Placeholder

Contents

How I Do an AVS Kickstart ---------- 3

IAC Module Contest Winner ---------- 6

Crystal Lattice Viewer for AVS ---------- 12

Module Submission Criteria ---------- 15

AVS Support in the United Kingdom ---------- 16

Obtaining Modules ---------- 17

Using UCD for the Visualisation of Semiconductor Device Simulation Data -------- 18

AVS Courses Offered ---------- 22

Volunteers Needed ---------- 23

In Upcoming ***AVS Network News*** ---------- 23

In Touch with the IAC ---------- 23

AVS '93 Announcement ---------- 24

Errata

Please note the following corrections for Volume 1, Issue 3 of ***AVS Network News***.

Page 3 — Article authors should be listed as *Graham Walker and Steven Whalley, BT Laboratories and Ian Curington, Advanced Visual Systems Inc.*

Pages 3 and 4 — Figures 1 and 2 should be transposed.

Page 6 — Figure 1 should be rotated 90 degrees counterclockwise, and captions for Figures 2 and 3 should be transposed.

AVS Network News

The International AVS Center (IAC) serves as a catalyst for expanding the AVS user base and for increasing AVS functionality by fostering discipline-specific module development and new AVS uses. Located at the North Carolina Supercomputing Center, a division of MCNC, the worldwide clearinghouse collects, ports, and distributes user-contributed, public-domain modules and acts as liaison between users and vendors.

AVS Network News is the IAC's quarterly magazine serving users and vendors with updates on AVS programs and their use, users' color images and editorial viewpoints, and product information postings from vendors and independent software developers. For an annual subscription, send a $12 check or money order ($22 if outside the United States), payable to the International AVS Center, to Post Office Box 12889, 3021 Cornwallis Road, Research Triangle Park, NC 27709-2889.

David Bennett, AVS Project Leader, **avs@ncsc.org**
Katie Mohrfeld, AVS Specialist, **avs@ncsc.org**
Steve Thorpe, AVS Specialist, **avs@ncsc.org**
Dianne Reid, Database Specialist, **avs@ncsc.org**
Joel Page, Contributing Editor, **page@mcnc.org**
Fran Wise, Design Coordinator, **wise@mcnc.org**

For more information, contact
International AVS Center
Post Office Box 12889
3021 Cornwallis Road
Research Triangle Park, NC 27709-2889
Telephone 919-248-1100
Fax 919-248-1101

How I Do an AVS Kickstart

Alan Barnum Scrivener, Advanced Visual Systems Inc., Laguna Hills, CA, USA

I have found that an important part of the AVS environment is a persistent oral tradition. Users who have had to learn the product without ever seeing a demo or a video — or even just hearing an explanation of the whole point of the program — have had a harder time of it. Though there are of course exceptions, virtually all active communities of AVS users have had this oral tradition passed on to them, usually by a systems engineer like myself.

Many, many times since 1989, I have gone to visit groups of new users — ranging from one to 20 or so individuals — in order to deliver this oral tradition in what I called an "AVS kickstart" session. The format of the sessions has evolved during these four years into about a four-hour presentation — one hour on a white board, one hour of hands-on demonstration, and two hours of working with the customer's data.

In this column I will give a rough description of how this session goes, 1993 (AVS5) vintage. Of course every session is different, because the users interact with me and have different skill levels and project needs.

I always like to make sure that everyone present has seen either a demo or video of AVS in actual applications before this session. They are more motivated to master the details of AVS if they've seen what results it can deliver to them.

I begin in a conference room with a white board. While waiting for everyone to arrive, I draw diagrams of 2-D and 3-D uniform, rectilinear, and irregular field grids on the white board. (The 3-D diagrams look very much like volume bounds of the standard AVS hydrogen.fld, rectilinear.fld, and bluntfin.fld field files in /usr/avs/data/field, with only a few of the grid lines drawn.)

When everyone has arrived, I introduce myself and then go around the room asking each person to say who they are, what kind of things they want to do with AVS, and what they want to learn today.

I begin by suggesting that AVS can be learned sensibly in the same order that it evolved. What we now call the **Geometry Viewer** began as AVS1, so I cover it first.

AVS1

The **Geometry Viewer** is a user interface for rendering views of these 3-D geometric primitives — points, lines, polygons, and spheres.

In the **Geometry Viewer**, you can assemble aggregates of primitives, which we call geometries. You can combine geometries with orientation, color, and property information to form what we call objects. You then can create window views of one or more objects under a set of lighting conditions, which we call cameras, and then combine the cameras into what we call scenes. This whole process usually is called rendering.

The fastest way to learn the **Geometry Viewer** is to read the chapter in the AVS tutorial manual that covers it. The most important thing to learn is the mouse button bindings in geometry windows.

Right now I want to cover how one reads data into the **Geometry Viewer**. The **Geometry Viewer** intrinsically knows how to read three types of files: geometry files, which end with a .geom suffix (e.g., jet.geom); object files, which used to end in a .obj suffix but are now CLI script files and so end in .scr (e.g., math.scr); and scene files, which used to end in .scene but also are now CLI script files and so end in .scr (e.g., molecule.scr).

Currently you can read the old suffixes for backward compatibility, but you can't write them anymore from the menus. Anyway, what happens if you click on a file with a suffix AVS doesn't intrinsically recognize? Let's say you select "saddle.mesh" (which I'll tell you how to find momentarily). If there was an executable at AVS startup in /usr/avs/bin called **mesh_to_geom**, then AVS would run that program with "saddle.mesh" coming in to its standard input, divert its standard output to "saddle.geom" on disk, and lastly read the new.geom file into the **Geometry Viewer**.

The source code to this “geom filter” and others is in the directory /usr/avs/filter. For example, the executable **mesh_to_geom** is compiled from source file “mesh.c” in the filter directory. A makefile there compiles all of the filters. Note that these are UNIX filters in the classic Kernigan & Plaugher “Software Tools” sense of the word. (Later, there will be AVS filter modules that will be something entirely different.)

In the subdirectory /usr/avs/filter/example are example input data files (ASCII text) that the geom filters can convert to geom files. This is where “saddle.mesh” lives. There are a couple of these formats I’d like to point out; most of them are documented in their respective source code files in the comments. For example, the following is from /usr/avs/filter/polygon.c.

```
* The format is
*
*  <type> (either “facet” or “smooth”)
*  n_verts_polygon_1(decimal)
*  vert[1]X(float) vert[1]Y(float) vert[1]Z(float)
*  ...
*  n_verts_polygon_2(decimal)
*  ..
*  <EOF>
```

And /usr/avs/filter/example/cube.polygon is an example data file in that format. But one filter, **sphere_to_geom**, has no example data file. Here is a simple one.

```
::::::::::::
test.sphere
::::::::::::
-1 -1 -1
1  1  1
3
1.0  0.0  0.0  0.3  1.0 0.0 0.0
0.0  1.0  0.0  0.3  0.0 1.0 0.0
0.0  0.0  1.0  0.3  0.0 0.0 1.0
```

The first two lines are extent information that can be used to auto_normalize the data. I recommend you always use -1 -1 -1 1 1 1 here, so you don’t auto_normalize. That way, the sphere geometries will end up in the same coordinate system as all other geometries you read in from files or create by using AVS modules.

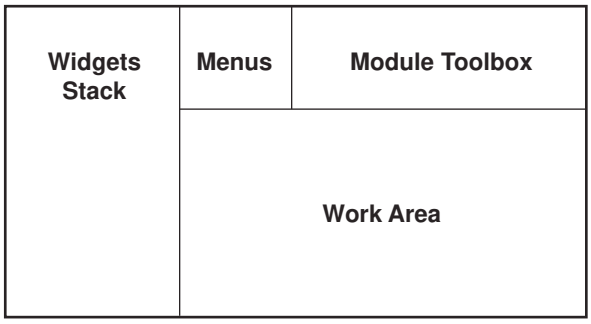
Next is the number of spheres (three); then for each sphere, the center X, Y, and Z coordinates, the radius, and the color as RGB values from zero to one. In this example, a red sphere is out the positive X axis, a green sphere is out the positive Y axis, and a blue sphere is out the positive Z axis.

These geom filters are handy ways to create static geometries quickly. The source code also will teach you about the AVS GEOM function calls, for which you also need to write AVS mapper modules.

Another handy thing to do is to create sequences of geometries to cycle through over time as you render. An example file is in /usr/avs/data/geometry/math.scr, which points to a sequence of .geom files, in this case successive phases of a sine-cosine surface generated by Mathematica. If you read in such an action sequence file, you then can use the **Geometry Viewer** action menu to turn on automatically cycling through the individual geometries even as you move them around with the mouse.

AVS2

With AVS2 came the Network Editor. This is what we call the “under the hood” controls for AVS. The Network Editor window has this layout.



To build AVS applications, drag modules into the work area from the toolbox and then connect them with the mouse to form a data flow diagram, or what we call a network. There are more mouse bindings to learn for these operations.

The Network Editor recognizes a number of different data types, and you also can define your own. But the most fundamental and important data type in AVS is what we call a

"field," which is a multidimensional array of numbers with spatial meaning. When you define a field to AVS, you answer several questions. What is the dimensionality of the array? What are its extents? What is the dimensionality of the space it represents or in which it's imbedded? Is it scalar or vector data? What numeric type is it — byte, integer, or floating point? And lastly, what kind of grid is it?

