Volumetric objects and hybrid rendering

In addition to being able to render geometry, the RenderMonster application in the Geometry Toolbox is also capable of rendering volumetric data directly. A voxel splatting approach is used for volumetric rendering. A voxel dot approach is also available for faster, more interactive rendering.

The RenderMonster application does not make a strong distinction between geometric data and volumetric data and is capable of rendering both geometric data and volumetric data together in a single rendered scene.

3.4.9 PV-WAVE

The presentation capabilities of this package are slanted towards presenting 1D/2D data and images with some facilities for 3D arrays of data.

Image display

Besides displaying images they may be warped, frequency domain filtered, and similar image processing tasks applied. It appeared that only indexed colour was supported, in other words images were required to have a colour table.

Geometric objects

3D data may be plotted as surface and contour plots, with either a network (wire mesh) surface or shaded surfaces (flat or Gouraud). There is no facility for Phong shading.

There are also some procedures to create, manipulate and display polygon and vertex data within PV-WAVE and these are described in Reference Volumes 1 and 2 and its use in the PV-WAVE User's Guide, Chapter 6: Advanced Rendering Techniques.

Translation and rotation of objects is specified with 4x4 matrices using a command language. There does not appear to be a direct manipulation interface for this. A form of pick is available but this gives a 2D position in pixel coordinates rather than a 3D position in world coordinates.

Volumetric objects and hybrid rendering

Volume processing produces a 2D image from a particular view. The package does not support the mixing of volumetric and geometric objects.

Chapter 4: Data Import

4.1 Introduction

The tables which follow are used to list the currently available data readers for each visualization system. It has not been possible to verify the robustness of each reader or the availability across the platforms each system supports during the review. The following key is used to indicate the source of the reader:

- standard(s): is supplied with the visualization system by the vendors;
- public domain(p): is obtained from an anonymous FTP site. The full list of sites and FTP addresses can be found in Chapter 8: Additional Information;
- commercial(c): can be obtained from a commercial supplier. These readers have a cost associated with them and more information can be obtained by contacting the supplier.

4.2 Application Visualization System (AVS)

4.2.1 Supported data readers

This section outlines the data readers for specific formats which are available as part of the visualization system.

Data format	Key
AVS UCD	s
AVS Field	s
AVS Geometry	s
NASTRAN (SciViz)	c
ANSYS (SciViz)	c
Abaqus (SciViz)	c
DYNA3D (SciViz)	c
FIDAP (SciViz)	c
FLUENT (AdamNet)	c
PATRAN(Tessella)	c
AutoCAD DXF	p
FLOW3D Harwell CFDS	c

SDRC Ideas	p
PAFEC FE	c
TIFF	p
GIF	p
RLE	p
Wavefront OBJ	s
Movie BYU	s
Protein Data Bank	s
Mathematica 3 Script	s
Maple V	c
Abacus A60 Video	p
HDF (SDS and image)	p
JPEG Image Compression	p
MPEG Video Compression	p
Postscript	s/p
Gaussian	c
CadPak	c
Shakall	p
Eclipse Intera	c
GE CT Scanners	p
Siemans CT Scanners	p
Vis5d	p
Sun Rasterfile	p
ICC	p
LBL	p
Plot3d	s
STAR-CD (Tessella)	c
Electra Vector Fields	c
KIVA II	p

Speed (Imperial College)	c
Semper	c
Vbase GEC Marconi	c
SunVision IPLIB	c
Crystal Image	c
Intra Vascular Ultrasound	c
MoPAC	c
RayShade	p
Scitex CT2D (cmyk print)	p
Encaps. Color Postscript	s/p
Sun Icon & Cursor	p
Sun TAAC	p
Apple MacPaint	p
Portable Bitmap Files (pbm)	p
Zsoft IBM-PC Paintbrush	p
Portable Gray Bitmap	p
PIXAR image files	p
Apple QuickDraws/PICT	p
QuickTime Apple/SGI	p
Alias Research image file	p
SGI RGB image file	p
Wavefront raster image(.rla)	p
Utah Run-length image	p
Synu Image	p
XBM X11 Bitmap	p
XWD X11 Window Dump	p
MARC FE	c
FITS - Astronomical	p
QUAL Field	p

netCDF	p
CATIA - IBM	c
ARC/INFO MOSS Markers	p
ARC/INFO MOSS Polylines	p
ARC/INFO MOSS Polygon Covers	p
ARC/INFO TIN	p
GOCAD	p
SET	p
NTF (Ord. Survey Maps)	c
MADYMO TNO Delft (Tessella)	p
CRANK (Tessella)	p
SunVideo XIL Movies	c
Oracle SQL	c
Khoros Library	p
Simpa Library (France)	c
Stratamodel (Intera)	c
HP-Deskjet 550-C (Tessella)	c
CGM Binary	s
CGM Clear Text	s
CGM Character Encoding	s
FLAVIA - CFDS/Flow3D	c
SEG-Y Seismic Format	p
TVAM 3D Velocity Models SPT	c
IDL library	p
Kingfisher - Leading Techn.	c
INVERMOD - Jason Geosystems	c
FOCUS - Medical	c
Xidak Database Server	c
GRASS IP Library	c

MPGS - Cray Research	c

4.2.2 Tools for importing data

There are three main methods of importing data into AVS:

Standard AVS modules: there are a number of modules provided to read data files in one of the defined AVS datatype formats e.g., read_ucd, generate_colormap, read_geom;

AVS read_field: this module parses an ASCII header file which describes the organisation and structure of a binary/ascii datafile. The data is then imported and mapped onto the AVS field datatype. As the AVS field datatype is used then it is restricted to general N dimensional arrays of M data elements;

AVS Data Interchange Application (ADIA): this tool provides the ability to import data which matches the AVS field datatype but has a number of advantages over the read_field module:

- interactive graphical user-interface;
- supports variables and expressions removing the need for absolute values;
- reads information from the datafile itself e.g., dimensions, length;
- searches the file for token keywords which specify format values e.g., length=72;
- the forms created can be saved or loaded for re-use.

Overleaf Figure 8 shows a sample of the ADIA interface.

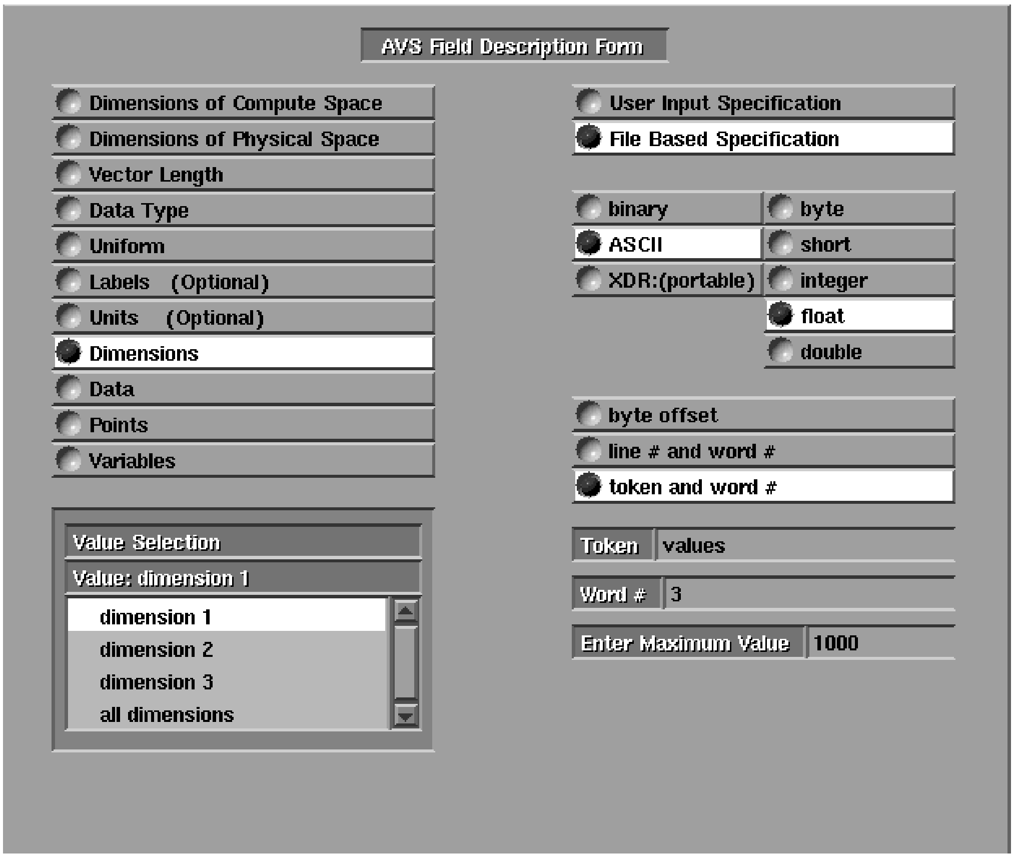

Figure 8: Example of the ADIA interface

4.3 IBM Data Explorer

4.3.1 Data readers

A number of modules and macros are available via anonymous ftp from the DX repository at Cornell (ftp.tc.cornell.edu) under the directory `pub/Data.Explorer` more details are given after the table.

Data Format	Key
NetCDF	s
HDF	s
DX-format	s
General Array Format	s
ABAQUS Standard 5 (SciViz)	c

Data Format	Key
FIDAP 7 (SciViz)	c
FLUENT (SciViz)	c
MSC/DYTRAN 2 (SciViz)	c
STAR_CD 2.1xx (SciViz)	c
ANSYS 5 (SciViz)	c
PAM-CRASH 12 (SciViz)	c
MSC/NASTRAN V67.5 (SciViz)	c
RAMPANT	c
ARCInfo	c
NEKTON	c
PLOT3D	p
PHOENICS	p
PDB	p
JPEG	p
FLUENT Universal files 4.x	p

4.3.2 Tools for importing data

Data Explorer being an object-oriented based systems allows objects to be one of the following classes: group, series, multigrid, compositefield, field, array, constantarray, gridpositions, regulararray, productarray, gridconnections, patharray, mesharray, xform, string, light, camera, clipped, screen.

