internal structure. Provides parameters to select starting and ending layers, number of steps between views, and run control.

btf bitvol

Creates a bit volume for use with **btf volume** renderer. This module generates a bit volume from a 3D scalar byte or 16-bit integer field using the opacity channel from a colormap. The rendering speed of the **btf shade** and **btf anim** modules can be increased substantially by using the **btf bitvol** module. Performance can be increased by a factor of 2-4 depending on the occupancy level of the volume, i.e. the number of non-zero voxels in the volume.

norm8 encode

Computes gradient vectors for 3D data sets and encodes them into bytes. The **norm8 encode** module computes the gradient vector at each point in a 3D field of data. The gradient vector can be used as a "pseudo surface normal" at each point. This vector is then encoded into a byte and packed into a 16-bit integer value along with the original input value.

norm8 table

Generates a lookup table for an encoded gradient. The **norm8 table** module accepts an optional transformation matrix as input. It builds a 256-entry intensity lookup table using the ambient and diffuse coefficients and the light source direction for all possible values of the encoded normal information (gradient vector). This table is then used during the shading process by the **btf shade** and **btf anim** modules to interpret the gradient information that was encoded into a byte by the **norm8 encode** module.

Using AVS/Voxel

On platforms that support AVS/Voxel, the BTF modules are located in the Unsupported library. There are several demonstration scripts that can be found in the *$AVS_PATH/demo/man_scripts* directory, all named with titles beginning with "BTF volume rendering". These are found by selecting the **Help** button and then selecting **Help Demos** to bring up the Script Controller which presents a list of demonstration scripts.

The BTF modules no longer require any additional licensing from Advanced Visual Systems. You may see a "AVS/Voxel" entry in your AVS5 license but this is no longer needed for AVS 5.5. You may wish to retain this feature line for older versions of AVS 5 still in use at your site.

Documentation

The man pages for the BTF modules are available online in the *$AVS_PATH/runtime/help/modules* directory along with the other AVS 5 modules. Press the module's dimple to get the Module Editor panel and then press "Show Documentation".

Demos

AVS5.5 NOTE: These demos have been rebuilt and retested under all AVS 5.5 platforms and problems have been addressed.

This release provides an optional package of AVS Demos that can be loaded from the CD after AVS has been installed. These demos can be used to see additional ways that AVS can be used and to obtain new demonstration datasets to augment those found in *$AVS_PATH/data*. Most of these datasets and demonstrations have been contributed by AVS users for the benefit of the AVS user community. While our users have permission to use them for demonstration purposes, they are not necessarily public domain and should not be used in products outside of AVS without express written permission from Advanced Visual Systems Inc.

Installing the Demos

You will require approximately 25 Megabytes to unload the Demos from the CD and another 5 to 10 Megabytes to build them for a total of 50 Megabytes. The demos are NOT precompiled but must be built in order to work.

1. Install AVS if you haven't already done so. The Demos will expect to reference a standard AVS product in $AVS_PATH (*/usr/avs* by default).

2. Install the **DEMOS** product from the AVS CD using the RPM (avs5demo_*-*.rpm) or the *install.avs* script used for installing the standard product. They will be installed into *avs_demos*.

3. If you are installing the demos from a remote login window you must make sure that your DISPLAY environment variable is set to an open display, e.g. "setenv DISPLAY :0.0". The installation procedure will attempt to run *avs* to run some data conversion scripts and this will fail if there is no DISPLAY set.

4. Change directory (*cd*) to the *avs_demos* directory and run the *install_demo* script found there to install a reference (link) from the *$AVS_PATH/demosuite* area to the place where the Demos are located. The *install_demo* script will do the following:

- Move aside the existing *$AVS_PATH/demosuite/demo_menu* file to *demo_menu.orig* and install a new version of the *demo_menu* file with the Demos scripts appended as an "Extended" menu.

- Create a link from *$AVS_PATH/demosuite/DEMOS* to the *avs_demos* directory. This link is required for making the Demos files since some of them expect to find *DEMOS* in *$AVS_PATH/demosuite*.

- Offer to "make" the Demos. You should reply "yes" to immediately make the modules and local data files that make up the Demos.