A uniform grid has uniformly spaced data elements in each dimension, with all grid intersections perpendicular. This means the grid is made entirely of squares in 2-D or of cubes in 3-D. When you define this type of field, you usually just provide the data values and let AVS generate the grid for you.

A rectilinear grid still has perpendicular grid intersections, but they may be unevenly spaced. This means the grid is made entirely of rectangles in 2-D or of rectilinear solids in 3-D. When you define this type of field, you provide the data values and then the coordinate spacing, almost like "hash marks" along each axis, and AVS then generates the grid from this information.

An irregular grid, often called curvilinear, has the perpendicularity constraint lifted. Topologically, it must be possible to transform the grid back to a uniform lattice, such as a square lattice in 2-D or cube lattice in 3-D. (I'll use the 3-D case only for simplicity.) In other words, each grid point in 3-D must have six neighbors, which we usually think of as front, back, left, right, top, and bottom in a cube lattice. In an irregular grid, these six neighbors can be severely twisted and stretched. Still, while the grid can wander all over the place (for example, in a cylindrical or spherical coordinate system), each grid point can be found through indexing in with the three values I, J, and K. When you define this type of field, you provide the data values to AVS as well as explicit X, Y, and Z coordinate values for each grid point.

I've found the biggest challenge to using any visualization software is getting your data into it. In AVS, the ease of doing this has evolved over the years. In AVS2, the thinking of the developers was that you could write your own data reader module for whatever data you had, so you were required to be a C or FORTRAN programmer. The developers added a built-in **Read Field** module as an afterthought, and it only read AVS native field files. Since these were binary files that only a program could create, you still were required to be a C or FORTRAN programmer. (These native field files end with a .fld suffix, and you can learn about the file format in the man page for **Read Field**, available online or in the AVS module reference manual.)

In AVS3, the **Read Field** module was enhanced to read text headers that described where to find the data and what format it was in. All you had to was be able to use a text editor on your system. (These headers also end in a .fld suffix and are documented in the **Read Field** man page.) Well, it turned out a lot of users really couldn't use a text editor either. So in AVS4, we added the **Data Interchange Application**, and now you can read data in by filling out a form on-screen entirely with the mouse. (This is documented in the *AVS Application Guide*.) The remaining constraint is that you still have to know how your data is structured. If you know what the bits mean, AVS can read it.

*[Look for part 2 of this article in an upcoming **AVS Network News**.]*

IAC Module Contest Winner

Katie Mohrfeld, International AVS Center, Research Triangle Park, NC, USA

The International AVS Center (IAC) is pleased to announce Jonathan Cox, from the University of Manchester, as the winner of 1992's fourth-quarter module contest. He submitted two modules, **ucd_line** and **ucd_planes**, each among the top 10 most popular modules submitted during the fourth quarter. Cox also has written an article for ***AVS Network News*** and will be presenting a paper at AVS '93.

We also would like to congratulate Wes Bethel, from Lawrence Berkeley Laboratories, who was selected as the winner of the third-quarter module contest. He has contributed numerous modules that are very useful to the AVS community.

The IAC looks forward to selecting the winner of next quarter's module contest. The author of the best module(s) contributed in a quarter will receive his/her choice of $250 cash (U.S. funds) or $250 credit towards attendance at AVS '93. Each module author is evaluated according to multiple criteria, including module utility, uniqueness, and popularity.

The IAC makes the final decision as to who wins the award. Normally, submissions for each quarter must be received before midnight on the final day (EST) of the quarter. However, the deadline for second quarter is May 1 (rather than June 30), so that the awards can be presented at AVS '93.

Staff from IAC and MCNC (where IAC is located) and their families are not eligible for the contest. Employees and their families from Advanced Visual Systems Inc. also are not eligible for the contest. All other entries will be equally considered.

New Modules Submitted Last Quarter of 1992
Thanks from the IAC and the AVS user community for the following 33 modules submitted during the fourth quarter. We now have more than 600 modules at the ftp site.

Name:	**ColorToBW**
Version:	1.000
Module Number:	1599
Author:	Steve Thorpe, International AVS Center
Description:	This module converts an image from color to black and white, keeping the alpha channel the same in the resulting image.

Name:	**EditColors**
Version:	1.000
Module Number:	1547
Author:	D. Le Corre, CONVEX S.A.
Description:	This filter module allows you to define interactively a color map to be used with an input field. Once the number of subdivisions is set, you can modify individually the size of each slot and the associated color.

Name:	**Field_Formula**
Version:	1.000
Module Number:	1545
Author:	Ronald E. Stogner, Computer Sciences Corporation
Description:	The **Field_Formula** module accepts a reverse polish notation (RPN) formula, specified by the input string. This formula then is applied to all elements of the field, and a field of the same dimension is output with a vector length of one. This field contains the result of the applied formula for all locations in the field. "Variables" are defined by their field labels and are delimited by "[]"'s.

Name: **Input_Formula**
Version: 1.000
Module Number: 1546
Author: Ronald E. Stogner, Computer Sciences Corporation
Description: The **Input_Formula** module accepts an input field and uses labels in the field as variables or “species” in an algebraic expression. The user can select various parts of the expression from the available widgets and build a formula interactively. The module parses this expression and outputs an RPN string expected by **Input_Formula** when the user selects the “Parse Formula” widget.

Name: **Read_Unversal_FI**
Version: 1.000
Module Number: 1562
Author: Digital Equipment Corporation
Description: **Read_Unversal_FI** is a utility module for conversion of I-DEAS Universal Files into AVS UCD data. This module only extracts Stress data and node-element information.

Name: **Sin_Wave**
Version: 1.000
Module Number: 1600
Author: Steve Thorpe, International AVS Center
Description: This little module was written while playing with the graph viewer. A simple module to spit out a 1-D field was desired, and this did the trick. It computes a sin wave over the range 1..npoint degrees and spits out a “field 1-D 1-space 1-vector uniform float” suitable for display with the graph viewer module. This may be a useful starter module if you want to learn how to get your 1-D data into AVS.

Name: **Vector_Label**
Version: 1.000
Module Number: 1563
Author: Charles J Williams III, Naval Research Laboratory
Description: This module produces labels that are geometries. They can be scaled, rotated, and translated. Other geometric operators work as well, such as color, perspective, and material properties. The source for the vector information for the fonts is derived from the Hershey Fonts, stored in **comp.sources.unix**, volume 4, on your nearest archive site. The Hershey Fonts originally were created by Dr. A.V. Hershey while working at the U.S. National Bureau of Standards. The format of the font data in the Internet distribution originally was created by James Hurt at Cognition Inc.

Name: **combine_vectors**
Version: 1.000
Module Number: 1554
Author: Earl W. Hood, CONVEX Computer Corporation
Description: The **combine_vectors** module combines up to four vector and/or scalar fields into one vector field. The input ports of the module are processed from right-to-left. The resulting output field will have a vector length equal to the sum of all the vector lengths of the input field. Any labels and units will be carried over to the output field. The module was written to avoid using numerous **extract_scalar** modules to process a vector component separately and then using the **combine_scalars** module. (e.g., process the R,G,B components of an image together, and then combine the R,G,B components back with the Alpha channel).

Name: **complex_math**
Version: 1.000
Module Number: 1558
Author: Earl W. Hood, CONVEX Computer Corporation
Description: The **complex_math** module performs binary math operations on two-vector fields that represent complex numbers. The first vector component represents the real part of the complex number. The second vector component represents the imaginary part of the complex number.

Name: **exponential**
Version: 1.000
Module Number: 1559
Author: Earl W. Hood, CONVEX Computer Corporation
Description: The exponential filter module computes ‘e’ raised to each field value (i.e., newvalue = e^value).

Name: **extrude_field**
Version: 1.000
Module Number: 1555
Author: Earl W. Hood, CONVEX Computer Corporation
Description: **extrude_field** generates a 3-D volume by applying a transformation matrix defined by the parameters to a 2-D field (i.e., it "sweeps" a 2-D field through 3-space to generate a 3-D [irregular] field. Some researchers had datasets that represented a slice of something axisymmetric in 3-space. With **extrude_field**, researchers can recreate the 3-D representation. This is also good for researchers who have 2-D datasets, but the modules that they would like to use only accept 3-D datasets.

Name: **field_averaging**
Version: 1.000
Module Number: 1556
Author: Earl W. Hood, CONVEX Computer Corporation
Description: The **field_averaging** module computes the average field of a series of multiple fields. The module is useful if there exists multiple fields (e.g., images) that represent a series of samplings of a given object, and an average of the samples will help cancel out the noisy data contained in each sample.

Name: **field_lines**
Version: 1.000
Module Number: 1533
Author: Rob Kuyper, University of Technology Delft, The Netherlands
Description: **field_lines** is a mapper module that converts a 2-D field in an object of lines and planes. This module creates black lines (in optionally i- or j-direction) with a "no-lighting" plane below it, so the field will look like a series of nontransparent graphs.

Name: **field2_from_Math**
Version: 1.000
Module Number: 1542
Author: Oleg Perelet, Wolfram Research
Description: The **field2_from_Math** module allows a 2-D scalar field to be supplied to AVS. When the **field2_from_Math** module is invoked, an xterm is started, running Mathematica with the input and output of the xterm directed to and from the Mathematica session main loop. Entering Mathematica commands into the window will effect the appropriate Mathematica results. The session is initialized with a package of Mathematica commands that have started a MathLink communication channel to AVS. Other initializations are made so that the Mathematica command AVSWriteField[data] will write a 3-D scalar field to AVS. Presently, the IAC does not have Mathematica installed on a DEC, Sun, or Kubota platform. As a result, this module was placed on the ftp site, without testing, using the Makefiles provided by the author.