For the majority of data input, DX relies on a general array importer. A simple text header needs to be created to describe the structure and location of the data then the import module can read in the data directly. To use a graphical interface to define this header, type:

```
dx -prompter
```

This interface can import many simple arrays, both binary and ascii. For binary data it is possible to specify whether the data is in most significant byte (MSB) or least significant byte (LSB) first. The 2 common forms of vector interleaving are also supported:

1. x1,y1,z1,data1, x2,y2,z2,data2,....
2. x1,x2,..., y1,y2,..., z1,z2,...,data1,data2,...

As part of the prompter it is possible to browse through a data file to set marks and determine offsets in bytes or lines which can be useful when describing ascii data. Although the prompter cannot completely describe irregular grids (e.g. FE data) it can be useful to determine some of the description then complete the header by editing the resulting text.

For a more complete guide to the data model see Chapter 3, “Understanding the Data Model” and for importing data into DX see Chapter 4, “Importing Data”, from the User’s Guide.

4.4 IRIS Explorer

4.4.1 Data Readers

The following is a list of readers supplied with the package or available in the public or commercial domain:

Data Format	Key
AVS Field	s
AVS Image	s
AVS ucd	s
AVS Volume	s
SGI audio file (AIFF or AIFF-C)	s
HDF	s
series of SGI RGB images into a 3D lattice	s
SGI format FIT (SGI)	s
SGI Movie file and outputs a single frame	s
Movie BYU	s
PDB	s
Plot3D grid and solution data	s
CSS crystal file format	s
MOPAC electrostatic potential, coordinates and force files	s
Gaussian 92 log file	s
Sybyl mol file	s
bmp Microsoft Windows bitmap image	p
cur Microsoft Windows cursor image	p

eps Adobe Encapsulated PostScript file	p
gif Compuserve Graphics image file	p
hdf Hierarchical Data File	p
ico Microsoft Windows icon image	p
icon Sun Icon and Cursor file	p
iff Sun TAAC Image File Format	p
mpnt Apple Macintosh MacPaint file	p
pbm PBM Portable Bit Map file	p
pcx ZSoft IBM PC Paintbrush file	p
pgm PBM Portable Gray Map file	p
pic PIXAR picture file	p
pict Apple Mac QuickDraw/PICT file	p
pix Alias image file	p
ppm PBM Portable Pixel Map file	p
pnm PBM Portable aNy Map file	p
ps Adobe PostScript file	p
ras Sun Rasterfile	p
rgb SGI RGB image file	p
rla Wavefront raster image file	p
rle Utah Run length encoded image	p
synu SDSC Synu image file	p
tga Truevision Targa image file	p
tiff Tagged image file	p
viff Khoros Visualization image file	p
x AVS X image file	p
xbm X11 bitmap file	p
xwd X11 window dump file	p
FERead	p
MARC output t16 or t19	p

Multi-blocked Plot3D file	p
Nastran xdb file	p
SEGY file	p
Floating point vectors TCP/IP socket	p

In addition, the following CFD-related readers for IRIS Explorer should soon be available commercially from SciViz:

- ABAQUS
- ANSYS
- FIDAP
- FLUENT
- LS-DYNA3D
- MSC/DYTRAN, MSC/NASTRAN

Finally, NAG has developed readers for a number of formats, including PHOENICS, USGS, NTF, etc. These will be made available as part of the next release (3.0) of IRIS Explorer.

4.4.2 Tools for importing data

Explorer provides a number of tools for importing data.

Standard IRIS Explorer Datatype Readers: There are a set of modules which are capable of reading standard Explorer data types from files, one for Pyramids, Lattices and Geometry. They are capable of reading data saved in either ascii or binary format.

Basic Ascii Format to Explorer Format: There are several modules which take files in simple ASCII formats and produce lattices. For example, the module ReadXData creates a 1D lattice from the data in an ascii file in the following format: n, x1, f(x1), x2, f(x2),... xn, f(xn), where n is the total number of data pairs. These modules are all written using DataScribe (see below).

DataScribe: This is IRIS Explorer's visual programming interface for creating modules to read data in a variety of user-defined formats. It takes ASCII or binary data files and outputs lattices (or vice versa). An example from DataScribe is shown in figure 9. Some of the features of DataScribe are:

- It is possible to use pattern matching to skip text headers or actively search for character strings within a file to find formatting values. The pattern matching includes the use of UNIX "*" and "?" wildcards.
- DataScribe provides a *ruler* for formatting ASCII data that makes reading and writing formatted files easier. It allows the interleaving of text and values at specified columns and reading values only from specific columns.
- Values describing the size and shape of the data can be read in from the data file or set to a

constant value. Also, it is possible to read to EOF removing the need to know the whole extent of the data.

- A number of different files can be read by one DataScribe module and combined into one or more output lattices. DataScribe can also be used to read Explorer formats and transcribe sections or channels of data.
- Parameter values can be attached to widgets on the control panel of a data input module and can be used to section the data. The control panel can be edited from DataScribe by altering the appearance, default values and position of the widgets.
- A number of example DataScribe readers come with the system and the scripts they use are available as a starting point for making further readers. Having created a data reader the script and module resource files are saved and can then be inserted as a module in the Explorer map.

DataScribe only works with Lattices. However, there are examples in the manual (with the source provided on-line) of readers which produce Pyramids, plus the module ComposePyr can be used to build a pyramid from a set of lattices.

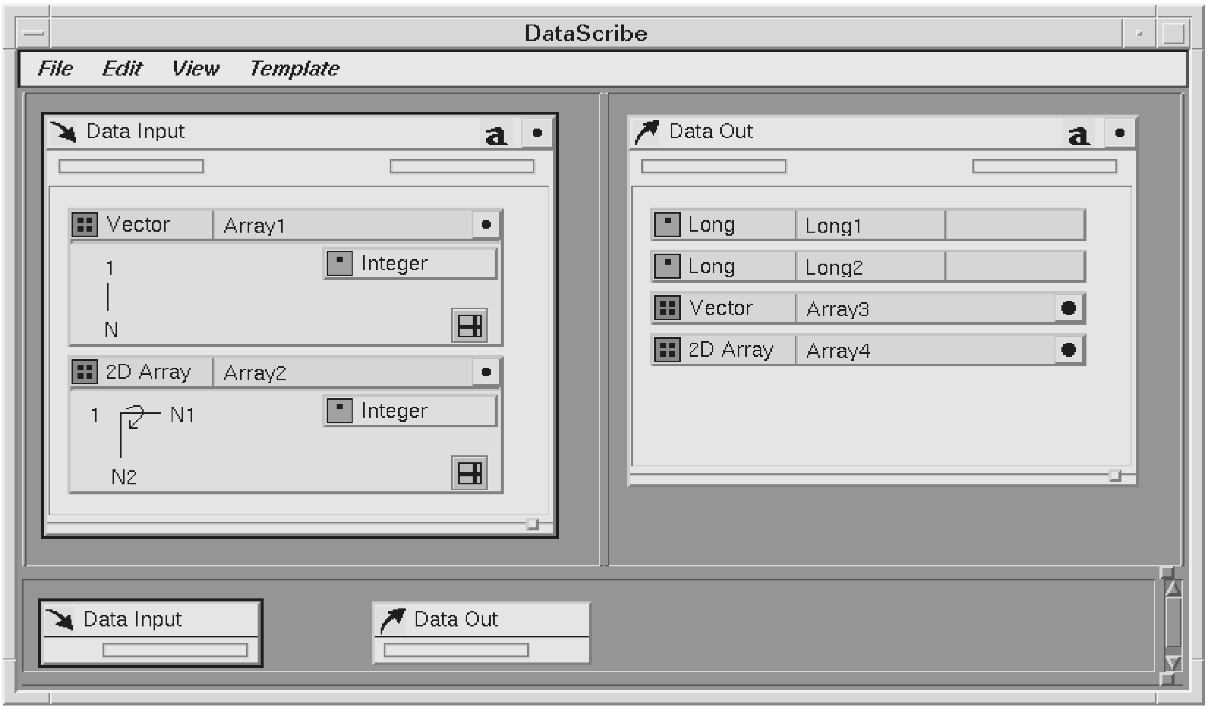

Figure 9: Example of DataScribe

4.5 Khoros

4.5.1 Data readers

The specific formats which are supported within Khoros 2.0:

Data Format	Key
Khoros 2.0 VIFF (viff)	s
Khoros 1.0 VIFF (xvimage)	s
PNM (pnm)	s
Sun Raster (rast)	s
X Bitmap (xbm)	s
X Pixmap (xpm)	s
X Window Dump (xwd)	s
AVS Image (avs)	s
ARF (arf)	s
ASCII (ascii)	s
RAW (raw)	s
Encapsulated Postscript (eps) - output only	s
GIF	s
Erdas (version 7) lan (lan) - input only	p
Erdas (version 7) gis (gis) - input only	p
Digital Elevation Model (dem) - input only	p
Grass ASCII	p
PICT (part of PbmPlus)	p
PC Paintbrush.pcx (part of PbmPlus)	p
TrueVision Targa file (part of PbmPlus)	p
NCSA Interactive Color Raster (PbmPlus)	p
TIFF (part of PbmPlus)	p
FITS (part of PbmPlus)	p

A number of public domain utilities are available; for example the PBM Plus utilities could be used for importing data into a pnm format which can then be read by all Khoros operators.

4.5.2 Tools for importing data

Khoros has a different view on data import. In general, data access is performed through data services using one of the Khoros data models. Data services transparently supports a number of file formats. When a file is opened, data services checks the file to determine if it is one of the supported file formats. If it is, then the data contained in the file will be made available through the various segments in the data model. Applications which use data services can simply open up a file, and if the file is one of the supported formats, it will be able to access it. while most file formats can be automatically understood by Khoros operators, some file formats may need extra information to be imported correctly. The following routines are available for importing ASCII and raw data.