Uninstalling the Demos

Change directory (*cd*) to the *avs_demos* directory and run the script *uninstall_demo* which will do the following:

- Remove the Demos version of *$AVS_PATH/demosuite/demo_menu* and move the original *demo_menu.orig* back into place.

- Remove the link from *$AVS_PATH/demosuite/DEMOS* to the *avs_demos* directory.

- Offer to remove the entire current directory (assumed to be the *avs_demos* directory).

Running the Demos

Once installed, the Demos are accessible as a top level menu in the standard AVS demosuite. To run them, run *avs*, select **AVS Applications** and then select **AVS Demo**. The Demos should appear as a new menu called **Extended** and are organized by different market segments. Each demo consists of one or more demonstration scripts.

The following demos are part of the AVS Demos package:

Medical Imaging

Helen's Neck

MRI (Magnetic Resonance Imaging) study of Helen's neck. The data set consists of eleven slices of 256x256 images taken sagittally. On slice 5, you can clearly see her brain, spinal column, and the vertebrae surrounding the spinal cord. This is a real life example based on a study performed on a friend of Advanced Visual Systems Inc.

Brain

Stereotactic Radiosurgery: This demonstration shows the planned treatment for a real patient. The patient was a 2 year old girl with a brain tumor located dangerously close to her brain stem and inoperable by conventional surgical methods. Provided by Dr. Hanne Kooy at the Dana Farber Cancer Institute.

Geographic Information Systems (GIS)

World

This demo shows outlines for the following global features: continental outlines; islands; lakes; rivers; country borders; United States borders.

FlightPath

The **flight path** module takes a scatter field position list and animates the camera path along the trajectory. The modules is derived from *$AVS_PATH/examples/camera.c*. The scatter path field is in the form of "field 1D 3-space irregular float" where the coordinate information in the field specifies the path. The data values are ignored. It moves %top, not the camera, so multiple camera views can be set up to watch the path. The **file_descriptor** or **read_field** modules can be used to get the path from an external file. The **animated integer** module can be used to control time, for a moving sequence.

Terrain

Terrain Reconstruction from Digital Elevation data. This demo shows an elevation map and a landsat image (full color) of Orange County, with a perspective view. The image is rendered with a synthetic light source direction. The texture processing is performed by the AVS software renderer, or hardware adapters that support this operation.

Seal Tracks

Southern Elephant Seal movement and dive data in the South Atlantic. Provided by the SEA Mammal Research Unit, British Antartic Survey, NERC, Cambridge, UK. Ref: Dr. Ollie Cox and team.

Meteorology

Weather

Weather Satellite Images: If you have ever watched TV weather forecasts, you probably have seen the short "movies" of the clouds moving over the earth, especially when there is a hurricane off the coast. A new satellite photo is available every hour via the Internet. We have put together a series of 12 such photos from daytime

on July 21, 1992. The images register infrared light and therefore indicate temperature. These images were taken by the GOES-7 satellite and retrieved from a machine at University of Illinois at Urbana-Champaign, Illinois.

Oil and Gas

Velocity 3D

This is a demo of a synthetic 3D velocity profile field, with a "plumb line" interactive fence diagram picking module.

Intera

This is a series of representative models of oil and gas exploration projects produced by InteraView, a product of Intera Information Technologies Limited, Petroleum Production Division that uses AVS to deliver end user solutions.

Computational Fluid Dynamics (CFD)

Phoenics

PHOENICS fluid flow simulation, courtesy of CHAM (UK). The PHOENICS CFD code is interfaced to AVS for both pre-processing and post-processing. Includes Hot Electric Box, Smoke Spread in House (two versions), Cooling Tower and Turbine Blade.

Vortex

CFD RIBBONS - generate ribbon representation for streamlines. The **ribbons** module generates a set of geometric ribbons by taking the polyline output of the **streamlines** module and replacing them with finite width colored and textured polytriangle ribbons. The data set shows a 3D vortex field from NCSA researchers.

Mechanical

Automobile

Max vonMises Stress on a Mercedes: In this demonstration, NAS-TRAN was used to compute the Max vonMises stresses for a Mercedes-Benz automobile. This finite element dataset contains 15,843 cells. Provided by Keith Redner, Scientific Visualization Associates.