Name: **field2_to_Math**
Version: 1.000
Module Number: 1539
Author: Tom Wickham-Jones, Wolfram Research
Description: The **field2_to_Math** module allows a 2-D scalar field to be imported into Mathematica from AVS. When the **field2_to_Math** module is invoked, an xterm is started, running Mathematica with the input and output of the xterm directed to and from the Mathematica session main loop. Entering Mathematica commands into the window will effect the appropriate Mathematica results. The session is initialized with a package of Mathematica commands that have started a MathLink communication channel to AVS. Other initializations are made so that the Mathematica command AVSReadField[ ] will read a 2-D scalar field from AVS. Presently, the IAC does not have Mathematica installed on a DEC, Sun, or Kubota platform. As a result, this module was placed on the ftp site, without testing, using the Makefiles provided by the author.

Name: **field3_from_Math**
Version: 1.000
Module Number: 1540
Author: Oleg Perelet, Wolfram Research
Description: The **field3_from_Math** module allows a 3-D scalar field to be supplied to AVS. When the **field3_from_**M**ath** module is invoked, an xterm is started, running Mathematica with the input and output of the xterm directed to and from the Mathematica session main loop. Entering Mathematica commands into the window will effect the appropriate Mathematica results. The session is initialized with a package of Mathematica commands that have started a MathLink communication channel to AVS. Other initializations are made so that the Mathematica command AVSWriteField[data] will write a 3-D scalar field to AVS. Presently, the IAC does not have Mathematica installed on a DEC, Sun, or Kubota platform. As a result, this module was placed on the ftp site, without testing, using the Makefiles provided by the author.

Name: **fourier_filter**
Version: 1.000
Module Number: 1548
Author: Earl W. Hood and Allen Schiano, CONVEX Computer Corporation and University of California at Irvine
Description: The **fourier_filter** module takes a 1-3D input field, filters the field in the frequency-domain, and outputs the processed field. The module is designed specifically for image and volume enhancement.

Name: **fourier_transfor**
Version: 1.000
Module Number: 1549
Author: Earl W. Hood and Allen Schiano, CONVEX Computer Corporation and University of California at Irvine
Description: **fourier_transfor** computes the Fourier transform of a field. If the input field is scalar, the module assumes that the imaginary component of the Fourier transform is zero. If the field is 2-vector, the first component is the real part, and the second component is the imaginary part. In either case, the output field is 2-vector with the real part in the first component and the imaginary part in the second component of the vector. The module uses Fast Fourier Transform algorithms to compute the desired Fourier transform. Therefore, the output field might have certain dimensions expanded as a result of the algorithms.

Name: **gamma_correct**
Version: 1.000
Module Number: 1550
Author: Wes Bethel, Lawrence Berkeley Laboratory
Description: The **gamma_correct** module performs gamma correction on an AVS image. The user also may control the amount of correction to apply. The user may choose how to correct the image, either in a manner that will result in a hue shift in the image or so as to preserve hues.

Name: **geom_from_Math**
Version: 1.000
Module Number: 1541
Author: Oleg Perelet, Wolfram Research
Description: The **geom_from_Math** module allows 3-D graphics commands in Mathematica to supply AVS with an input geometry. When the **geom_from_Math** module is invoked, an xterm is started that runs Mathematica with the input and output of the xterm directed to and from the Mathematica session main loop. Entering Mathematica commands into the window results in the appropriate Mathematica results. The session is initialized with a package of Mathematica code that starts a MathLink communication channel to AVS. Other initializations are made so that the typical Mathematica 3-D graphics commands, such as Plot3D or ParametricPlot3D, send their results into AVS rather than rendering them in the more typical Mathematica ways. At the present time, the IAC does not have Mathematica installed on a DEC, Sun, or Kubota platform. As a result, this module is placed on the ftp site using the makefiles provided by the author without testing.

Name: **luminence**
Version: 1.000
Module Number: 1557
Author: Earl W. Hood, CONVEX Computer Corporation
Description: **luminence** computes the luminence of an image. However, it differs from the standard AVS luminence module in that one can control how much red, green, and blue contribute to the brightness of an image. This functionality is useful when using the module with the **replace_alpha** and **composite** modules (e.g., making any color with red fully opaque while leaving other colors transparent).

Name: **natural_log**
Version: 1.000
Module Number: 1560
Author: Earl W. Hood, CONVEX Computer Corporation
Description: The **natural_log** filter module computes log base e of a real field (i.e., newvalue = LOG e [value]).

Name: **pdb_to_geom**
Version: 1.000
Module Number: 1551
Author: Advanced Visual Systems Inc.
Description: The **pdb_to_geom** module reads the description of a molecule from a file in the Brookhaven Protein Data Bank data format. Typically, such files have a filename suffix. The output is an AVS description of the molecule.

Name: **read_AIFF**
Version: 1.000
Module Number: 1543
Author: Charles J Williams III, Naval Research Laboratory
Description: This module is meant to be a companion to the **Write_MooV** module. It operates rather simply, reading in an AIFF file (this is a rather Macintosh format, although many programs can translate and understand AIFF) and generating a 1-D field.

Name: **read_F3D**
Version: 1.000
Module Number: 1537
Author: Michael Rangitsch, Dow Chemical
Description: This module reads an ASCII formatted multiblock FLOW3D solution file (from Harwell FLOW3D version 3) and stores the cfd data as a ucd dataset. The dummy nodes stored at the edges of each computational block are removed by the module. The n-vector data at each computational cell is stored as cell data to use most of the ucd modules; this must be converted to nodal data. The cells of the ucd structure are all hexahedra. See the sample data directory on the IAC's anonymous ftp site for two sample datasets to read with this module.

Name: **rgbtogray**
Version: 1.000
Module Number: 1536
Author: Wes Bethel, Lawrence Berkeley Laboratory
Description: The **rgbtogray** module receives an AVS image as input, then computes an output field that is a gray scale version of the rgb components of the input image.

Name: **string_example**
Version: 1.000
Module Number: 1553
Author : Larry Gelberg, Advanced Visual Systems Inc.
Description: The **string_example** module sends a single user-specified filename string to one or more string parameter ports on one or more receiving modules. Its purpose is to allow you to control simultaneously filename parameter input to more than one module using only a single file browser input widget. This code was based on the AVS **file browser** module.

Name: **string_to_file**
Version: 1.000
Module Number: 1561
Author: Ian Curington, Advanced Visual Systems Inc.
Description: This module sends the contents of the string input port to an output file, opened with a file browser. The string can replace the current contents of the file or be appended to the file, opened during the session, as a log of strings in the network controlled by a mode toggle. This is designed to send command strings via a named pipe (see mknod pipe.file p) to another application program, where command input has been redirected to the same pipe. This allows an AVS user interface module to control an external command-line driven application.

Name: **table_editor**
Version: 1.000
Module Number: 1538
Author: Ian Curington, Advanced Visual Systems Inc.
Description: Like a mini-spreadsheet application, this module presents a segment of a 2-D array and allows interactive editing of the field values. It takes an optional input field and displays the contents as a 2-D array table of button cells. Each cell or group of cells may be selected (highlighted) and new values entered. The modified field then is sent to the output port. Values can be replaced, added, multiplied, or divided into existing field values. The code shows how to create a page layout from within the module code. With no input port, this module can be used to synthesize new fields. If the input field is multivector, each vector can be selected from a choice list. Values can be edited in each channel, while synthetic data can be created only in scalar mode.

Name: **ucd_extract_new**
Version: 1.000
Module Number: 1564
Author: David Bock, North Carolina Supercomputing Center
Description: The **ucd_extract_new** module is a simple extension to the original ucd extract module. This module uses an integer instead of a string for one of its parameters. The module can be used in conjunction with animate integer to read and visualize a ucd dataset across time. By including every timestep of ucd data at the nodes, the animate integer can be connected to the integer parameter on this module to select and read each timestep from memory in succession.

Name: **ucd_line**
Version: 1.000
Module Number: 1535
Author: Jonathan Cox, Department of Computer Science, University of Manchester
Description: This module produces an arbitrary line of sampled node data from a 3-D UCD mesh and outputs it as a field. This can be a conventional AVS 1-D field with irregular 1-D coordinates or a 2-D field suitable for connection to the **graph viewer** module. It is also possible to output a uniformly spaced 1-D field.

Name: **ucd_planes**
Version: 1.000
Module Number: 1534
Author: Jonathan Cox, Department of Computer Science, University of Manchester
Description: The **ucd_planes** module supports the interactive subsampling of node data held within a 3-D UCD structure. The region sampled is selected interactively in the region of a transparent “probe” object. Several different types of probes, all based upon a collection of planes, are supported.

Name: **write_MooV**
Version: 1.000
Module Number: 1544
Author: Charles J Williams III, Naval Research Laboratory
Description: This module is a first shot at incorporating QuickTime™ into AVS as a method for producing convenient, compact, and high-quality video segments. QuickTime™ is rather convenient since more support is likely on other systems, and a growing base of QuickTime™ is cheaply available for the Macintosh. This code currently requires the QuickTime™ Movie Exchange Toolkit, which is available from Apple Inc., and a C++ compiler (you also need the C++ compiler to build the QuickTime™ libraries). Since the IAC doesn’t have the Exchange Toolkit, this module hasn’t been tested.