- *kimportasc*: this routine allows you read an ASCII file into the Khoros 2.0 VIFF format, while specifying the size, data type, and index order of the ASCII data. Support for skipping points in the ASCII data is also provided. Simplified versions of this routine which read ASCII data into each of the polymorphic data segments are also available.
- *kimportraw*: this routine allows you to specify details of the raw data format (such as what machine the data was created on) and import the data file into a Khoros 2.0 VIFF file.
- *kformats*: this routine allows you to convert a file stored in any of the supported file formats to any of the other supported file formats.

4.6 PV-WAVE

4.6.1 Data readers and importing data

PV-WAVE CL provides a number of procedures for data input, which are reasonably comprehensive and easy to use, although readers for standard data formats are not available other than for TIFF.

The data input procedures and DC functions include:

- READ (formatted/free format read on the standard input stream)
- READF (formatted/free format read from a specified file)
- READU (binary unformatted read from a specified file)
- DC_READ_FIXED (ASCII fixed format)
- DC_READ_FREE (ASCII free format)
- DC_READ_8_BIT (8-bit image data)
- DC_READ_24_BIT (24-bit image data)
- DC_READ_TIFF (Tag Image File Format)
- Also full support for HDF and netCDF

Equivalent functions broadly exist for data output (please see the data output chapter).

Data import/export error messages and associated options can be specified through the functions:

- DC_ERROR_MSG (returns text string associated with error condition)
- DC_OPTIONS (set error message reporting level)

In general data input is simple and straight forward and the supplied documentation provides useful examples.

Chapter 5: Data Output

5.1 Application Visualization System (AVS)

5.1.1 Hardcopy facilities

There are two main methods within AVS to generate postscript files (monochrome and colour) from scenes and images in the Geometry Viewer and Image Viewer modules are:

Image to Postscript Module

This module takes as its input a standard AVS image and converts the output to Postscript in a file on the system. The module supports two types of postscript output:

- An 8 bit gray scale image for monochrome printers
- A 24 bit true colour image for colour printers supporting the level 1 Postscript operator colorimage or any Postscript Level 2 printer.

The module also allows the user to specify: orientation: landscape or portrait style: encapsulated (for inclusion into other packages) size: the page x and y size in inches

One of the problems with this module though is that it generates images at a resolution of the screen (100 dpi) whereas most printers have a higher (300 dpi) resolution. One method of increasing the resolution the module can produce is to enlarge the image using the geometry_viewer facilities.

geom_save_postscript CLI command

This is the preferred method of generating output from the geometry_viewer module. This CLI command will output the contents of the geometry_viewer window using Postscript lines and text where possible to improve the quality of output. There is however no conversion of polygons to the appropriate postscript representations. To use the facility you must run AVS with the command line interpreter running and when you are happy with the scene in the geometry_viewer execute the CLI command.

5.1.2 Files

There are a number of modules which produce output files from various stages in an AVS network. These are listed below with a brief description:

Standard AVS modules

Image to CGM: converts the input image into the Computer Graphics Metafile (CGM) format. All three encodings, binary, character and clear text are supported and control is provided for the page height and width and orientation.

Write field: writes an AVS field data type to disk. The file has two sections: an ASCII header and the data written in a binary format. The module allows control on the format of the binary portion and it can be written in the machine's native format or in Sun's (external data representation) XDR format which is useful for transporting files across machine ranges.

Write UCD: writes unstructured cell data to a file in either binary (compact) or ASCII (human readable) format.

Write image: writes an AVS image data structure to a file. These files can be read by the AVS module read_image and some public domain image conversion software e.g., customised versions of xv and it is a supported format for the San Diego Image Toolkit.

Write volume: writes an AVS volume data structure to disk. These files can be read by the AVS module read_volume.

ip write vff: converts an AVS image data structure into a SunVision binary vff image format file.

Write structure file: convert an AVS Molecule Data Type (MDT) data structure into a structure file (and associated formal charge file). A structure file is one of the formats supported within the AVS Chemistry Developers Kit.

Public domain modules

Create_MPEG: creates MPEG movies from a series of AVS images. It makes use of Andy Hung's MPEG which is available by anonymous ftp from Stanford. Geom_to_Wavefront: generates an equivalent Wavefront (.obj) file for each polyhedron and polytriangle in an AVS Geometry object.

WRITE_ANY_IMAGE: This module writes an image from an AVS Network in a variety of formats which the San Diego Supercomputing Center's image tools support. Any of the following image file formats can be written by this module:

Abbreviation	Description
eps	Encapsulated PostScript file
gif	Compuserve Graphics image file
hdf	Hierarchical Data File
icon	Sun Icon and Cursor file
iff	Sun TAAC Image File Format
mpnt	Apple Macintosh MacPaint file
pbm	Portable Bit Map file
pcx	ZSoft IBM PC Paintbrush file
pgm	Portable Gray Map file

Abbreviation	Description
pic	PIXAR picture file
pict	Apple Macintosh QuickDraw/PICT file
pix	Alias image file
pnm	Portable aNy Map file
ppm	Portable Pixel Map file
ps	PostScript file
ras	Sun Rasterfile
rgb	SGI RGB image file
rla	Wavefront raster image file
rle	Utah Run length encoded image file
rpbm	Raw Portable Bit Map file
rpgm	Raw Portable Gray Map file
rpnm	Raw Portable aNy Map file
rppm	Raw Portable Pixel Map file
synu	Synu image file
tiff	Tagged image file
x	Stardent AVS X image file
xbm	X11 bitmap file
xwd	X Window System window dump image file.

field2_to_Math: allows a two-dimensional scalar field to be imported into Mathematica from AVS. The session is initialised with a package of Mathematica commands which start a MathLink communication channel to AVS. Other initialisations are made so that the Mathematica command AVSReadField will read a two-dimensional scalar field from AVS.

field_to_EXCEL: ASCII Excel spreadsheet format files are produces containing field data in a table format. These files can be imported into a suitable excel format for analysis and plotting. Separator characters may be specified for differing output versions, three possible floating point output formats are selectable, and the maximum number of values per line is adjustable.

Output a60: The output a60 module takes an AVS image data structure as input, converts it into Abekas YUV format and sends it to an Abekas a60 digital disk recorder.

ucd_to_wave: The ucd to wave module writes a ucd structure to disk, in wavefront format, which is supported by Data Visualizer. Ucd to wave works only with node data, therefore if you have cell data, you have to transform the cell data to node data using the cell to node module.

write_Dore_i: converts an AVS image data structure into Dore format

write_KSWAD: converts an AVS image data structure into a KSWAD format

write_MooV: Create QuickTime sequences from AVS images. 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.

write_jpeg: Compresses an image with the JPEG compression standard and writes it to a file.

5.1.3 Animation and Video facilities

Standard AVS

With standard AVS users can control the behaviour of downstream modules in an AVS network by using the animated float and animated integer modules. These modules both output a stream of numbers which can be used to control the parameters of modules downstream in the network. The modules allow the user to input the minimum and maximum range of the numbers along with the number of intervals or steps to proceed with. This allows the control and generation of *flipbook* style of animation.

The modules image_viewer and geometry viewer both have facilities for the creation of simple flipbook animation by recording a sequence of images or geometries and then playing them back from memory. The playback speed and performance is highly dependent on the platforms rendering speed and memory configuration.

Public domain modules

There are a few public domain modules which provide animation facilities extra to the ones in standard AVS. The create_mpeg module has been mentioned in the previous section on file output.

CICA Keyframe module: this module is connected to the geometry viewer module and controls the of objects by manipulating their transformation matrixes. The user can define a number of keyframes and then interpolate between them using linear or spline interpolation.

Fast Animate: this module is designed animate a series of AVS geometries which have been written to disk. The files are defined with a prefix and then a frame sequencing number with controls to step forward, back and play a sequence.

AVS Animator

The AVS animator package consists of a number of modules and a separate license to the standard AVS one is needed to use the AVS animator module.

The package consists:

- AVS Animator
- Read Frame Seq
- Write Frame Seq
- Output ImageNode
- Output VideoCreator
- Prepare Video

AVS Animator is the main module within the package and provides a front end to the keyframe style of animation. The animator module can control the rendering modules: image viewer, geometry viewer, graph viewer etc. It can also control the parameters associated with modules in the same AVS network upstream of these rendering modules.

The user generates an animated sequence or script by defining a number of keyframes and then asking the AVS Animator to in-between or interpolate between these keyframes. The keyframe definition consists of the objects attribute information within the rendering module and also the parameter settings in the modules upstream of the renderer. An example scenario could be a rotating dataset with the isosurface value changing from its minimum to maximum value. Obviously more complex animations are possible.

When an animation has been generated the user has to hand a more complex user interface to control the aspects affected at each keyframe (see figure 10). This allows the user to edit the final sequence.