Crankshaft

Displacement on a Crankshaft: In these demonstrations, stress is applied to a crankshaft. This finite element dataset contains 609 cells. Provided courtesy of Pafec, Ltd.

Chemistry

Fast Animate

Water over Clay: Dr. Keith Refson, University of Oxford, UK uses a molecular dynamics program to simulate the interaction of water and cat ions with layers of clay. The simulation requires significant processing per time step, so it is run offline with the molecules' positions at each time step saved into files. This demonstration shows Dr. Refson's AVS animation module.

BGF Ribbon

The code in this directory is for the **PROTEIN** module, which reads a Biograf file and creates various displays of the molecule.

Statistics

City Scape

City Scape Data Analysis Display: Network Line Statistics are displayed using a 3D Bar-Chart technique, called "City Scape". The height and color of each cell shows two variables, while row and column averages are computed dynamically and shown as projected side panels. Provided by Advanced Visual Systems, Inc.

Jigsaw

The **jigsaw** module is a tool to show irregular boundary regions, defined in 2D, like map boundaries, and apply data to loft them into 3D with data defined height, and to apply color to indicate a second variable. Provided by Advanced Visual Systems, Inc.

Magnetics

Vector Fields

Vector Fields Magnetic Flux Modeling: This structure represents the result of an electro-magnetic flux analysis using "Tosca" from Vector Fields Ltd. The magnet core is shown as a ring structure, with air cells surrounding, with interpretive vector display modules. The model is axial-symmetric, so a geometry duplication method is used to construct the complete model for viewing. The Vector Fields interface, this network, and the visualization project is by Janet Haswell, Rutherford Appleton Labs, Oxfordshire, UK.

Texture

Texture Sampler

A set of texture image files such as rock, sand, and clouds from Evans and Sutherland.

DEBUGGING IN AVS 5.5

CHAPTER SIX

Introduction

This chapter is a summary of suggestions and tips on developing and debugging AVS modules; it draws together a number of techniques and features that are available in different areas of the product. More detailed documentation is referenced when available elsewhere.

AVS 5.5 NOTE: This entire chapter is new for AVS 5.5. It covers features documented in prior releases as well as new features or documentation in such areas as internal debugging flags, include files, and test data generator modules.

The information is organized into the following sections:

- Coding and Porting - general advice on approaching portability
- Building for Debugging - suggestions on making modules easier to debug
- Examples - borrowing code and techniques from module source code examples
- Command Line Interpreter - setting up reproducible tests and using debug flags
- Debugging Modules - using avs_dbx
- Input and Output Data - using standard test data sets during shakeout
- Kernel Debugging - seeing what is going on in the overall system
- Working with AVS Customer Support - preparing questions, options

Coding and Porting

Module Generator

If you are writing a new module, use the Module Generator module to create and manage the module throughout its life span. Described in Chapter 2 of the Application Guide, the Module Generator will not only create a source code "template" for your new module but it will read your module back in later and allow you to make changes as it evolves. It provides support for the following operations:

- Creating source code - you enter the desired name, properties, inputs, parameters, outputs, etc. and it generates the matching source code for either a subroutine or coroutine, creating the source file with appropriate header file includes, description and compute functions with proper declarations in C or FORTRAN, and initialization calls
- Creating a makefile - a working Makefile using the recommended Makeinclude definitions file is generated, supporting general portability
- Creating man page template - creates a sample man page to fill in
- Compiling the module - compiles the module for debugging or not as desired
- Loading the module - loads the module into the current Network Editor, current module library
- Debugging the module - runs the module in *avs_dbx*, the module debugger
- Reading existing module source - reads the module back in, interpreting the header information (inputs, outputs, parameters) and remembering user code between special markers to include in the new output.

Common issues

There are a few areas that frequently cause problems that you should be aware of.

Porting between 32 and 64 bit platforms

On 32 bit platforms, the integer (int) and pointer (int*, void*, etc) data types are both 32 bits and can be (and often are) treated as the same; poor programming practice has tolerated code in which these types are

cast back and forth to each other, or in which pointer values are accepted and stored as ints, etc.