Crystal Lattice Viewer for AVS

Tim Robinson and Fred Hollander, College of Chemistry, University of California at Berkeley, Berkeley, CA, USA

Introduction

AVS provides a straightforward environment for prototyping applications for image processing and interactive 3-D visualization. Often, the prototype applications are completely suitable for end-use. This article discusses one of the applications that we have constructed in the AVS environment.

We present here the design considerations for an instructional tool suitable for use in an introductory course in X-ray crystallography. The current version of the CRYSTAL LATTICE VIEWER allows students to visualize crystal lattices in real space and corresponding diffraction data in “reciprocal space.”

This article is targeted for the entry level AVS developer and the general AVS user community. It contains a section on AVS tips and tricks encountered in the development of this application.

Background

Chemists trying to determine the structure of a novel compound often grow small crystals and then hand them over to a crystallographer for analysis. Crystals consist of ordered arrays of molecules that can be characterized by repeating identical “blocks” called unit cells, which pack together to fill space. The corners of the unit cells lie upon points of a (usually) non-orthogonal 3-D space called the real space lattice.

The symmetry and nature of the molecular arrangement within the unit cell determines the space group of the crystal and the symmetry of the real space lattice. The physics of X-ray diffraction experiment is such that the data sampling is much more conveniently described as taking place in another 3-D space called the reciprocal lattice.

The real space lattice is described by six parameters, the lengths of the three edge vectors of the unit cell, and the three interaxial angles between these vectors. By convention, the lengths are called a, b, c; the angles are called α, β, and γ.

The angle α is the angle between the vectors **b** and **c**, opposite vector **a**. β is opposite **b**, and γ is opposite **c**. The cubic lattice is an orthogonal basis representation with a=b=c and $\alpha=\beta=\gamma=90$ degrees. Crystal lattices exist in six other discrete forms.

The reciprocal space lattice vectors **a***, **b***, and **c*** are related to the real space lattice by relationships of the form

$$\mathbf{a}^* = (\mathbf{b} \times \mathbf{c}) / V$$

where V is the volume of the unit cell. For all orthogonal lattices, the expressions simplify greatly. With the other lattice types, the

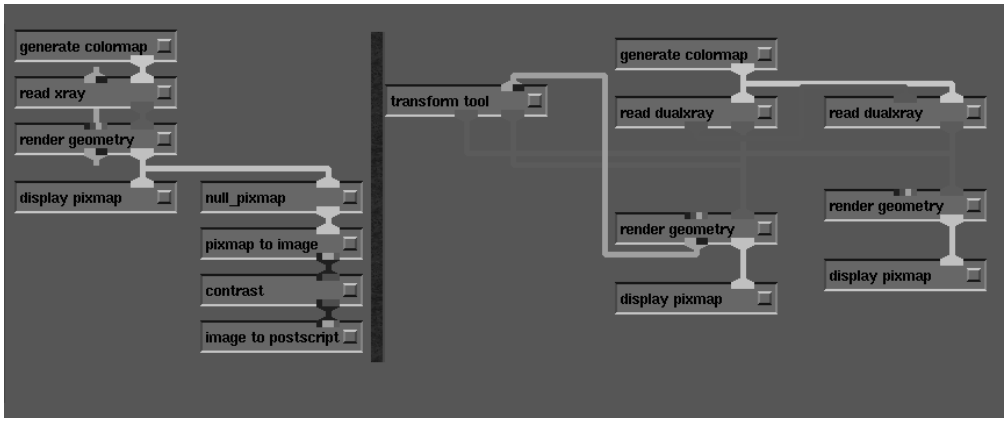


Figure 1. *The two sample networks for distribution as part of the CRYSTAL LATTICE VIEWER. The network on the left can be used to visualize a dataset with non-orthogonal basis. The right-hand side of this network is disabled until nu**ll_pixmap** is toggled. This provides an easy mechanism for users to get postscript out without having to reconfigure the network. The network on the right simultaneously displays two matching datasets. Transformations applied to the left-hand re**nder geometry** module are copied over to the other **render geometry** module by the **transform tool** module.*

relationship between the real and reciprocal lattice is not as trivial.

The objective of this application is to provide the framework needed to show students

- clear models of the lattice points in these various crystal systems,
- the spatial relationships between real and reciprocal spaces, and
- the influence of crystal symmetry in reciprocal space data.

Implementation

The CRYSTAL LATTICE VIEWER currently is implemented in two sample networks. Only four modules other than the standard AVS-supported module library are used — two reader modules with very similar functionality and two modules that are simple utilities, discussed further in the following tips section. This application required some fairly thorough user interface manipulation, all accomplished through the Network Editor's Layout Editor.

One network (left side of Figure 1) lets students visualize a single dataset. Either a real space lattice or a reciprocal space data sample can be viewed, because the only difference in these data files is a flag in one of the records in the file. Dial widgets allow the user to adjust the number of displayed data points by cropping along the axial directions.

In this network, the students can reproduce experimental diffraction photographs by orienting a plane of interest perpendicular to the screen and using a cropping plane to isolate a particular layer (see Figure 2).

The other network (right side of Figure 1) is designed to let students simultaneously visualize a real space lattice and a simulated diffraction dataset for that lattice in reciprocal space. The side-by-side display is accomplished with two **render geometry** modules connected to two independent **display pixmap** modules (see Figure 3).

Transformations in the left-hand window are grabbed and applied to the right-hand window by a utility module discussed below. The students also can move the data in the windows independently by transforming the geometry in the right window. When the correct spatial orientation again is required between the data in the two windows, a small

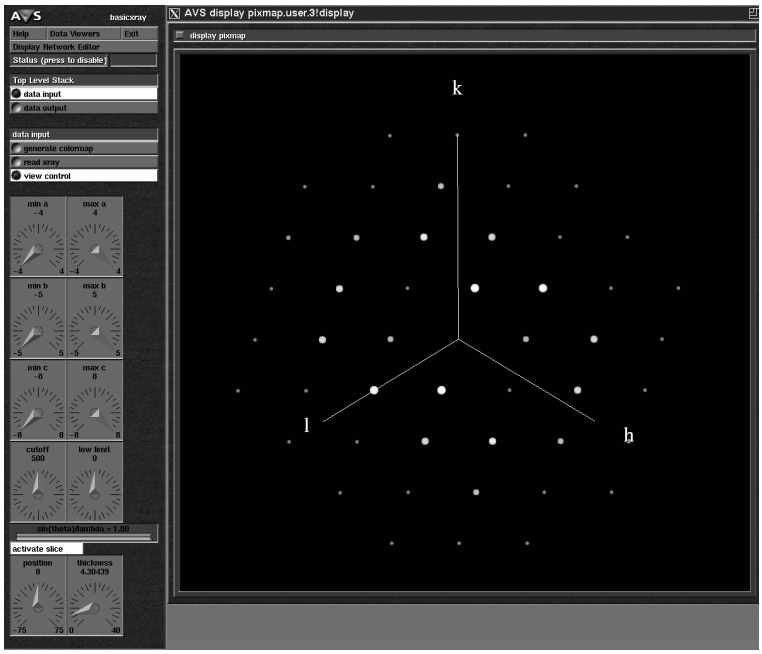


Figure 2. *A simulated precession photograph for a diffraction dataset. The dataset simulation is based on a crystal in the monoclinic crystal system (a = 8.5, b = 10.2, c = 17.1, $\alpha = 90$, $\beta = 107$, $\gamma = 90$). On the left, the user controls for viewing are displayed. All layout work was done with only the Layout Editor.*

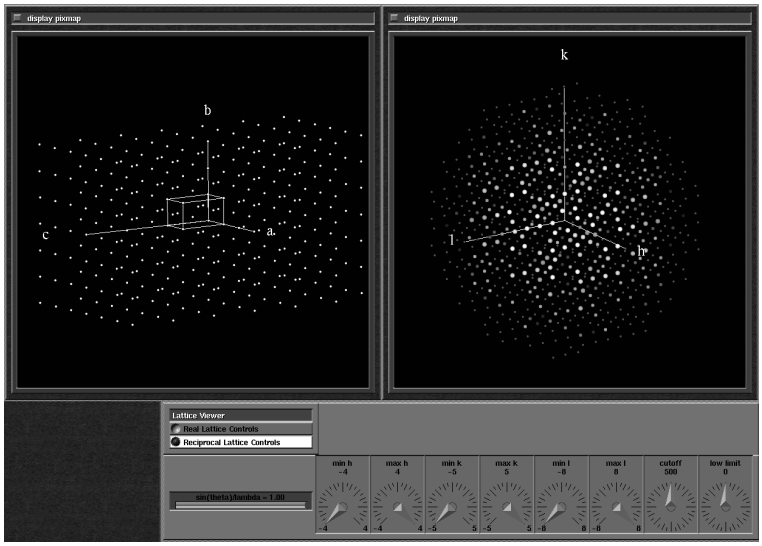


Figure 3. Monoclinic crystal system (same parameters as in Figure 2) real space lattice and reciprocal space diffraction pattern. Below the windows, the view controls for the reciprocal space data are shown. Both sets of view controls lie upon pages with appropriate names. All of the other control panels in the network have been hidden for simplicity.

transformation in the left window will snap the geometry in the right window into place.

Fred Hollander has written a FORTRAN program to create the sample datasets used with the CRYSTAL LATTICE VIEWER. Currently, a total of 21 datasets correspond to some common crystal systems suitable for visualization. The data input and processing modules read a two-letter code at the beginning of each line.