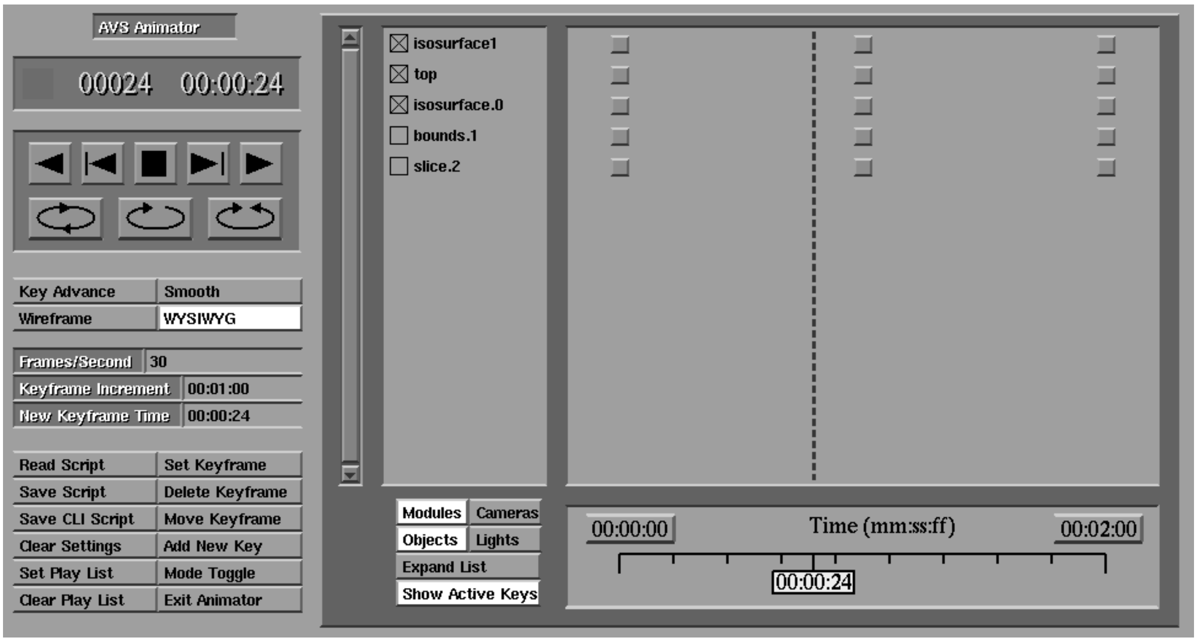

Figure 10: AVS Animator Interface

As the animator module generates an animated sequence useful information is sent out of its output ports: frames/second, frame number and current time.

Finally the animator allows the user to save and load scripts that they have generated.

Along with the main animator module come a number of support modules:

Write Frame Sequence: compress a series of images which make up a flipbook style of animation into a single file. The module can also add and delete frames from an existing frame sequence file. It is useful for collecting images from a simulation where each time step takes a time to calculate.

Read Frame Sequence: read a series of files written to a frame sequence file by the write frame sequence module. The module has the animator style control panel and optional controls over the playback speed and current frame.

Prepare video: pre-process animations before sending them to a video device. The processing that is supported:

- Low pass filter
- Interlacing
- Gamma correction

Output ImageNode and Output VideoCreator: two example modules for the DiaQuest Inc. and Silicon Graphics VideoCreator video boards. These modules serve as examples for users who wish to interface to other video output hardware.

5.2 IBM Data Explorer

5.2.1 Hardcopy

There are 3 modules in DX that will produce an image from some data: render, display and image. Render produces the image but doesn't display it while display will display it too but requires a camera module to produce transformations on the viewpoint. The image tool combines both display and a camera, in addition allows the user to save and print images. If the render tool is used it would be necessary to output the image from the render module into write image or the image tool. The write image requires the user to open its control panel and type in the name of the format (new users will need to refer to manual pages to determine what this string is and may find the control panels a little difficult to understand). The image tool uses pull down menus to select the format. Using print image under the "file" menu of the Image tool, the standard formats provided to send directly to a printer are: Colour PS, Greyscale PS, Encapsulated Colour PS and Encapsulated Greyscale PS. In addition it is possible to save images using "save image" under the "file" menu: as a PS file, in RGB (grouped together), R+G+B (all R, then all G then all B), or TIFF. Having saved the image this can be converted to other typical hardcopy image formats using a public domain image tool, e.g. San Diego Supercomputing Centre (SDSC) tools. Example macros and networks are available at the Cornell anonymous ftp site, under `pub/Data.Explorer/Extensions/Import.and.Export.Macros`. The macros automatically create a temporary tiff file, convert it to the requested format using the SDSC imconv routine, and finally delete the temporary tiff file.

5.2.2 Files

Apart from the creation of image formats, as detailed above, the only other standard module for creating output files from dx is the export module. This exports data from Data Explorer but only produces dx formatted or bin files (dx is the default, needs to have the text “dx” or “bin” in the “format” entry of the control panel). The bin format is only recommended on IBM POWER Visualization Systems. With “dx” format additional modifiers apply:

- MSB or LSB (most or least significant byte order)
- 2 formats of dx are currently supported ieee (or binary) and text (or ascii).

5.2.3 Animation/Video Creation

The primary tool used for the creation of Animation sequences is the sequencer - this can be found under the category “Special”. Given a min, max and step value (passed from other modules or set explicitly), this basically creates a sequence of integers. Although like more advanced animation utilities, the sequence can be played forwards, backwards, set to a particular value, paused, or set to loop (either to the last then jump back to the first or bounce forwards then backwards). The default interface resembles a simple VCR control panel, see Figure 11

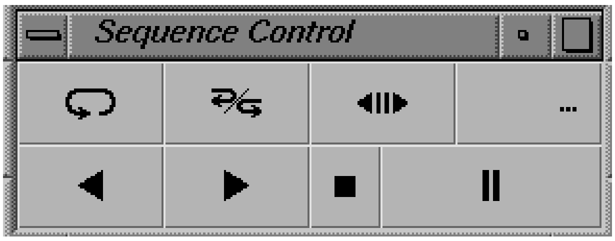

Figure 11: VCR interface of the sequencer

To set the values used by the sequencer, an additional control panel needs to be opened. By selecting the box containing “...”, the panel shown in Figure 12 becomes visible.

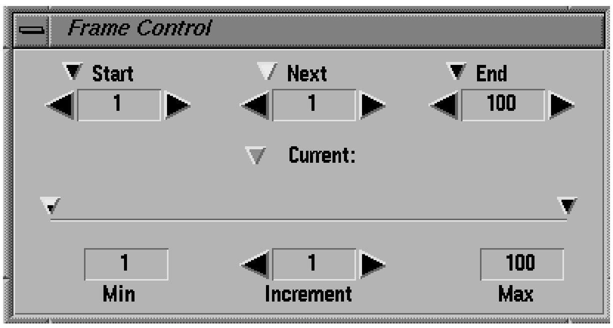

Figure 12: Additional control panel for sequencer

On its own this module is not particularly useful for general purpose animations, but in combination with others can become a powerful animation aid. For example, when used with the transformation tool Compute it can produce a sequence of float values that could modify camera position, or with the interactor Selector could cycle through time series data. Several networks exist in the standard example directory, `/usr/lpp/dx/samples/programs`, to illustrate this. Only one sequencer is allowed in a network, but more experienced users can build networks such that this produces side by side animations.

5.3 IRIS Explorer

5.3.1 Hardcopy

Hardcopy output of visual information is provided by colour Encapsulated PostScript through the Render module. Other modules producing visual output, such as Graph, Histogram, DisplayImage, do not output PostScript, though a new module NAGGraph, available soon, will do so. The Render module File menu has a Print selection with options to set an output size in inches and orientation (Landscape or Portrait), sending directly to the printer or to a file. When sending to a file it is also possible to output RGB.

5.3.2 File output

DataScribe

DataScribe is a visual programming interface for creating user-defined data readers. It takes ASCII or binary data files and turns them into IRIS Explorer Lattice data types, or vice versa. It works only to/from the various Lattice types (see Data Import section).

Image Formats

ReadImg and WriteImg module

These IRIS Explorer modules read/write a lattice containing a 2D image to a file. The lattice can have 1, 3 or 4 data variables denoting greyscale, RGB or RGB Opacity.

ImportImage and ExportImage modules

These user-contributed modules convert between an IRIS Explorer Lattice containing a 2D image and one of the following formats:

Abbreviation	Description
bmp	Microsoft Windows bitmap image file
cur	Microsoft Windows cursor image file
eps	Adobe Encapsulated PostScript file
gif	Compuserve Graphics image file

Abbreviation	Description
hdf	Hierarchical Data File
ico	Microsoft Windows icon image file
icon	Sun Icon and Cursor file
ff	Sun TAAC Image File Format
mpnt	Apple Macintosh MacPaint file
pbm	PBM Portable Bit Map file
pcx	ZSoft IBM PC Paintbrush file
pgm	PBM Portable Gray Map file
pic	PIXAR picture file
pict	Apple Macintosh QuickDraw/PICT file
pix	Alias image file
ppm	PBM Portable Pixel Map file
pnm	PBM Portable aNy Map file
ps	Adobe PostScript file
ras	Sun Rasterfile
rgb	SGI RGB image file
rla	Wavefront raster image file
rle	Utah Run length encoded image file
synu	SDSC Synu image file
tga	Truevision Targa image file
tiff	Tagged image file
viff	Khoros Visualization image file
x	AVS X image file
xbm	X11 bitmap file
xwd	X Window System window dump image file

Render

The File menu of Render can also be used to output files in IRIS Inventor format. The file can subsequently be read into a Showcase document on an SGI machine whilst retaining the 3D relationships recorded therein.

ReadXXX and WriteXXX modules

IRIS Explorer's internal data formats (Lattices, Pyramids, Geometry including camera and lighting specifications) have an associated module for writing out to a file, whether in ASCII or binary. Binary output typically is used where the file is to be read back into Explorer at a later time, using the corresponding ReadXXX module. ASCII format is useful to elucidate the data structures and for visual inspection during debugging. Storage of a colourmap is as a specific instance of a Lattice. Parameters and Pick data types can be output to the IRIS Explorer log window using PrintXXX

5.3.3 Video/Animation

The AnimateCamera module is provided to animate the viewing position. Typically, camera positions are output from the Render module and are input to AnimateCamera in *Learn* mode. In *Run* mode, AnimateCamera interpolates at user-specified intervals to produce camera geometry which is then input to Render. Camera positions can also be stored in a file and read into AnimateCamera using FileList.

Loop control modules such as For, Trigger, While*, and* Repeat allow the generation of a sequence of parameters to drive, for example, automatic isosurface calculation using the IsosurfaceLat module.

WriteAnimation writes incoming 2D lattices into a single 3D image in FIT format, which is defined and used by the IRIS ImageVision Library. This format is accepted by SGI moviemaker in which it can be edited and saved in *movie* format which is accepted by SGI movieplayer. This forms flip book animation.