Unfortunately, the SGI N64 and Dec Alpha platforms are 64 bit platforms that store ints as 32 bits, and pointers as 64 bits. This can result in bizarre values arising inside your 32-bit code when it is run on a 64-bit platform - pointer values can be sliced in half, sometimes resulting in extreme values or 0.

The bottom line is to carefully use ints and pointers correctly and look for problems like this if you are having porting problems.

FORTRAN and 64 bit platforms

When AVS 5 was originally developed, FORTRAN had no pointer variable capabilities. In order to pass pointers to fields and other data structures, FORTRAN functions would receive pointer values as INTEGERs, then pass these into C functions to do operations such as reallocating memory or computing offsets. In order to have FORTRAN code that runs on both 32 and 64 bit platforms, you need to change the declarations for values that receive pointer arguments from INTEGER to INTEGER*8. See "FORTRAN Modules on 64 bit Systems" in Chapter 3, "AVS 5.5 on UNIX Platforms", for more information on how best to do this.

XDR data format

Binary data file formats vary from one platform to the next and can cause serious problems between 32 and 64 bit platforms and also between big- and little-endian systems. Use "XDR" data format whenever possible to minimize these issues; see documentation on the Read Field and Write Field modules for more information.

Incorrect input and output declarations

It is easy to misdeclare the compute function arguments or not handle data allocation correctly, causing unexpected behavior downstream when the data isn't what is expected.

- Use the Module Generator and the "include hints" operation as much as possible; this will help reduce initial coding errors.

- Review the available AVS 5 example modules in *$AVS_PATH/examples* to find an example of data being passed or allocated similarly to what you need. The new *corout_f* FORTRAN coroutine example shows most data types being passed in and copied to outputs. The *widgets* examples are also good

samples of how to pass parameter values.

- FORTRAN string output requires that you allocate a string that will persist after you exit the compute routine, something that is difficult to do in FORTRAN. See the newly documented *AVSoutput_string* function in Chapter 3, "AVS 5.5 on UNIX" under the "New documentation" section for help in doing this.

- Use output diagnostic modules downstream to check that you are getting what you expect (see section below).

- Visit the IAC module repository (*http://www.iavsc.org*) for related module source.

- Request AVS 5 module source code for modules handling similar operations. For more information see "Working with AVS Support" below.

Performance - Reducing data copying and processes

AVS 5 uses a variety of techniques to minimize the number of data copies it needs, including shared memory and direct module communications. These work reasonably well but there are still times where AVS 5 will make extra copies of data, for example to send the data from module A (original copy), through the kernel (2nd copy), to module B downstream (3rd copy). This particularly affects user data types which do not use shared memory.

The best solution, when possible, is to combine modules into a single "mongo" module and clean them up to be "reentrant" and "cooperative" (see next issue about Global Data). When the modules are in the same process, they can pass data just by passing pointers rather than having to serialize the data and transmit it between processes, which takes time and makes extra copies. Cutting down on the number of module processes also saves memory usage and disk space, and reduces the number of processes at run time. See "Multiple Modules in a Single Process", page 4-21 in the *AVS Developer's Guide*, for more information on making modules behave well in shared processes.

You can combine your modules with AVS 5 standard modules as needed as well, because the module binaries are included in a series of libraries. See the *$AVS_PATH/examples/multi_hog.c* for an example of how to do this.

Global Data and AVSstatic

In order to make your module code "cooperative" and "reentrant" (allowing multiple module instances to share the same process), you need to be especially careful about using global variables (variables

declared outside a particular function). These are often used for remembering state information between compute function calls, but when more than one module instance is involved this "state" information may not be saved as originally expected.

The solution is to use the **AVSstatic** variable, documented in the section on Mutiple Modules (page 4-21 *AVS Developer's Guide*), mentioned above. You declare a local "state" data structure that can hold all the "global" state info for your module, make an instance of this data structure and store the pointer to it in AVSstatic. When your module is finished the value is stored away and then reset back when your module runs the next time. The AVSstatic variable is now declared in *$AVS_PATH/include/avs.h*; in releases prior to AVS 5.5 you will need to declare this variable locally as follows:

```
extern char *AVSstatic;
```

Port.h

An undocumented header file, *$AVS_PATH/include/port.h*, contains a number of platform specific compiler flags and declarations that permit AVS supported modules to work across machines. It provides flags to define characteristics such as endian-ness, BSD or not, data sizes of LONGS, etc. If you run into platform specific behavior, you may want to first review this file to see if there are appropriate declarations covering the areas of concern. Few changes have been made in this file over the last few releases.