The code sets the expectations for data content on each line. Formatting actual experimental data is possible through a FORTRAN filter program also available at the present time. In fact, our first use of the application was on an experimental dataset that was not yielding to traditional analysis methods.

The current semester provides our second chance to use the CRYSTAL LATTICE VIEWER in an instructional setting. We have designed a visualization laboratory session and problem set that is based on this application for the chemistry department's X-ray crystallography course. At the time of this writing, a few students are working on some of their "unknown" lattice structures.

Since these students are not learning UNIX or AVS, the application is designed to insulate them as much as possible from both. In the future, we will be designing additional modules and networks to illustrate experimental aspects of crystallography. An AVS-based simulation environment would be very useful since computer-generated crystals can be inexpensive, complex, and robust.

AVS Tips and Tricks

Use the Layout Editor to simplify and streamline the user interface. This is especially important if you are designing networks for end-users who aren't very experienced with AVS. Often, AVS developers forget how complex and intimidating the environment can be. If possible, design networks that do not require the initial AVS user to build or modify the network while they are learning all the other aspects of AVS.

For this application, more time was spent on designing the user interface aspects of the networks than was spent on the modules and their interactions. Sometimes the native AVS data types don't work with your data. Don't despair, it happens to all of us. When this happens, your best options are to write modules that work in a **data_to_geom** mode or break up your data into multiple fields, ucds, or whatever AVS is providing that you can use.

In this case, it was a bit simpler to add the code for cropping rather than output a field of transformed points, crop with the **crop** module, and then use a module to create the geometry. If you find an application using functionality provided in more than one AVS supported module, it's time to abandon the **data_to_geom** approach. When you want to add functionality in a modular manner, it's also time to abandon the **data_to_geom** approach.

Transformations 1 — If you have data that is best represented by a nonorthonormal basis, try out the **non-ortho** module at the International AVS Center. It is based on the transformation matrix code used in the CRYSTAL LATTICE VIEWER. Currently, it takes a uniform field and applies a transformation to the basis specified by the parameters to the module, creating an irregular field.

Transformations 2 — If your update rate of transforms done in the geometry viewer drops dramatically whenever you connect an output to the image port, a work-around is possible.

For this application, it was unreasonable to expect the users to connect the **image to postscript** module to the output of the **geometry viewer** only when they needed postscript output. Instead, the **render geometry** module was used, and the pixmap output was connected to the **display pixmap** and **null pixmap** modules. The **null pixmap** module waits for a toggle, then it sends its output to the pixmap output port. This then is used as input data for the **pixmap to image** module. That conversion is essentially what holds up the geometry viewer each time the image output port is connected on many AVS platforms. With this approach, the network that the user sees can be configured with a "take snapshot" button, and geometry transforms in the **display pixmap** window stay interactive.

Transformations 3 — How can I grab transformation information so that split screen displays work? Of the two ways to do this, the fastest is to modify the module that generates the geometry so that it gets the transformation matrix for its objects, applies it to its objects, and then passes the matrix on to a friend that can apply the transformation as well. A developer could produce a split-screen stereo display in this manner.

But what if you don't have source code to the module? Try the **transform tool** module. This module connects to the geometry input of the **geometry viewer** module and gets notified whenever a selected object gets moved. If another object is specified in the second type-in, it applies that transformation to the edit list for that object. This is how the CRYSTAL LATTICE VIEWER carries out split-screen viewing of multiple objects.

Acknowledgments
This work was supported in part by National Institutes of Health grant #850RR05651A and by a Digital Equipment Corporation Science Innovators Program Grant.

Additional Reading
George H. Stout and Lyle H. Jensen, *X-Ray Structure Determination, A Practical Guide* (Macmillan, New York, 1965).

Author's Note
For a copy of the CRYSTAL LATTICE VIEWER as a package, including all source for modules, sample data, and networks, send a note to **timro@cchem.berkeley.edu** for anonymous ftp instructions.

Copyright and Trademark Notice
AVS is trademark of Advanced Visual Systems Inc.
CRYSTAL LATTICE VIEWER is a copyright by the University of California Regents (1992, 1993).
UNIX is a registered trademark of UNIX System Laboratories/Novell Inc.
PostScript is a registered trademark of Adobe Systems Inc.
X WINDOW SYSTEM is a registered trademark of the Massachusetts Institute of Technology.

Module Submission Criteria

We encourage modules of all types. We are not looking for perfect productizable code but rather code that is used every day and would be useful to users. Several users have indicated that their code was not good enough or would not be useful to the AVS user community. This is not true.

We have many different levels of users — from the total novice to the experienced professional — and code is needed for all levels. All we require is source code, a Makefile, and either a README file or man page. Data, networks, etc. are strictly optional.

Source file and Makefile — Source code and Makefile must accompany all modules submitted. We encourage extensive commenting and ask that port-specific lines be so commented.

Module description — A detailed description of the module must be included with its submission. This may be in the form of a man page or ASCII text format (as in a README file). Module description is one of the most important criteria for submission. Please note if multiple modules are in one source code file. The International AVS Center staff will rewrite the documentation as necessary.

Data — If your module uses a nonstandard data format, you must include an example (not necessarily real) of that data. Explanations of how to read the data should be placed in the required manual or README file.

Miscellaneous — You are requested to provide sample network and scripts when appropriate.

To submit modules, ftp to **avs.ncsc.org** and cd to the **SUBMIT** directory. You will be prompted in how to create a directory. The directory itself will be invisible so that others may not inadvertently copy over your files. When you are in your directory, use standard ftp protocol to submit your modules and associated files. Information on using ftp for basic submission and retrieval can be obtained by sending a request to **avs@ncsc.org**.

You also can submit modules and associated files by email to **avs@ncsc.org** and by tape (tape will be returned if requested) to the International AVS Center, Post Office Box 12889, 3021 Cornwallis Road, Research Triangle Park, NC 27709-2889.

AVS Support in the United Kingdom

Steve Larkin, Manchester Computing Centre, University of Manchester, Manchester, United Kingdom

Introduction
The following article outlines some of the events and activities under way in the United Kingdom (UK) to support visualisation and AVS. As the UK academic community in general were taking an interest in the emerging software for visualisation, the Advisory Group on Computer Graphics (AGOCG) provided funding for a Visualisation Support Post based in the Computer Graphics Unit, Manchester Computing Centre.

The overall aim during this period is to raise the awareness and promote the use of visualisation systems in the UK. To meet these goals, a number of activities have been organised during the coming year, and some of these are outlined in this article.

Campuswide Deal for AVS
One of the activities that AGOCG arranges are evaluations of both hardware and software. An evaluation of visualisation software systems was completed in January 1992 with one of the recommendations being to promote the use of visualisation systems using AVS. As a direct outcome from the evaluation, the Combined Higher Education Software Team (CHEST) negotiated a campuswide deal on behalf of the UK academic institutes for the AVS software.

UK AVS User Group
The UK AVS User Group currently is being established and has an interim committee with J.R. Gallop, Rutherford Appleton Laboratory, as the chairman and W.T. Hewitt, Computer Graphics Unit, Manchester Computing Centre, as the secretary. Membership is open to all, not only residents of the UK. The main source of members is the territory of the UK offices of Advanced Visual Systems Inc., which covers northern Europe.

The purpose of the group is to encourage the free exchange of information concerning visualisation and the products and services of Advanced Visual Systems Inc. and to provide a focal point for the needs and requirements of its members. The group also will liaise with Advanced Visual Systems Inc., the AVS Consortium, and the International AVS Center.

The user group is planning to hold a one-day meeting sometime in late March or early April in the UK, and this will include speakers from Advanced Visual Systems Inc. and users. At this meeting, the UK AVS User Group formally will constitute itself.

The interim committee will propose two classes of membership.
- Individual — The annual fee is £20.
- Corporate — The annual fee is £100 and entitles the organisation to nominate seven people for individual member services.

Membership will entitle members to reduced rates for local meetings and publicity material that will include copies of ***AVS Network News***.

If you would like more information on joining the group or the first meeting, contact

Mr. W. T. Hewitt
Computer Graphics Unit
Manchester Computing Centre
University of Manchester
United Kingdom
Telephone +44 (0)61 275 6095
Fax +44 (0)61 275 6040
Email **w.t.hewitt@mcc.ac.uk** or
hewitt@hpa.cgu.mcc.ac.uk

AVS Courses
A number of AVS courses have been arranged in the UK for users in the academic community. Two courses are currently available. An introductory course covers the basics of AVS, and the advanced one teaches users to write their own AVS modules. These two courses are scheduled for the following dates and will be held at the University of Manchester:
- Introductory Course — 12th May 1993
- Advanced Course — 14th April 1993

For a booking form, please contact Steve Larkin (**s.larkin@mcc.ac.uk**). As a follow-on to the courses, a meeting in spring 1993 is planned for producing a set of training materials for AVS.

Electronic Mail Service for Visualisation
An electronic mail service is provided to disseminate general information concerning scientific visualisation and its associated

software products. The electronic mail list is hosted as part of the mailbase service at Newcastle. Activities and events being organised within the UK Academic Community with the theme of visualisation are circulated via this list and include those organised by AGOCG and the EASE Visualisation Community Club.

How to Join the List
To join the electronic mail list, simply send the following message to mailbase (**mailbase@uk.ac.mailbase**).

join chest-visual <first name> <surname>

For example,

join chest-visual John Smith

This will add your name (John Smith) and mail address (taken from your message header) to the chest-visual discussion list. For more general help on mailbase, send a message that contains the word help to **mailbase@uk.ac.mailbase**.