The module pair WriteMovie and ReadMovie allow an SGI movie file of concatenated images to be written out and read back in frame by frame. When used in conjunction with the loop control constructs this allows flipbook animation of images.

A suite of modules is available to control an external video recorder connected to a hardware video board. The hardware supported includes the *VideoLab Board*, *Starter Video*, *VFR* and *VC* and the options available are frame rate, timing and output format, all of which can all be set from the VideoDevice module.

The VideoControl module controls an external video device and has support for V-LAN as well as VC and VFr. There are 3 modules designed to work with the Starter Video board, VideoStarterIn, VideoStarterInWin and VideoStarterOut:

- VideoStarterIn takes input video via the Starter Video board and outputs a series of user specified frames in Lattice format. Various settings may be altered including Hue and Saturation of the image as well as the scale. Both NTSC and PAL formats are supported.
- VideoStarterIn and VideoStarterInWin do the same job except that VideoStarterInWin has a drawing region on which to display single frames before they are grabbed.

- VideoStarterOut sends video output via the Starter Video board and has inputs either of a lattice or a user specified region of the hi-res display. It also supports both NTSC and PAL formats.

There are similar versions of VideoStarterInWin and VideoStarterOut for the VideoLab board called VideoLabInWin and VideoLabOut with only slightly different options.

5.4 Khoros

By default, all Khoros operators write out in the Khoros 2.0 VIFF file format. This is done by default because not all formats are capable of storing the full polymorphic data model. For example, the PBM format is only able to store map and value data. Thus, if you create a data object with explicit location data and then save the object using the PBM format, your location data will be lost. The Khoros 2.0 VIFF format is the only supported format which is capable of generally supporting all data segments and attributes. Note, however, that since most of the supported formats are designed for storing images, this is typically not a limitation if you are working with image data.

5.4.1 Hardcopy

The kformats operator can be used to convert any of the supported file formats to any of the other supported file formats.

The specific formats which are supported within Khoros 2.0:

- Khoros 2.0 VIFF (viff)
- Khoros 1.0 VIFF (xvimage)
- PNM (pnm)
- Sun Raster (rast)
- X Bitmap (xbm)
- X Pixmap (xpm)
- X Window Dump (xwd)
- AVS Image (avs)
- ARF (arf)
- ASCII (ascii)
- RAW (raw)
- Encapsulated Postscript (eps) - output only
- Support for GIF will be in available with the CD release.

5.5 PV-WAVE

5.5.1 Hardcopy

PV-WAVE CL provides a number of options for producing hardcopy output and includes support for, Postscript, HPGL, PCL, QMS QUIC, CGM, Tektronix 4510 (rasterizer(*) DEC LJ250, SIXEL and PICT.

Five steps are required to produce hardcopy output and these are (PV-WAVE CL Commands in brackets):

1. Select the hardcopy output device (SET_PLOT)
2. Configure the output device (DEVICE)
3. Create your graphics output (PLOT, PLOTS, SURFACE etc.)
4. Close the output file (DEVICE)
5. Send the output file to the plotter (Operating System Dependent e.g lpr <filename> for Unix)

Specific formats and configurations are specified through the use of keywords to the above commands. Where commands are not specified e.g. no DEVICE following a SET_PLOT, default values are taken. The command INFO can be used to obtain information about the currently selected device.

In general hardcopy output is easily created and the supplied documentation provides useful examples.

5.5.2 File output

File output from PV-WAVE CL is created by the use of the data export procedures and DC functions, which include:

- PRINT (formatted/free format write to standard output)
- PRINTF (formatted/free format write to specified file)
- WRITEU (binary unformatted)
- DC_WRITE_FIXED (ASCII fixed format)
- DC_WRITE_FREE (ASCII free format)
- DC_WRITE_8_BIT (8-bit image data)
- DC_WRITE_24_BIT (24-bit image data)
- DC_WRITE_TIFF (Tag Image File Format)
- HDF and netCDF are fully supported

Equivalent functions broadly exist for data input (please see the data input chapter).

Data import / export error messages and associated options can be specified through the functions:

- DC_ERROR_MSG (returns text string associated with error condition)
- DC_OPTIONS (set error message reporting level)

In general file output is easily created and the supplied documentation provides useful examples.

5.5.3 Video/Animation

The programming constructs available provide the mechanism to produce animation sequences, and a sample animation procedure is provided with the supplied documentation. The constructs and procedure enable simple flip book animation sequences to be created. There are also the two interactive procedures:

- WgMovieTool
- WgAnimateTool

Chapter 6: Incorporating Application Code

6.1 Introduction

For most of the visualization systems there are public domain repositories of examples and demonstrations to illustrate the inclusion of application code. The location of these sites is given in Chapter 8: Additional Information.

6.2 Application Visualization System (AVS)

6.2.1 Programming language

AVS provides direct support for the following programming languages:

- C (K & R and ANSI styles);
- Fortran 77;
- C++

6.2.2 General overview and structure

Programmers can include user-written modules into the AVS system. A module consists of source code files and an associated help page. Within the source code of the module the user specifies what input/output data ports the module has and what parameters are associated with the module.

There are two types of modules which differ in the way their execution is controlled:

- Subroutine: these modules are controlled by the AVS Kernel (Flow Executive) which will invoke a module only when its input ports or parameters have received new or changed data;
- Coroutine: these modules execute independently of AVS and explicitly inquire data on the input ports and parameter settings whenever they require this information. They also send data into an AVS network asynchronously; this could cause problems if large amounts of data are sent so coroutines have synchronisation routines made available to them.

6.2.3 Automatic generation

There are two tools associated with the code generation and user interface aspects of AVS modules:

Module generator

The user provides a specification of the module (see figure 13) they wish to construct via a number of menus. This includes information such as programming language (currently C and Fortran 77 options), input/output port definitions, parameters and widgets (slider, dials). When the specification is complete the module generator will output a skeleton source code file utilising the information specified along with a makefile, manual page and optional programming hints;

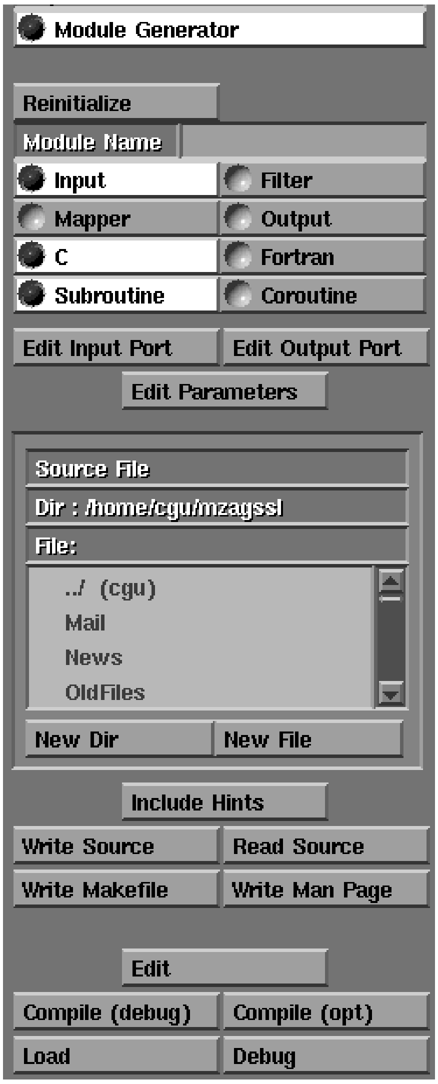

Figure 13: AVS Module Generator Interface

Layout editor

The parameters associated with a particular module all appear in a stack fashion towards the left-hand side of the network editor by default. The layout editor allows a user to interactively change, group and position these parameters to create a custom layout definition for a particular AVS network. This information is saved with an AVS network thus allowing the interface to a particular module to be customised differently in other AVS networks.

6.2.4 General topics

Application control of the module

An AVS coroutine module can execute independently of the AVS Kernel allowing it to provide a useful method of interfacing an external applications or devices into AVS.

Shared Data

AVS makes use of shared memory and Unix sockets for data communication between modules on the same local machine.

User interface

As mentioned earlier the Layout editor can be used to interactively change, position and group a modules parameters within an AVS network. There are also examples of modules which provide their own user interface through X11 and associated widgets and toolkits.

The current version of AVS (AVS5) was based upon their own graphical user interface, LUI. AVS6 will now have a Motif user interface on the workstation platforms and AVS6 is being planned for release onto PC platforms with the first release based on Windows NT and later versions for Windows 95 and DOS.

Compilation

You can perform cross compilation from within an AVS module and make calls to external subroutine libraries.

Debugging support

AVS provides mechanisms a number of mechanisms for debugging:

- An AVS module can be debugged using the native system debugging tools;
- Facilities to track down problems with memory allocation errors;
- Verification of the integrity of geometry data types a user has built;
- Modules exist which output the data representations to ASCII files;

Training

Two AVS courses, introductory and advanced, have been developed as part of the Advisory Group on Computer Graphics (AGOCG) Visualization Support Project at the Computer Graphics Unit, Manchester Computing Centre, University of Manchester.

The materials are available in postscript format along with the supporting data files and modules via anonymous FTP from the University of Manchester `ftp.mcc.ac.uk` (130.88.203.12) under the directory `pub/cgu/avs/avs_course` or the International AVS Center (`avs.ncsc.org`).

6.3 IBM Data Explorer

6.3.1 Programming language

C is the main language but using C wrappers both Fortran and C++ code can be incorporated into modules.

6.3.2 Overview of modules

There are two specific types of modules that can be implemented:

- Inboard: these are linked with the existing DX executive and do not require any socket communication;
- Outboard: these are not linked with the DX executive and can sometimes be easier to manage than the Inboard modules. Outboard modules are easier for Application Developers to ship extensions;

The API that is made available to users of DX is the same one used by the IBM DX development team.

On SGI & SUN Symmetric Multi Processor (SMP) machines such as Onyx DX modules will run in parallel within a module. Using data decomposition the DX executive spawns tasks on each of the available processors to execute the module function against a sub-area of the dataset. The sub-areas are grown into each other to cater for boundary conditions.