Building for Debugging

When you write your module, it can be helpful to build in debugging aides that will produce diagnostic output when you want it. There are a few ways to do this.

AVSmessage

The AVSmessage function and its convenience forms (AVSinfo, AVSwarning, AVSerror, and AVSfatal) provide a means of sending multiple levels of information to dialog boxes, stdout, and a log file. Using AVSinfo may allow you to include debug output from your module. For more information on AVSmessage see "Handling Errors in Modules", page 3-13 in the *AVS Developer's Guide.*

The AVSmessage function has four levels of severity - (1) information, (2) warning, (3) error, and (4) fatal - corresponding to the four shortcut function calls. By default there is a "dialog cutoff level" of 2 - levels 2 through 4 will result in a dialog box being displayed with the message,

requiring the user to acknowledge before the module continues. Below this cutoff level (2), lower level messages (in this case just "(1)information") are displayed to standard output (stdout). All messages are recorded to a log file, called */tmp/avs.log_<ProcessId>* for the duration of the session. You can tell AVS to preserve this log file after the session is completed by using the *SaveMessageLog* option in your *.avsrc* file.

This "cutoff" level can be adjusted using the CLI "debug" command on the "AVSmessage_dialog" flag. Start up using "avs -cli" and then type "debug" to see or change the current debug flag settings. The level can be raised or lowered to suppress or enable dialog output above a certain level.

See the example *$AVS_PATH/examples/widgets* for an example of how AVSmessage can be used.

Invisible parameters

You can include extra debug control parameters in your module declaration that are not shown to users by default. Use the *AVSconnect_widget* function to select "none" (or clicking the option in the Module Generator) and no widget will be attached by default. Later when you wish to use this invisible parameter you can either use the CLI "parm_set" command for your module or bring up the Module Editor and from that the Parameter Editor to attach a widget to the parameter, making it accessible. One thing your parameter might control is additional debug output to stdout or AVSinformation().

Text browsers

The **print field** module shows an example of a module that writes text output to a file and then provides a window onto that file using a "text browser" widget. The parameter provides the name of a file that the module then writes to; when the module signals for a widget update, the widget reads in the text file to refresh its output. Though narrow by default, the module can select to make this window wider and parent it to the main shell window if desired.

The end user can do the same thing using the Layout Editor mode in the Network Editor, reparenting the module panel to the shell window, stretching the text browser, and reconfiguring the module panel. For an example of how to use text browser widgets this way see the new example module, *$AVS_PATH/examples/geom_write.c*

Memory leaks

A number of good commercial memory management packages, such as Purify from Rational Software, are available to help you track down memory leaks and mismanagement. AVS 5 also contains a less powerful memory management diagnostic package, invoked by including *$AVS_PATH/include/mem_defs.h* and defining MEM_DEFS_ENABLE during compilation.

This package redefines the front end to many string management and memory management functions and can provide verbose output. These macros insert a layer of error checking routines between the application and the system memory allocation routines. For instance if you call 'malloc', this substitutes a macro for the function call so it calls 'MEMmalloc' instead. It collects statistics, does error checking, and optionally prints out debugging information. These substitutions can cause some strange syntax errors under certain conditions. See "Memory Allocation Debugging", page 4-1 in the *AVS Developer's Guide* for more details.

AVS Examples

When you need module source code examples to see how more complex operations are done there are a number of sources.

- *$AVS_PATH/examples* - the first place to start for examples of C, FORTRAN, and C++ example modules. Several new module examples have been added in AVS 5.5; see Chapter 3, "AVS 5.5 on UNIX Platforms" under the "New Example modules" section.

- *AVS_DEMOS* - the extended demo package includes a number of customer module contributions which are compiled from source when the package is installed. See the chapter on "Extended Features" under the "Installing the Demos" and "Running the Demos" sections for more information.

- International AVS Center: The IAC has a large number of customer and AVS contributed source code modules. Visit their web site at www.iavsc.org. An older snapshot of the IAC web site is available on the "Cool CD" originally shipped with AVS 5.3 and all new shipments.