Further Information
The preceding sections have the appropriate contact details included. But if you have any comments or suggestions, then please feel free to contact me.

Steve Larkin
AGOCG Visualisation Support Officer
Computer Graphics Unit
Manchester Computing Centre
University of Manchester
Oxford Road
Manchester M13 9PL
United Kingdom
Telephone +44 (0)61 275 6095
Fax +44 (0)61 275 6040
Email **s.larkin@mcc.ac.uk** or
mzagssl@hpa.cgu.mcc.ac.uk

Articles Needed

If you have articles for ***AVS Network News***, send them to **avs@ncsc.org** or to the International AVS Center, Post Office Box 12889, 3021 Cornwallis Road, Research Triangle Park, NC 27709-2889.

Obtaining Modules

Modules may be obtained by three basic methods. The first is standard ftp protocol. When you ftp to **avs.ncsc.org** (128.109.178.23), text will be displayed that will guide you in using the ftp site. Please read the AVS_LICENSE file located in the top-level directory before downloading modules from the site.

Among the directories on the site is **avs_modules**, which has several directories within it:

catalogs/	**mappers/**
data_output/	**khoros/**
data_input/	**filters/**

The catalogs directory contains ASCII and PostScript versions of the latest modules, whereas the other directories contain the modules themselves. Each module subdirectory has many Makefiles, one for each port of the module and its associated files.

The second method for obtaining modules is through standard email. When you send an email message to **avsemail@ncsc.org**, you receive automated replies detailing this procedure. These replies also include a copy of the module catalog, the latest README file about the International AVS Center, and International AVS User Group information.

Finally, if you don't have network access, you may request a tape via mail. Please send a tape and a $5 handling fee ($15 if not in the United States) with your request to the International AVS Center, Post Office Box 12889, 3021 Cornwallis Road, Research Triangle Park, NC 27709-2889. State the hardware configuration you will be using, and we will send you a copy of all of the modules on the ftp site.

Using UCD for the Visualisation of Semiconductor Device Simulation Data

Jonathan Cox, Department of Computer Science, University of Manchester, Manchester, United Kingdom

Abstract
Many large-scale simulations are based on the finite element method. One example is 3-D semiconductor device simulation. AVS supports the visualisation of finite element data with the UCD data type and modules. This article provides an overview of how a postprocessing system for data from EVEREST, an existing device simulator, can be implemented using AVS. The article concentrates on the implementation of a custom file reading module and a new mapping module for sampling 3-D UCD on planes. This mapping module is available from the IAC. A paper to be presented at AVS '93 will expand on this article and examine how the device simulator itself can be integrated with AVS.

Device Simulation
A device simulator is a tool for analysing the behaviour of a semiconductor device, such as a transistor or diode, in ways that are impractical or impossible through manufacture and testing. A device simulator works directly from a description of the device's structure and operating conditions to produce a detailed picture of device behaviour. Typically, one dataset is produced per electrical variable per simulation time step. Some of these variables are scalar, such as electrostatic potential and charge density, and some are 3-D vectors (e.g., current density and electric field strength).

The device simulator used is EVEREST, a 3-D simulator developed as part of a European ESPRIT-funded project [1]. The simulator has several modular components covering device definition, the derivation of an initial finite element mesh, the generation of doping profiles (which specify the density of dopant atoms), the numerical simulation itself, and the analysis of results. Although the user is required to visualise complex geometric information throughout the simulation process, this article only considers the visualisation of simulation results.

Postprocessing EVEREST Data
A stand-alone postprocessor for EVEREST has been implemented using AVS. This was a three-stage process. The first stage was to provide a mechanism for importing EVEREST result data into AVS and converting it to the UCD data type. The second stage was to write mapping modules for sampling the resulting UCD data on planes and lines — methods that device engineers use extensively. The final stage was to link the modules together into data flow networks, each one implementing a particular visualisation technique, and then combine the networks together to make a usable package.

Importing EVEREST Data
EVEREST result data, which is held in RALBIC format [2], was imported into AVS by means of a custom file reading module and then output as UCD structures. This method was chosen over an offline conversion program, as it makes it possible to control interactively the conversion process. Although the conversion is trivial, this method is desirable as it requires the selection of a particular solution case and simulation variables.

A solution case (e.g., one time step) is selected through an integer parameter. If this parameter is connected to the output of the **animate integer** module, animation sequences can be created. In addition, the file reading module outputs the range of available solution cases whenever a new simulation is selected. The animation module's parameters then can be reset to this range via further network connections. Due to this close coupling and the use of the layout editor to integrate the modules' widgets, the creation of animation sequences appears to be an integral part of the reading module. This approach avoids the writing of a coroutine module.

Some 12 scalar variables and four vector variables could be included in the structure's nodal data vectors. Including all of them would be excessive, particularly due to a high degree of redundancy among them. Therefore, the file reading module allows at most two variables of interest to be selected. In fact, the simulator only outputs three primary scalar variables, and the file reading module uses these and the device's two doping profiles to calculate the remaining variables as required.

The module also allows the user to display the magnitude of vectors and to map variables to a logarithmic scale. The latter is particularly important for those variables — such as current — that vary exponentially; without it, linear interpolation (which most UCD modules use somewhere) can produce erroneous values.

UCD Modules for Device Simulation Data

Once the simulation data has been converted to UCD structures, modules must be provided that map these structures to geometries for display. AVS supplies several standard modules for doing this. The principle module is **ucd to geom**, which can be used to display the device surface. This is shown in Figure 1, which shows a simplified MOSFET transistor coloured according to electrostatic potential.

The device's contacts are located on the top of the device. Left to right they are the drain, gate, and source. The transistor is in an "on" state (the gate is held high), and current flows from the drain (held high) to the source (held low) via a narrow inversion region just below the gate contact. This is illustrated in Figure 2, which uses the **ucd streamline** module to show current flow. In the figure, current flows from left to right and electrons from right to left. Other UCD modules, such as **ucd iso** and **ucd isolines**, also can be used to good effect.

Sampling UCD on Planes

Due to the structured nature of the majority of device simulation data, one of most popular methods for visualising device simulation data is to display data on a 2-D slice, particularly in the form of a 3-D graph. A module, **ucd planes**, has been written for this that allows the user to sample 3-D UCD data with one of a selection of probes consisting of planes. Figure 3 shows an example of the module in use. Figure 4 shows the other types of probe that are available. The module and the closely related **ucd line** module for sampling on lines, together with manual pages, are available from the International AVS Center.

The slicing algorithm used within the module is based on the ideas of Don Speray and Steve Kennon [3]. In this algorithm, cells are tested

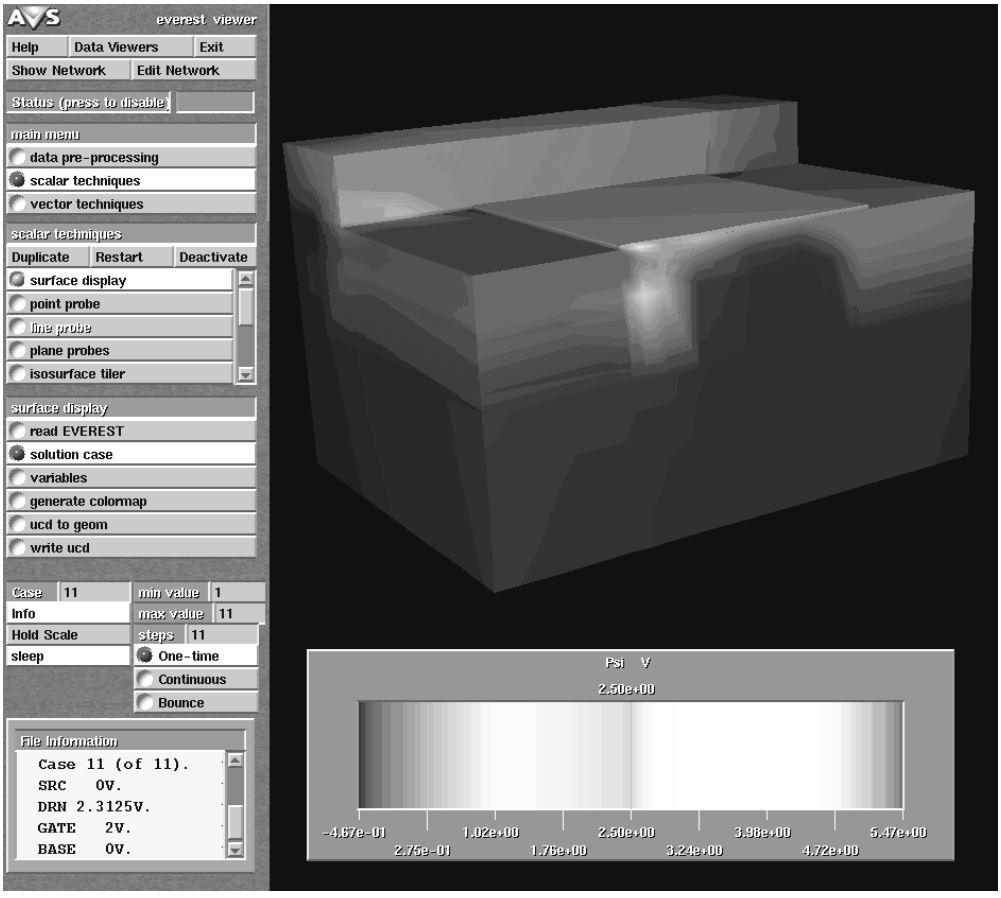


Figure 1. MOSFET device in "on" state coloured according to electrostatic potential.