This is an important area for DX as it was originally written to run on a parallel machine.

6.3.3 Automatic generation

DX has a module building tool that will automatically generate most of the C code, along with the modules description file and makefile. To run this the user needs to type:

```
dx -builder
```

This is an easy to use tool, for example Figure 14 shows how the builder was used to outline a transformation module that takes 1 input and 1 output.

Apart from describing the data types so DX can set up the appropriate data, the user also includes comments to describe what the module does and what the input and output data are.

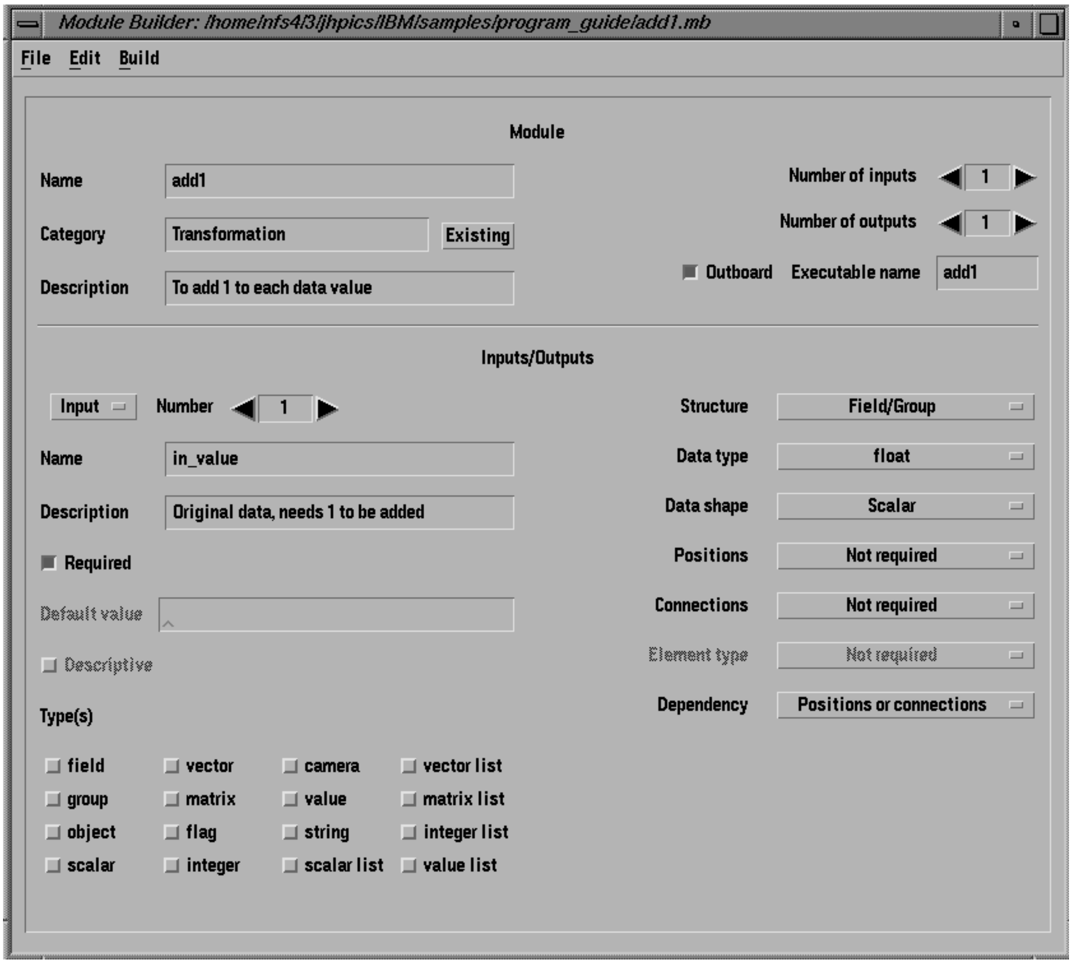

Figure 14: IBM Data Explorer module builder interface

A text version of this description can be saved for future use:

```
MODULE_NAME = add1
CATEGORY = Transformation
MODULE_DESCRIPTION = To add 1 to each data value
OUTBOARD_EXECUTABLE = add1
OUTBOARD_PERSISTENT = FALSE
ASYNCHRONOUS = FALSE
PINNED = FALSE
SIDE_EFFECT = FALSE

INPUT = in_value
DESCRIPTION = Original data, needs 1 to be added
REQUIRED = TRUE
STRUCTURE = Field/Group
DATA_TYPE = float
DATA_SHAPE = Scalar
```

```
POSITIONS = Not required
CONNECTIONS = Not required
ELEMENT_TYPE = Not required
DEPENDENCY = Positions or connections

OUTPUT = out_val
DESCRIPTION = in_value with 1 added
STRUCTURE = Field/Group
DATA_TYPE = float
DATA_SHAPE = Scalar
POSITIONS = Not required
CONNECTIONS = Not required
ELEMENT_TYPE = Not required
DEPENDENCY = Positions or connections
```

The Module Description file generated is as follows:

```
MODULE add1
CATEGORY Transformation
DESCRIPTION To add 1 to each data value
OUTBOARD add1;
INPUT in_value; group; (none); Original data, needs 1 to be added
OUTPUT out_val; group; in_value with 1 added
```

The Makefile:

```
FILES_add1 = add1.o
BASE = /usr/lpp/dx
CFLAGS = -Dindigo -O -I$(BASE)/include LDFLAGS= -L$(BASE)/lib_indigo
LIBS = -lDX -lsun -lgl_s -ly -ll -lX11 -lm -lmpc
OLIBS = -lDXlite -lm

add1: $(FILES_add1) outboard.o
  $(CC) $(LDFLAGS) $(FILES_add1) outboard.o $(OLIBS) -o add1

# how to make the outboard main routine

outboard.o: $(BASE)/lib/outboard.c
  $(CC) $(CFLAGS) -DUSERMODULE=m_add1 -c $(BASE)/lib/outboard.c

# make the user files

useradd1.c: add1.mdf
  mdf-c add1.mdf > useradd1.c
```

The C code generated was over 500 lines and so too long to list here. This may appear to be very long, but this is well commented code and the user is only required to modify the tail end of the code:

```
int add1_worker
  (
```

```
     int in_value_knt,
     float *in_value_data,
     int out_val_knt,
     float *out_val_data
     )
  {
     /*
     * The arguments to this routine are:
     *
     *
     * The following are inputs and therefore are read-only.The default
     * values are given and should be used if the knt is 0.
     *
     * in_value_knt, in_value_data: count and pointer
     * for input"in_value"
     * no default value given.
     *
     * The following are outputs and therefore are writable.
     *
     * out_val_knt, out_val_data: count and pointer
     * for output "out_val"
     */

     /*
     * User's code goes here
     */
  }
```

When 2 inputs were used the number of lines of code increased to over 650 lines but again the user is only required to insert code into the *worker* routine.

6.3.4 Further examples

There are further examples which can be found in the IBM Data Explorer Programmer's Reference Manual. There is also an FTP site of public domain modules available `ftp.tc.cornell.edu` under `pub/Data.Explorer.`

6.4 IRIS Explorer

6.4.1 Programming language

The programming languages that are supported directly by IRIS Explorer are:

- C
- Fortran77
- C++
- Fortran90: The addition of Fortran 90 to this list is currently under consideration at NAG, although the timescale would be determined by the degree of user interest.

6.4.2 General overview and structure

Modules are written as a user function and associated subroutines called from the user function. These are then compiled and automatically embedded within an Explorer wrapper. The input and output ports of the final module are expressed as mappings from the parameters of the user function to Explorer data types. It is possible to create *hook* functions that can be called in specific circumstances i.e. the initialisation hook function is called when the module is launched into the map and is used to do initialisation for the module.

Module compilation is done using a graphical interface called Module Builder (mbuilder) which requires the name of the source code file, the Module name, any flags, any Libraries and any User Makefile. There are then subsequent visual interfaces for:-

- Input Ports: List the names, types and optionality (required or optional) of input ports. Data must be present on all required input ports before firing.
- Output Ports: List the names and types of the output ports.
- Function arguments: List the names and types of the arguments to the user function. Also specify their type, either scalar or array (or pointers to either), the return type (if any) and the language.
- Connections: This lists the input ports, function arguments and output ports in 3 columns and allows the user to wire the input ports to the required function argument and then to the output port. It also allows subtypes and structure members of the root type on the port to be wired either to a function argument or, if left unaltered by a module, direct to an output port. I.e. It is possible to wire the data portion of a lattice into a function argument, do some calculation on it, then wire it to the data portion of an output lattice while just wiring the coordinate portion directly to the output.
- Control Panel: See User Interface later in this section.

Once all aspects of the module have been entered into the Module Builder, selecting build will cause it to begin compiling the module. On building, a set of makefiles are produced and all error messages are returned to the screen. Once these makefiles are produced the module can be recompiled externally to Module Builder by using UNIX make. Building a module not only produces the executable, but also the C source code for the interface between the user function and the remainder of Explorer.

Explorer uses three controlling interfaces or *wrappers* to mediate between the module's user function and the rest of the system. These are the Module Control Wrapper (MCW), the Module Data Wrapper (MDW) and the Generic Wrapper. All three of these are generated automatically on building, though the MDW can be turned off but the user must then incorporate the MDW functions in their own code. The MCW is the module data manager, contains the firing algorithm and provides interfaces to input and output functions, including port and widget settings. The MDW performs conversions between Explorer and user-defined data types on the ports and the data format required by the user. The user can thus develop modules with customised interfaces without having to interact with Explorer data types. The Generic Wrapper is not accessible to the module writer, except indirectly through Hook Functions menu.