- AVS supported module source code: You can request the source code for most supported modules from AVS Customer Support after signing a source code agreement. Nearly all non-builtin modules are available. For more information see "Working with AVS Support" below.

Command Line Interpreter

The AVS Command Line Interpreter (CLI) can be useful for testing in the following ways:

- providing access to a number of debug switches for producing more verbose output
- timing how long it takes a particular command to execute
- recording test and demo scripts to test networks or reproduce problems more easily
- playing back test scripts providing either stepping or time delayed execution or early termination.

In order to use the CLI directly, just start "avs -cli" and the "avs>" prompt indicates AVS is listening for CLI commands. For more general information see Chapter 5 in the *AVS Developer's Guide.*

Debug switches

The "debug" command will either list or change a series of debug switches. The values shown are the default settings. A number of these are unsupported and incomplete but work well enough to be very useful.

```
debug  [UNSUP] Set or view internal debug switches
Without arguments will list and describe the known switches.
With just the name of the switch it will show the current value.
All switches take an integer value, with off being 0 in most cases.
     Usage: debug {<switch> <value>}
```

- AVScommand_debug 0 # Echo commands received through AVScommand and their results
- AVScomm_debug 0 # Enable module communications debug output (0 or non-zero)
- AVSfield_debug -1 # Enable field debug output levels -1, 0-4 increasing output
- AVSmessage_dialog 2 # AVSmessage level that presents dialog (0-4 = info-fatal)
- CLIrecord_LUI 0 # Enable recording of LUI button hits during script output (INCOMPLETE AND SUBJECT TO SEVERE LIMITATIONS)

- CLIscript_dialog 1 # AVSmessage (during scripts) present dialog (1)
- CLItime 0 # Display the time each CLI command takes to execute (1)
- EDITORmacro_mode 1 # System mode for implementing macros. 0 means old macros, 1 is new
- FlowConcurrent 0 # Turn on or off concurrent starting of networks with parallel forks
- FlowVerbose 0 # Make internal kernel operations report status information
- MEM_verbose 0 # Print a line for every allocate & free. 1 = within module compute function only; 2 = outside; 3 = Everywhere.
- MEM_check 0 # Fill memory on allocate or free. Also report leaks on second and subsequent module compute function calls.
- MEM_history 0 # If non-zero: Fences and history of allocations. More checking and better messages on free. 2 (bit 1) = check fence posts on every allocate and free. 4 (bit 2) = when exiting, list items allocated and never freed.
- XSynchronize 0 # Control X calls being flushed immediately (1) or not (0)

Some of the more interesting flags are AVScommand_debug, which shows what commands are coming in from modules using AVScommand; CLItime, which activates timing and reporting on individual CLI command executions; and FlowVerbose for gaining insight into what module is currently executing. The "MEM" flags will affect new module instances after they are set if the MEM package is compiled into those modules.

Writing scripts

CLI scripts are helpful for automated testing as well as reproducing specific problems. In order to write a script use the "script" command:

```
% avs -cli
avs> script -open <filename>.scr -echo yes
avs_script> ....  <do what you want>
avs_script> script -close
avs>
```

By saying "echo -yes" you will see the CLI commands being recorded going to *stdout* as well. This helps the user learn basic CLI commands and confirms what is and isn't being recorded. Most interactive operations will be recorded - Network Editor operations and module widget interactions in particular. Some additional operations will be recorded when you enable the debug flag, CLIrecord_LUI, by setting it to 1. Then when you record a script, more button hits and geometry viewer operations will be recorded. **Note:** The CLIrecord_LUI option is not supported as it is incomplete in some areas - direct interaction with the Geometry Viewer views is not recorded for example. However it can sometimes provide the "missing glue" that script writers need to perform a specific operation.

For more information on the CLI, see Chapter 5 of the *AVS Developer's Guide.*

Playing back scripts

Scripts can be played back in several ways - using the "avs -cli" command line option, the Network Editor/Help/Demos script browser, the AVS Demos pull down menus, or the "script -play" command as follows:

```
% avs -cli
avs> script -play <filename>.scr -echo yes -action user -break step
```

This last is the most useful and allows you to break at every command (*step*) or only at special points (*check*), either waiting for a user prompt or pausing for a specified amount of time. You can break at any time by hitting return in the CLI and it will ask if you wish to continue or abort the script.