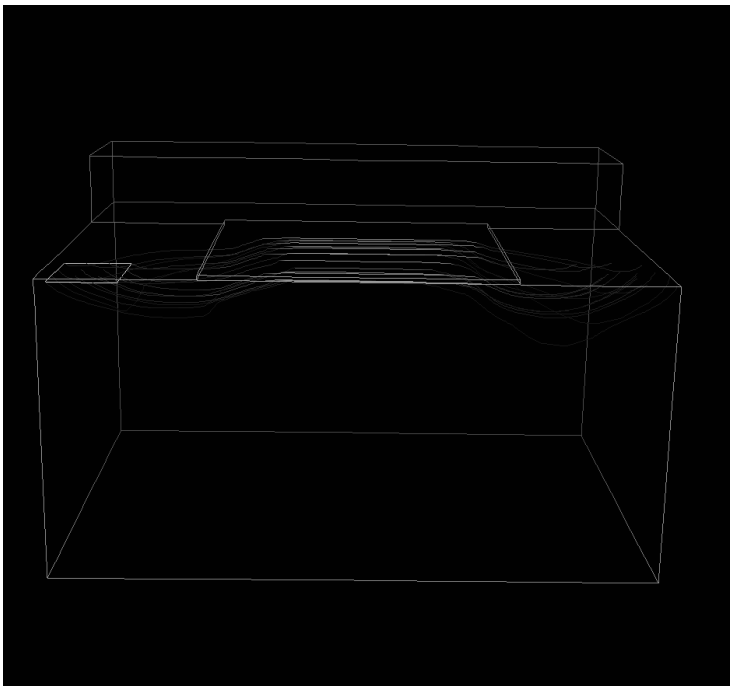


Figure 2. Current flow in the MOSFET.

for intersection with the plane by using precalculated tables. It is possible to test if a cell is intersected with a single lookup. Furthermore, this lookup yields the intersected edges in the correct order for making a cutting polygon.

The actual points of intersection and their data values are calculated by linearly interpolating from the edges' vertices. Points of intersection are stored with the corresponding edge so that the intersection point on shared edges is only calculated once. This has the added bonus that the resulting polygon object has shared vertices.

The position of the probe object can be controlled by either manipulating control widgets or by direct manipulation of a graphical probe object (Figure 3). When the control widgets are manipulated, a transformation matrix is generated by using routines from the mat(3V) library. The actual matrix is derived as follows.

$$\mathbf{M} = \mathbf{T}_{d.\mathbf{n}}.\mathbf{T}_{\mathbf{c}}.\mathbf{Rx}_{x}.\mathbf{Ry}_{y}.\mathbf{Rz}_{z}.\mathbf{S}_{s}.\mathbf{T}_{-\mathbf{c}}$$

c is the centre of the device; *x*, *y*, *z*, *s*, and *d* are the values of the XRotation, YRotation, ZRotation, Scale, and Distance parameters, respectively; and **n** is the normal to the plane — the direction of the distance offset.

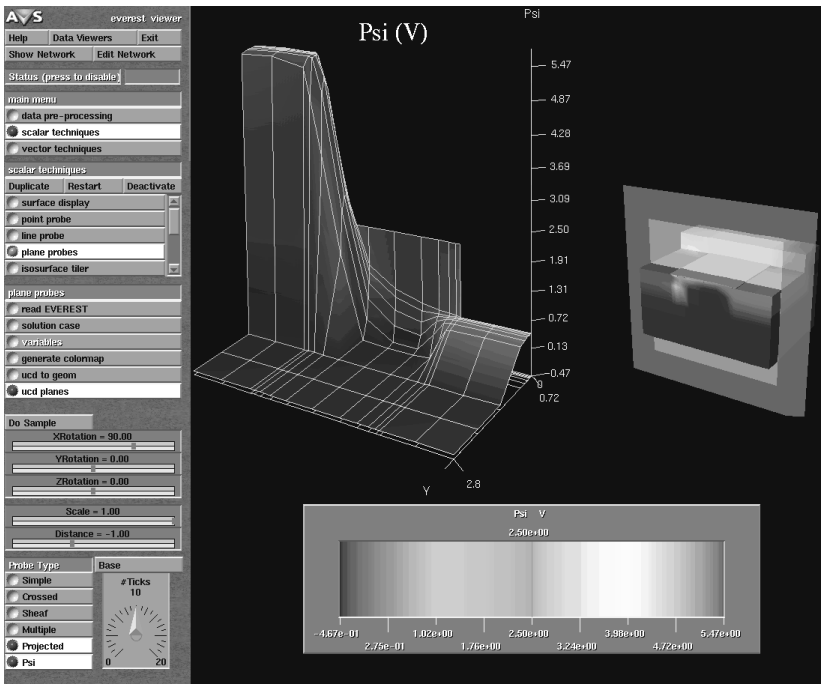


Figure 3. 3-D graph of electrostatic potential on a vertical plane through the front of the MOSFET.

The order of rotations used to generate **M** is actually the opposite of the ordering used in the geometry viewer's Transformation Options panel. However, in this case, this ordering makes more sense. This is because as the original equation of a single plane probe is z = 0, rotating in z first does not actually alter the definition of the plane. The ZRotation parameter is still useful, however, when the probe contains a plane orthogonal to z = 0. It also can be used to alter the orientation of 3-D graphs with respect to their axes. Similarly, the Scale parameter does not alter the definition of single plane probe but does affect probes with multiple planes and the aspect ratio of the 3-D graph. Therefore, when using this transformation order, the widgets have distinct functionalities.

If the probe object is manipulated via direct manipulation in the geometry viewer (or from the transformation options panel), an upstream transformation matrix is generated. The **ucd planes** module interprets this and sets the control widgets to a consistent state. This is important as it provides a numerical definition of a plane that has been derived visually. This is done by assuming that **M** was formed by the equation above.

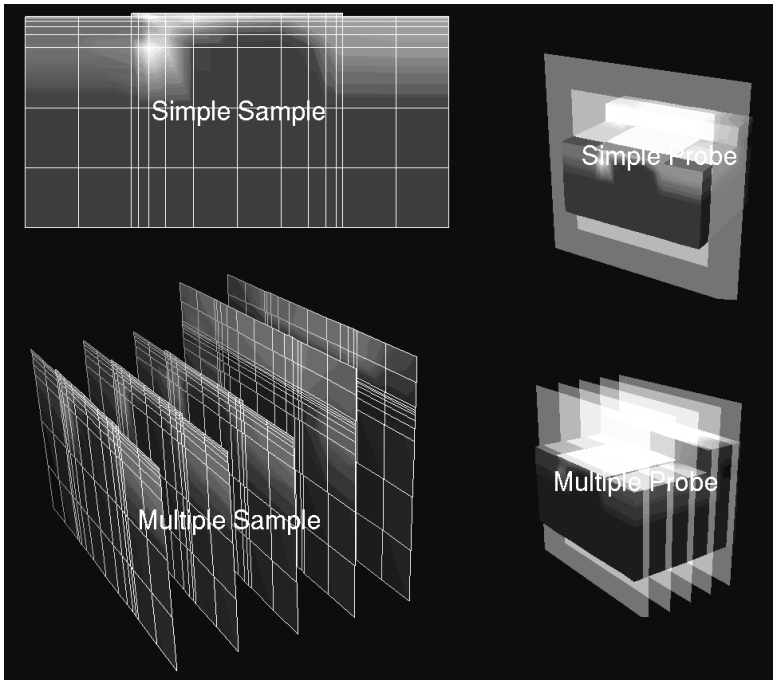


*Figure 4. Probe types implemented by the **ucd planes** module.*

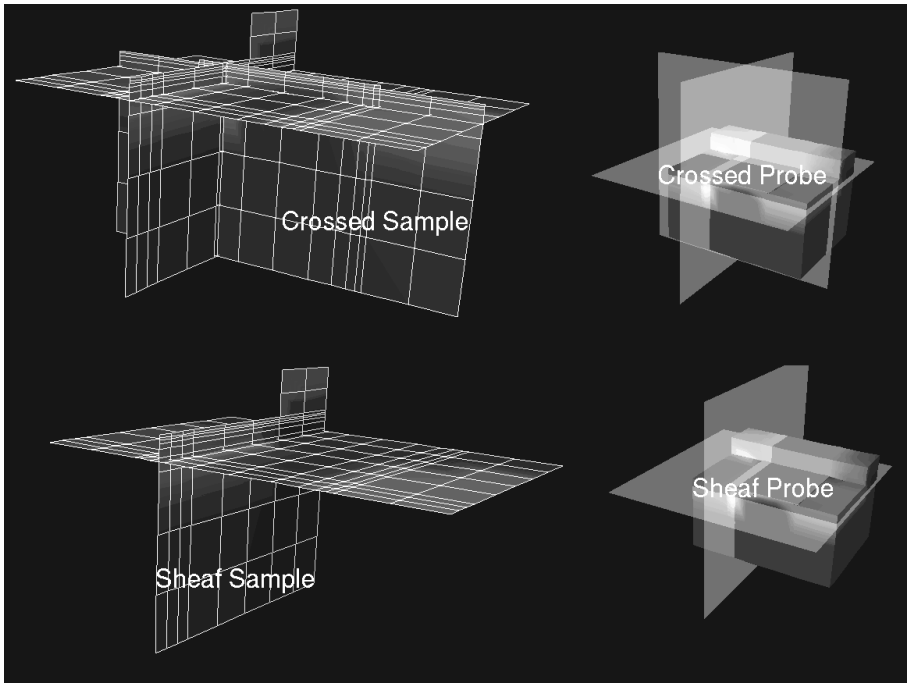


*Figure 5. Probe types implemented by the **ucd planes** module.*

The Distance parameter can be derived by taking the dot-product of the vector M**.c - c** with **n**. Deriving the rotation parameters is more problematical, but it can be done by observing transformation of **i** and **k**, the unit vectors in x and z, under **R =** $\mathbf{Rx}_X.\mathbf{Ry}_Y.\mathbf{Rz}_Z.\mathbf{S}_S$, which is the upper 3x3 matrix of **M**. The Scale parameter can be found by taking the magnitude of any row of **R**.