As an alternative to writing C/Fortran code and having to build, link and compile after each alteration, Explorer provides a method of generating user defined modules that act on the Lattice data type, without having to use Module Builder. The LatFunction module is an interpreter for a lattice manipulation language called Shape that makes it easy to operate on Lattices. It allows the user to interactively change the way LatFunction acts on the data it receives, which provides a quick way to prototype a new module without having to actually compile it. The user passes the name of the Shape program file as a parameter to the LatFunction module. Shape is an interpreted, array-oriented language with a C-like syntax which makes it easy to create, process and manipulate lattices. Since the program is interpreted, changing the output lattice can be done by changing the name of the file passed to LatFunction - i.e. a new module doesn't have to be launched every time the lattice needs to be changed. This makes LatFunction useful in prototyping or testing new modules.

How is control passed to/from a module?

The modules are all free to fire independently of each other, the only constraint being that they have the data on their input ports that they require. In practice this means that control is passed from upstream modules to downstream modules as data (or parameters) are generated/altered and passed on.

How is data passed to/from a module?

Data flow between modules is specified by the user by wiring modules in the Map Editor together. The connections are made from the output port of one module (always on the right of a module control panel) to the input port of another module (always on the left). This is done by pointing and clicking over the relevant port which pops a list of available output ports, selecting one of these then causes the other modules with compatible input ports to be highlighted. A *wire* is then displayed in the map editor linking the two modules together. On workstations supporting shared memory only the pointers are copied, not the data itself.

6.4.3 Automatic generation

The tool for this with Explorer is Module Builder (see figure 15), used as described above, filling in details of ports and function arguments to describe the flow of data through the module. However, instead of using the build option, "Create function prototype" is selected from the Prototypes menu and a source code template is created with gaps for the user to write their function.

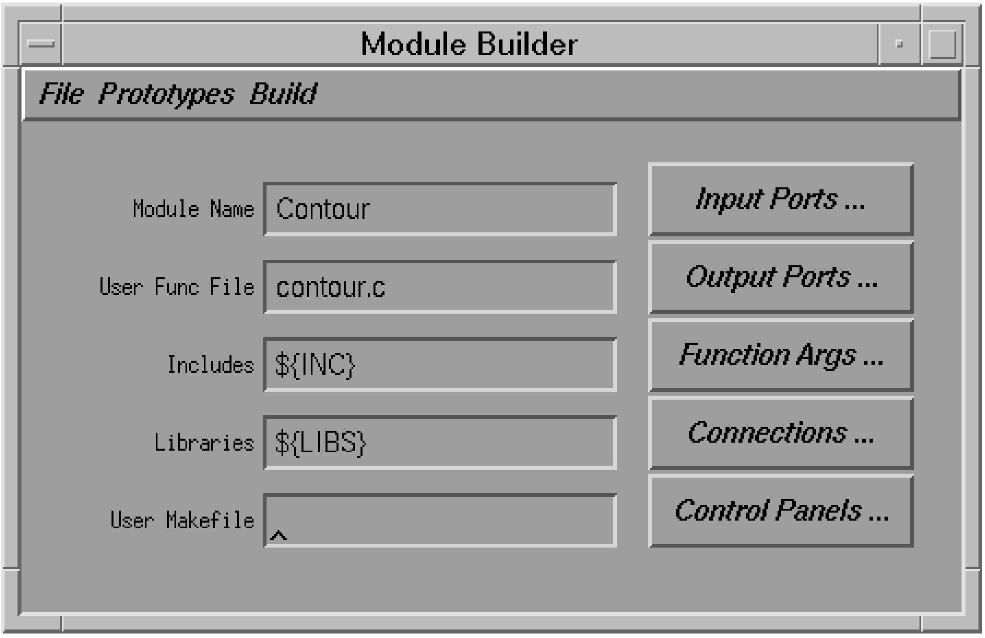

Figure 15: IRIS Explorer Module Builder Interface

One of the menu options on mbuilder is for creating document prototypes. This option produces a standard document file with all the input/output data structures listed, also, mbuilder automatically produces the template for the module help files in the standard format with all the port data types listed as for the doc pages. Gaps are left for the user to fill in specific details of the module and port descriptions using any standard editor.

6.4.4 General topics

Application control of the module

The firing algorithm in the Module Control Wrapper (MCW) is responsible for determining when the module is ready to fire. It is based on whether all the required frames of data are present on the input ports. The user can make a module fire by selecting the fire option from the pop up menu associated with each module. Explorer provides a means of registering file descriptors with the Explorer Kernel through the API function `cxInputAdd`. This descriptor is then *watched* by Explorer and when there is data to read a callback function is called. This in turn can then request that its associated user function be fired by calling the API function `fireASAP()`. Through this method it is possible to connect to external UNIX sockets and hence to other applications/devices.

Remote execution

In IRIS Explorer this is achieved by simply calling up a module Librarian on an alternative machine. This is done through the host pull down menu which gives a dialogue box into which the host name is entered. This new Librarian is used in the same way as the local one where modules are selected from it and placed in the map. Explorer takes care of the rest. These modules are marked with the name of the machine they are running on.

Shared Data

Explorer uses a shared memory model for storing data. This allows modules to share a single copy of, say a Lattice, which remains in memory until it is no longer being referenced by any modules. This is achieved by means of a reference count stored within the data structure.

User interface

In Module Builder there is an option for editing the control panel of a module. Selecting this gives a visual tool for designing and arranging the widgets associated with a module. It is possible to set size, default values, positions, visibility, sizeability and widget type. Also it is possible to add text labels, frames, and separators. The widgets are associated with the user function by expressing them in the argument list as cxParameter types. The P-func Editor can be used to calculate an input parameter value from one or more connected parameters on other modules in the map.

There is an option to build a module with X-MCW which uses the X mechanism for handling scheduling. Using this option allows Explorer to recognise and manipulate X Window or Motif widgets in a module. The advantages of this are that you have complete freedom in creating the user interface.

At a higher level, it is possible to group a set of modules that make up a map and edit a control panel made up from some (or all) of the widgets available from all the modules in the group. This includes the graphical output windows of DisplayImg and Render. This is done from within the map editor and can be saved as a map. This map can then be run in application mode which is where the map editor and librarian are unused and just the user defined control panel is visible.

Compilation

Module Builder allows the cross compilation of C and Fortran and also the inclusion of object code. All the required files are listed in the text box “User Func File” as a space separated list. Mbuilder needs the language type of the actual user function, but the rest are defined by their file extension. External libraries can be linked by inserting them in the Libraries text slot in the same way they are listed in the UNIX link command. It uses the form `-l<library>` for individual libraries.

Debugging support

Debugging support is provided by compiling the module with an environment variable DEBUG set to 1 to force the use of the `-g` flag. A module is then launched into a map and any of a number of debuggers can be attached through the modules process ID.

Explorer does some runtime checking of data and there are a few options as to the course of action taken if an error is found. These are selected at run time from the `-action <INT>` option where

- 0=ignore,

- 1=warning,
- 2=halt,
- 3=abort

and `-level <0/1>` which sets the level of data checking.

Training

The module writer's guide contains an initial chapter going through building a module with a fairly trivial example (the source of which can be found on line) explaining what the basic elements of Module Builder do. The next chapter runs in parallel with the first and gives a more general overview of Module Builder. There are chapters for each of the data types with examples of code in both C and Fortran. In addition to the Module Writer's Guide there are UNIX man pages in 2 sections, the first covering all the standard modules, the second covering all the API routines.

There are also available courses from NAG which include an introductory and advanced course. For more information you should contact NAG directly.

6.5 Khoros

Khoros 2.0 was designed to be a software development environment, so the support for incorporating new application code is very strong. Khoros is composed of toolboxes, which are simply collections of programs and libraries. Additional application code can be easily added to the system by creating new programs and libraries within a new toolbox [38] [39]. Since the Khoros system comes with complete source code, programming examples are always handy.

6.5.1 Programming language

Khoros is based on the C programming language with support for compiling with Fortran and C++. Support for sh, ksh, csh, and perl scripts as program objects is also available.

6.5.2 General overview and structure

In Khoros, application code is contained within distinct software objects. A software object a complete encapsulation of a program or library and consists of application source code, documentation, user interface information, and configuration. All software objects except libraries can be run independently from the command line, or from the visual programming language Cantata. The following types of software objects can be created.

- *kroutine*: A kroutine object is a non-interactive data processing routine.
- *xvroutine*: An xvroutine is an interactive program with its own graphical user interface.
- *pane*: A pane object is a special type of program object which is simply an alternate user interface for an existing program. Pane objects may be used as a Khoros wrapper for the easy integration of existing non-khoros program.
- *script*: A script object is a utility program which is written in some scripting language. The user interface of the script must be written and maintained by the shell script programmer.

- *library*: A library object is simply a collection of functions which can be called by other libraries or programs.
- *workspace*: A workspace object is a Cantata workspace which has been collapsed down into a software object. Note that a workspace object can also be run from the command line, as well as within Cantata.

6.5.3 Software lifecycle

The Khoros software development tools provide a complete environment which support the iterative process of developing, maintaining, delivering, and sharing software. These tools act as the programmer's assistant by providing automation where possible, enforcing consistency as necessary, and hiding underlying complexity of software configuration, code generators, and documentation formatters. This functionality is provided in the form of three applications which are described below.

- *Craftsman*: The Craftsman application is used to manage toolboxes and software objects. Toolboxes can be created or deleted from here. Software objects within any given toolbox can be created or deleted from here. Editing of a software object is initiated here by launching the Composer application.
- *Composer*: The Composer application is used to edit, manipulate, and compile existing software objects. Composer lets you edit the different parts of the software object such as the source code and the documentation. The user interface of the software object can be interactively edited by launching the Guise application.
- *Guise*: The Guise application is used to interactively create graphical and command-line user interfaces. These applications are all used in a dynamic way, allowing a developer to move easily between the different stages of development.

6.5.4 General Topics

Application control of the module

Each program object is an application unto itself giving full control to the user.

Shared data

Khoros provides data transport between processes via files shared memory, pipes, and streams.