In combination with other debug switches, you can slow down a prerecorded script and analyse the sequence of events taking place for each operation by using the "script -sleep" option. For more information type "help script" to the CLI to get the online help for this command.

Debugging Modules

The best way to debug modules is using the *avs_dbx* script that effectively wraps around a module and mediates with the debugger of your choice.

This tool is just a shell script that can be printed out or changed if necessary; by default it is set to use the most common debugger on a given platform. It can be run either with command line arguments to select a module or it will interactively query for the minimum information it requires. The steps for running it are as follows:

- Compile your module for debugging using either the Module Generator or by setting the "G" environment variable to the needed local debug flag and calling "make".
- Use the Module Generator "debug" option or invoke *avs_dbx <binary-name>*, selecting a specific module if needed with "-mod". If desired you can select a different debugger using the "-debug" option. Set any break points you need immediately.
- Instance the module in the Network Editor
- Type "run" AFTER the message "<module> instance waiting, fire when ready...".

More information on *avs_dbx* can be found in page 3-20, *AVS Developer's Guide.*

Input and Output Data

During development it can help to use test data patterns in addition to your own data sets. Test patterns can help confirm basic operations are working over a wide range of expected data. They can also help provide sample data sets showing problems with supported modules when working with AVS Customer Support.

Test input

Sample data sets can be either static - as found in *$AVS_PATH/data*, the AVS_DEMOS package, or the IAC - or dynamic, as output by test data generators like the new **test field** module (*$AVS_PATH/examples/test_field.c*) or the older FORTRAN examples, *test_field_f.f* or *test_fld2_f.f*. The new test field module provides a much more extensive set of test data patterns, dimensions, and sizes. It is described in the UNIX chapter and documented in its own man page.

Binary data file formats vary from one platform to the next and can cause serious problems between 32 and 64 bit platforms and also between big- and little-endian systems. Use "XDR" data format whenever possible to minimize these issues; see documentation on the Read Field and Write Field modules for more information.

Diagnostic output modules

Once you have run data through your module without crashing, the question is "how do you know it produced the RIGHT answer ?" There are several ways to see what came out. Note: for more information on these and any other modules, see the online man pages.

The most basic way is to see how the modules downstream respond to the output. If they crash or produce unexpected results, something is likely wrong with the data your module is producing. If you can debug the modules downstream you may see which aspect of your data sets are causing problems.

- **Print Field** - Some supported modules are provided that help you actually look at your data more directly. **Print field** in particular is a very useful tool for examining AVS fields. If attached to a field output, it will display information about the header and some or all of the data in a text browser widget (note: Use the Network Editor/Layout Editor to expand the default user interface layout for print field or use a separate text editor onto the same file name that print field is writing to. Print field allows you to take various text slices through the data by selecting different index ranges.

  What you are usually looking for using Print Field and similar modules is unusual data values, such as "NaN"s (not-a-number values) and extreme values with high exponents that are unexpected. Of course, any serious deviation from your expectations (all zeros, clipped values, etc) can give you a powerful clue to the problem.

- **Print UCD** - a similar module which outputs UCD data instead of fields.

- **Write Geom** - this new example module outputs binary data, calls *geom_to_text* to convert the data, and displays the result in a text window like print field. The source for Write Geom shows how to use the text browser widget to view a file.

- **Geometry Viewer** - The builtin Geometry Viewer module and the Geometry Viewer itself recognize a special environment variable, **AVS_GEOM_VERIFY**, as a request to print a verbose description of what they are seeing as they process a geom data type. This output will display NaN's and other oddities that may give some clue to the problem.

- **AVSGraph** - The AVSGraph module recognizes an environment variable, called **AGX_DEBUG**, to create a log file called */tmp/AVSGraph.log* of internal Toolmaster diagnostic messages. This may help you discern problems with input data or other problems.

- **Compare Field** - The compare field module serves a useful purpose when the data can be directly compared to an expected result or a similar baseline file. For example, in order to certify that similar C and FORTRAN modules produce the same output field, you would use compare field.