If the probe object is a single plane, it is useful to restrict the movement of the probe so that the normal of the plane always passes through the centre of the device. This can be done by resetting the translation component of **M** to *d***.n + c - R.c**. If *d* is kept constant during rotations, it is also possible to ensure that the probe is rotated about the centre of the device rather than the centre of the probe object.

If the transformation mode of the probe object is set to "notify," the resetting of its matrix will cause a second translational jump once the transformation is complete. However, if the "redirect" mode is used, this can be avoided by explicitly transforming the vertices of the probe before it is output. The decoding of the transformation matrix and the redirecting of the probe's transformations are documented in greater detail within the module source.

Building an Application

Once the necessary modules were in place, the construction and customisation of application networks was straightforward. However, to make a usable application, these networks needed to be combined into a usable package. AVS' viewer paradigm was used to do this. This gives access to a number of networks via a hierarchical menu system (Figures 1 and 3). AVS then can be started with the viewer option to hide the presence of the underlying networks from the user.

Conclusions and Further Work

AVS provides considerable support for the construction of a postprocessor for simulation data based on a finite element mesh. In fact, if the required visualisation modules already exist, the task is reduced largely to one of data conversion and network editing. However, most simulators also require visual support at other stages of the simulation process. This can be partially achieved by a postprocessing system.

For example, the postprocessing system described above is capable of displaying the geometry of the device, the doping profiles, and the initial computational mesh before the simulation's solver is run. However, using AVS to provide visual support for the whole of the visualisation process really requires that the simulator itself be integrated with the AVS environment. This is generally much more involved and is an issue that will be addressed in a full paper at AVS '93.

Acknowledgments

I would like to thank my supervisor, Hilary J. Kahn, and other members of the CAD group for their support and guidance; the staff of the Computer Graphics Unit at the Manchester Computer Centre for their help and the use of their resources (the figures were generated on their HP9000/730); members of the Computational Modelling Division of the Rutherford Appleton Laboratories for help and access to EVEREST; and the Science and Engineering Research Council for funding me.

References

[1] C. Greenough, D. Gunasekra, C.J. Fitzsimmons, P.A.W. Mawby, M.S. Towers, "Modelling Semiconductor Devices in Three Dimensions," Software Tools for Process, Device and Circuit Modelling, ed. W. Crans, Boole Press, 1989.

[2] K.P. Duffey and C.R.I. Emson, "RALBIC — A Simple Neutral File for Finite Element Data — File Definition," Technical Report RAL-87-102, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon., OX11 0QX, UK, December 1987.

[3] D. Speray and S. Kennon, "Volume Probes: Interactive Data Exploration on Arbitrary Grids," Computer Graphics, 24(5), pages 5-12, November 1990.

AVS Courses Offered

The North Carolina Supercomputing Center (NCSC) recently developed a five-day hands-on AVS curriculum that now is being offered several times during the year. NCSC, host of the International AVS Center, is a division of MCNC. Courses include

- **Visualizing Your Data with AVS** (2 days),
- **Writing AVS Modules** (2 days), and
- **Visualizing Chemistry Using AVS** (1 day).

Registration priority for this course sequence is given to industrial and academic institutions located within North Carolina, although a limited number of openings for non-North Carolina-based students is available. Please contact the International AVS Center at **avs@ncsc.org** for information on course schedules, pricing, and availability.

Volunteers Needed

The International AVS Center (IAC) needs volunteers to assist with preparations for AVS '94. Individuals are needed to chair lecture tracks, assist with workshop tracks, schedule speakers, and prepare and coordinate panel discussions. A volunteer from the West Coast is needed to work with IAC in preparing for AVS '94, and a volunteer is needed to assist with preparations for AVS '95. For more information, contact

International AVS Center
Post Office Box 12889
3021 Cornwallis Road
Research Triangle Park, NC 27709-2889
Telephone 919-248-1100
Fax 919-248-1101
Email **avs.ncsc.org**

In Upcoming *AVS Network News*

Look for these and other features in the ***AVS Network News*** published in conjunction with AVS'93 held May 24-26, 1993, in Lake Buena Vista, FL, USA.

- **AVS Projects at the Center of Innovative Computer Applications**, Brian Kaplan and John MacCuish, Center of Innovative Computer Applications, Indiana University, Bloomington, IN, USA

- **The Role of Visualization in Space Science**, E. Szuszczewicz, A. Mankofsky, and P. Blanchard, Laboratory for Atmospheric and Space Sciences, Science Applications International Corporation, McLean, VA, USA

- **UAMworld: Prototyping a Distributed Environment for Air Quality Modeling and Visualization in AVS**, Steve Chall and Kathy Pearson, North Carolina Supercomputing Center, a division of MCNC, Research Triangle Park, NC, USA

- Plus, images courtesy of Kubota Computer Incorporated, Tokyo, Japan

In Touch with the IAC

avs@ncsc.org
Email address for sending questions to the International AVS Center staff or articles for reprint in ***AVS Network News***.

avs93@ncsc.org
Email address for AVS '93 information.

avsemail@ncsc.org
Email address for an automated reply containing information on the International AVS User Group, current AVS module catalog, README file on the International AVS Center, and use of standard email to obtain modules.

avsorder@ncsc.org
Email address for ordering AVS module source code.

listserv@avsusers.ncsc.org
Email address for sending any message, with HELP as the message body, for an automated reply detailing the International AVS Center's various mailing lists and subscription procedures.

avs.ncsc.org
Anonymous ftp site address of the International AVS Center.

comp.graphics.avs
Newsgroup of AVS users' daily postings, also available by using **listserv**.

WHAT_IS_GOPHER
WHAT_IS_WAIS
Files on the International AVS Center's ftp site (in the **avs_readme** subdirectory) for information on other tools for accessing the ftp site.

AVS ’93 presents

The Magic of Science

2nd Annual International AVS User Group Conference
WALT DISNEY WORLD DOLPHIN
Lake Buena Vista, Florida

May 24-26, 1993

Visualization. Imaging. Computer graphics. Whatever term you use, the Application Visualization System (AVS) is now the worldwide standard for that spectacular industry. And now, AVS ’93 puts *The Magic of Science* in your hands with spellbinding visualization power.

AVS ’93 blends a masterful mix of workshops, tutorials, and lectures; mesmerizing panels and video reviews; plus entrancing exhibits. It’s a wondrous world of visualization capabilities that make the magic of science work for you.

Join us for a spell. You’ll see.

TOPICS
Computational chemistry • Computational fluid dynamics • Earth sciences • Education and research • Geographic information systems and remote sensing • Image analysis • Medical imaging • Molecular modeling • Oil and gas exploration • Scientific visualization • Structural analysis

REGISTRATION
To obtain an AVS ’93 registration form and detailed program, send your name; company name; complete mailing address, including city, state, country, and zip code; email address; and telephone number to International AVS Center, AVS ’93 Conference Management, Post Office Box 12889, 3021 Cornwallis Road, Research Triangle Park, NC 27709-2889, USA, Telephone 919-248-1110, Fax 919-248-1101, Email **avs93@ncsc.org**, Program Chair David Bennett, Telephone 919-248-1182.

SPONSORSHIP
AVS ’93 is presented by the International AVS Center and the North Carolina Supercomputing Center, a division of MCNC, in cooperation with the following corporations.

Advanced Visual Systems Inc. • CONVEX Computer Corporation • Digital Equipment Corporation
Hewlett-Packard Company • IBM • Kubota Pacific Computer Inc. • SUN Microsystems Inc.

ADVANCED VISUAL SYSTEMS INC. · CONVEX · digital · HEWLETT PACKARD · IBM · KUBOTA PACIFIC COMPUTER INC. · Sun

MCNC

Established in 1980 as a private, nonprofit research company, MCNC fosters technology research and education to attract and generate technology-based industry for North Carolina. MCNC matches the strengths of major research universities with those of established and emerging industry for economic growth and development in the state. The organization provides advanced capabilities and expertise in microelectronics, communications, and high-performance computing to support research, education, and industry.

The North Carolina Supercomputing Center (NCSC), the supercomputing division of MCNC, makes supercomputing time and related resources available to North Carolina’s academic researchers and encourages nonacademic institutions to explore collaborative ventures. NCSC also facilitates research in the areas of computer science, computational science, and scientific visualization.

The International AVS Center, housed at NCSC, is the worldwide clearinghouse for collecting, porting, and distributing public-domain visualization modules. These modules enable dramatic advances in the scope and speed of research in a variety of disciplines. The International AVS Center serves as a catalyst for expanding the AVS user base and for increasing AVS functionality by fostering discipline-specific module development and new AVS uses.