User interface

As mentioned earlier, interactive editing of Khoros user interfaces is done with the Guise application. The resulting user interface will be created at run time using widgets from one of the supported widget sets (Athena, Motif, OLIT).

Compilation

Compilation of Khoros software objects is performed at a button press from the Composer application.

Debugging support

Khoros has no actual debugger, but provides compilation rules for linking against Purify from Puresoft and CodeCenter from Centerline.

6.5.5 Training

Khoral Research Inc. offers a “Khoros 2.0 Software Development Course” which provides a complete tour of the Khoros system, detailed coverage on each of the Programing Services, as well as the Khoros Software development system and how to write new Khoros routines.

E-mail `ktraining@khoros.unm.edu` for more information.

6.6 PV-WAVE

Application code can be incorporated into PV-WAVE CL at two levels, via user developed command language procedures or more appropriately user developed C or FORTRAN code. The description here will concentrate on the incorporation of user developed C or FORTRAN code as the incorporation of command language procedures is considered an integral part of the PV-WAVE CL package.

Two supplied PV-WAVE CL procedures, SPAWN and LINKNLOAD enable application code to be incorporated into PV-WAVE CL and support the following languages:

- SPAWN: C and FORTRAN
- LINKNLOAD: C and FORTRAN

The SPAWN procedure provides a simple and efficient way to augment PV-WAVE CL with user developed C and FORTRAN code, and moreover provide a general mechanism for accessing the operating system. The procedure SPAWN spawns a child process to execute the application code, to which it communicates via a Unix pipe. Various input and output keywords are used to specify precise requirements. Where possible it is recommended that the required operation is performed entirely within PV-WAVE CL, as PV-WAVE CL is likely to be more efficient, particularly will small datasets.

The LINKNLOAD procedure is used to access external functions in an external shared object i.e. the external code must be linked as a shared object. LINKNLOAD is more efficient than SPAWN when communicating with application code and is the simplest method for attaching application code (C and FORTRAN) to PV-WAVE CL. Given that external objects must be linked as shared objects, LINKNLOAD is only available on systems capable of supporting dynamic linking.

Useful examples are provided in the supplied documentation, although the manual descriptions do tend to be rather technical.

Chapter 7: Distributed Support

7.1 Application Visualization System (AVS)

7.1.1 Remote module execution

AVS supports the remote execution of modules on heterogenous platforms. To transfer the data between machines the external data representation (XDR) is used to represent the data and the Unix socket mechanism to transport the data. If the machines both use the same data representation then the XDR translation is bypassed. All this is made transparent to the user and remote modules are used in the same fashion as local modules in an AVS network.

7.1.2 Remote access

AVS can be executed on a machine as an X client with the display set to point to a remote X server (display). When the geometry viewer tries to start up on the remote display it will try to initialize a hardware renderer of the same type on that machine. Currently only PEX/PEX and DGL/DGL pairings are supported. The software renderer within AVS can be used on remote 8 bit colour displays and is independent of the hardware capabilities of the remote machine.

7.2 IBM Data Explorer

7.2.1 Remote module execution

Modules are grouped into Execution Groups which can then be directed to run on any machine accessible to the user. The distribution can then exploit a mix of machine architectures. As with all distributed processing care must be taken not to require too much data to be passed between machines.

7.2.2 Remote access

Since IBM Data Explorer is based on the X Windowing System and Motif it can be accessed as a remote client with its display pointed to a local X server (display).

7.3 IRIS Explorer

7.3.1 Remote module execution

In IRIS Explorer this is achieved by simply calling up a module Librarian on an alternative machine. This is done through the host pull down menu which gives a dialogue box into which the host name is entered. This new Librarian is used in the same way as the local one where modules are selected from it and placed in the map. Explorer takes care of the rest. These modules are marked with the name of the machine they are running on.

7.3.2 Remote display

Version 2.0 of IRIS Explorer allows remote display via the `xhost/setenv DISPLAY` mechanism, but this requires the remote machine to be SGI with the proprietary graphics GL. This limitation is fixed at V2.2, which uses the de facto standard OpenGL. IRIS Explorer provides a module RemoteRender which allows the user to display geometry on another workstation in addition to their own, provided that the second machine has remote X connections allowed, and subject to the proviso mentioned above.

7.3.3 Printing

The Print menu of Render automatically offers the names of networked printers on which to produce output.

7.4 Khoros

The Khoros installation guide provides more detailed information [35]

7.4.1 Remote module exectution

A remote daemon, phantomd, is executed on the remote machine and handles all the requests to execute a remote task and the transporting of data to/from the remote machine. The transports supported for the transferring of data to remote machines include sockets and Sys V Transport Layer Interface (tli). In Khoros 2.0, local processes now communicate through a local phantom daemon. The local phantom daemon then communicates with remote phantom daemons. This method has the benefit of enabling more advanced distributed computing features to be added such as load balancing, security, encryption, process groups. This functionality is not availble in the current ftp release of Khoros 2.0, but is expected to be on line for the CD release next year.

7.4.2 Remote access

Khoros, since it is based on X, can be accessed as a remote client with its display pointed to a local X server (display). Khoros supports all visual types of X servers.

7.5 PV-WAVE

7.5.1 Remote execution

PV-WAVE does not provide true distributed support as available within AVS and IRIS-Explorer (i.e the ability to run specific procedures as part of the system but on different machines). However, various methods of interapplication communication are provided within the PV-WAVE CL environment, and these include:

- External program execution from within PV-WAVE CL (see also Module writing)
- Execution of PV-WAVE CL from within a C or FORTRAN program using the functions CWAVE or CWAVEFOR (without data transfer)

- External function call from within PV-WAVE CL via dynamic linking of shareable objects (see also Module Writing)
- Execution of PV-WAVE CL from within a C or FORTRAN program (with data transfer) via static linking.
- Client Server capability to remote machines via RPC's

7.5.2 Remote Display

Distributed support in the form of X Windows client server access is supported within PV-WAVE CL.

Chapter 8: Additional Information

8.1 Introduction

This chapter is intended to provide information on:

- Availability: includes cost, platforms supported and contact address for the products;
- Support
- Information: includes on-line help facilities, manuals, network services;
- User Groups

The information below may have changed since the compilation time of this report and the appropriate vendors should be contacted for the most up-to-date information on cost, special deals, etc. Some of the price information and reference to CHEST deals is only applicable to UK Higher Education Institutes. Commercial users and academic sites outside the UK should contact the vendors for price and availability information.

8.2 AVS

8.2.1 Availability

General information on availability can be found on WWW at:

```
http://www.avs.com/
```

The International AVS Center, North Carolina Supercomputing Center, US can also be accessed on:

```
ftp:avs.ncsc.org
http://www.mcnc.org/HTML/ITD/IAC/IAC.html
```

Supplier

The UK office of Advanced Visual Systems Inc is:

AVS/UNIRAS Ltd.
Montrose House
Chertsey Boulevard
Hanworth Lane
Chertsey
Surrey KT16 9JX

Tel: 01932 566608
Fax: 01932 568842
Email: `support@avs.com`

Platforms

A brief summary at time of writing is given here, but please see the following URL for current information. Workstation platforms supported are:

- Data General
- DEC Alpha AXP, MIPS and VAX
- Evans & Sutherland
- Hewlett Packard PA-RISC
- IBM RS/6000
- Kubota Denali
- Silicon Graphics
- Sun Microsystems SPARC and also on supercomputers e.g., CS-6400
- Convex
- Cray Y-MP
- Fujitsu VPX
- Intel
- Meiko
- NEC
- Thinking Machines

The next release of AVS (AVS6) will be released on PC platforms including both Windows NT and Windows 95.

For more up-to-date information access the URL:

`http://www.avs.com/products/avs.html`

Costs

The following is an extract from the NISS Bulletin Board; refer to the Bulletin Board for full details. CHEST and AVS/UNIRAS Ltd. have concluded an Amendment to the CHEST/AVS Agreement which offers a per-platform licence, as an alternative to the multi-platform licence originally offered. This deal is applicable only to UK Higher Education Institutes.

The annual charges are as follows:

- Multi-Platform Licence: £4,800 for each year that the Site is licensed; and: AVS Animator £900 per site licence.
- Per-Platform Licence: £2,150 for first platform ordered: And: £1,150 for each subsequent platform ordered; and: AVS Animator site licence £400 for first platform ordered. And: AVS Animator site licence £225 for each subsequent platform.

8.2.2 Support

UK Support Arrangements

The following is an extract from the NISS Bulletin Board; refer to the Bulletin Board for full details.

1. Telephone Advice and Support

   AVS/UNIRAS will provide telephone advice and support to the nominated contact and/or his/her deputy at licensed sites during normal business hours.

2. Training

   Training courses will be provided in accordance with prices in force from time to time. Current charges are £195 for a one day course and £350 for the two day course. AVS/UNIRAS Ltd. will provide courses on site but prices will be subject to negotiation.

3. On site Support

   Special consultancy and training, on site, current cost is approximately £590 per day.

International AVS Center

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. Located at the North Carolina Supercomputing Center, the worldwide clearinghouse collects, ports, and distributes user-contributed, public-domain modules and acts as liason between users and vendors. The International AVS Center also publishes a quarterly magazine called AVS Network News and a yearly module catalog. It also hosts the yearly International AVS User Group conference and coordinates User Group activities.

The AVS Consortium is made up of seven AVS sponsors and two affiliates who are funding and providing direction for the International AVS Center. The seven sponsors are Advanced Visual Systems Inc., Digital Equipment Corporation, IBM, Hewlett Packard Company, Kubota Computer Corporation, Kubota Pacific Incorporated, and Sun Microsystems. The two affiliates are Mobile Research and Development and Oki Electric, Inc.

International AVS Center North Carolina Supercomputing Center 3021 Cornwallis Road Research Triangle Park, NC 27709 Phone (US) 919-248-1100

The main email alias for the center is `avs@ncsc.org`

8.2.3 Information

Manuals

- AVS Applications Guide
- AVS Chemistry Developers Toolkit Guide