Kernel debugging

Debug output

Another source of problems relates more to how the system is transmitting the data or running the module. AVS 5 is capable of producing many different types of diagnostic output, though some of this information may be more useful for AVS Customer Support to help diagnose your problems than for you to make sense out of it without understanding the internal architecture.

The Network Editor is the best way to see the network and which module is currently executing. It provides Verbose mode under the "Module Tools" menu, which will cause diagnostic output about which module is executing and what connections are being made.

A related AVS command line option, "-mod_time", will time how long a given module takes to run when it is being used. Another way to see how long operations are taking is to use the CLI debug flag "CLItimer", which will time individual CLI commands and send the output to *stdout*.

A number of environment variables exist that produce some tracing information to describe communications or module execution. These are set using "setenv" in the C-shell.

- AVS_COMM_DEBUG - provides trace information about communications traffic between the AVS kernel and all other modules, including coroutines. This can highlight data transmission problems. Can also be set using the CLI debug command to set the AVScomm_debug flag.

- AVS_GEOM_VERIFY - mentioned already, as a flag to enable diagnostic output from the Geometry Viewer as it processes field data input.

- AVS_FIELD_DEBUG - provides AVS field data information as fields are allocated and deallocated in local or shared memory. This can provide some idea of the size of fields moving around as they are being created. Can also be set using the CLI debug command to set the AVSfield_debug flag.

- AGX_DEBUG - mentioned already, provides a diagnostic output log from the AVSGraph module.

Runtime conditions

AVS 5 provides a number of means of handling data such as shared memory (shm) and direct module communications (dmc) that expedite communications and reduce data copies, but that may obscure your problem. These communications options can be disabled using the "-noshm" and "-nodmc" options respectively.

The **AVS_OGL_INFO** environment variable is useful for determining information about your graphics adapter and OpenGL implementations. Modules may seem to produce different output when in fact the Geometry Viewer is rendering their output differently on different machines. Set this environment variable to 1 before starting AVS then enter the Geometry Viewer to get diagnostic output about OpenGL initialization.

Another option to consider when isolating problems is disabling modules selectively to pin point which module is the more likely "culprit". Use the Network Editor/Module Tools/Disable Module option or the CLI command, "module -off/-on" to disable individual modules.

Working with AVS support

Finally if you continue to have troubles with your module or application, and are currently on maintenance for AVS 5, you should contact AVS Customer Support for additional help or information.

The biggest problem we often face is being able to reproduce your problem. It helps immensely if you can demonstrate the problem using some of our supported modules and data sets. If not, the ideal is to provide us a copy of your module or application to review the situation you are facing.

Our FTP site (ftp.avs.com) provides the best means of sending large data sets or executables to us for help in investigating your problem and is much more reliable than email for large files. Please contact AVS Customer Support BEFORE you place files on our ftp server to ensure we are expecting your files and can pull them in quickly to attach to your case records. Follow the instructions to place them on the ftp site, in the *incoming* directory.

Less effective but still useful is seeing a Postscript snapshot of your network, obtained using the Network Editor/Network Tools/Print Network option or seeing output views from the system using *xv* or the *Write Image* module.

Requesting Module Source Code

In some situations, you may feel it would be most effective to obtain the source code to some of our supported modules, in order to make minor changes or investigate issues. In most cases it is possible for AVS Support to release the source code to individual AVS5 modules. This will be considered only for customers under a valid support contract.

Before the source can be sent we must receive or have on file a signed copy of our Confidential Disclosure Agreement. In addition a signed Source Code Addendum must be sent for each module or set of modules requested. Both of these documents can be downloaded from our web site at *http://help.avs.com/AVS5/faq/modsrc.asp*. The requests should be faxed to the AVS main office in the U.S. at (781) 890-8287 or through your local distributor or support office.

Once the request has been received, it will be reviewed for approval. Certain modules will not be made available. The following lists some of the modules not available as source code, though there may be more. Most of the following are "builtin" modules that only work as part of the kernel process anyway; these might be made available to Developer AVS customers that require them.

- geometry viewer
- image viewer
- graph viewer
- display image
- isosurface
- colormap manager
- display pixmap
- render geometry
- transform pixmap
- image manager
- render manager
- ucd legend