Chapter 1: Overview

1.1 Introduction

One of the responsibilities of the UK Advisory Group on Computer Graphics (AGOCG) is to stimulate and support the effective use of computer-based visualization.

AGOCG therefore requested DRAL Rutherford Appleton Laboratory and the Universities of Leeds and Manchester and NERC Computing Services (Keyworth) to conduct a review of visualization software, to help potential and actual users make effective choices.

AGOCG conducted a previous evaluation of visualization software [15] and the Stichting Academisch Rekencentrum Amsterdam have also produced an evaluation report [10]. At that time, AVS was already becoming available on many workstations, IRIS Explorer and IBM Data Explorer were quite new, Khoros was available free but was restricted to handling images, and apE was undergoing a difficult transition.

One consequence of that evaluation was that AVS was made available to universities and other HEI's by CHEST on favourable site terms.

Why revisit this work after what is a comparatively short time? There are several reasons:

- Revisiting this work was envisaged at the time as it was realised that IRIS Explorer and IBM Data Explorer were new in 1991/2 and it was not possible to evaluate them effectively.
- Since that evaluation IRIS Explorer is now developed and supported by a UK company (NAG Ltd.), moreover a company with strong links with the UK academic community.
- Interest in PV-WAVE has grown in that time. This system is general purpose but at the same time its command language system provides an interesting contrast to the other systems which are in general of the data-flow type.
- Since that evaluation, AVS Inc has taken over the UNIRAS company - which means that CHEST deals with these two distinct companies are now with only one.

The purpose of the work is therefore to review the current market for visualization software as it exists in the UK and to provide information to the UK Academic community for its visualization needs. There are many potential systems and the review had to narrow the field.

The review does not include basic plotting systems. These are already understood in the community. Thus any system under review had to at least treat 3D data adequately. Therefore there had to be:

- good, interactive support for viewing 3D geometry, taking advantage of 3D graphics hardware now widely available to accelerate 3D viewing
- good support for visualizing 3D data - (a common pitfall here is the often quoted 3D plot, which is often a plot of z=f(x,y) a single valued function of 2 variables which we would

therefore regard as 2D data)

- The system under review must be general purpose and extensible

1.2 Systems under review

The following systems which were reviewed are listed below (in alphabetical order):

- AVS: Commercial product from AVS Inc., supported in the UK by AVS/UNIRAS Ltd. AVS version 5.01 was reviewed and some comments about version 5.02 and the contents of version 6 have been made.
- IBM Data Explorer: commercial product from IBM Inc., supported in the UK by IBM(UK) Ltd. IBM DX version 2.0 was reviewed.
- IRIS Explorer: commercial product initially from Silicon Graphics Inc., bundled with Silicon Graphics workstations. It is also available from NAG Ltd. on other workstations and is supported by them (not bundled). It has been announced that the next version (3) will be unbundled from Silicon Graphics and available from NAG Ltd. IRIS Explorer version 2.2 was reviewed.
- Khoros: commercial product from Khoral Research Inc., distributed via a Free Access License. Currently the version 2.0 Developer's Release is only available via anonymous ftp. The Khoros Software, including a 7 volume manual set, Khoros 2.0 source code and the most commonly used binaries on CD-ROMs, will be distributed worldwide by Prentice Hall, Inc. in the second quarter of 1995. Khoros 2.0 (Beta release) was reviewed.
- PV-WAVE: commercial product from Visual Numerics Inc., supported in the UK by Visual Numerics Ltd. A related system IDL (`http://sslab.colorado.edu:2222/projects/IDL/idl_ssl_home.html`) is also available in the UK from Floating Point Systems Ltd. IDL was resold by IMSL before IMSL merged with Precision Visuals to form Visual Numerics. The syntax of PV-WAVE and IDL is similar but the two products have taken separate development paths for the past seven years. We do not cover IDL in this review because of the similarities with PV-WAVE. PV-WAVE version 4.2 was reviewed.

Chapter 2: Getting Started

2.1 Introduction

The following sections provide information for the novice user of each system. In some cases the manuals supplied with the system include suitable tutorial examples and these are subsequently referenced. Other systems have some simple examples included below. The notes assume that the particular system has already been installed correctly on the users target platform.

2.2 Application Visualization System (AVS)

2.2.1 Starting up the system

The command to run AVS is

```
avs
```

Typing this command should start up the copyright notice and result in one window down the left of the screen. If it does not, you might need some extra information in your environment variable PATH. Consult your System Manager for this.

2.2.2 Use a sample network

Place the mouse cursor (a pointing hand) over the panel "Network Editor" and click the left button once. This results in a small window on the left, the Network Control Panel, and a larger one to the right, the Network Editor (see figure 1). In AVS, complete networks do the work of visualizing data and are made up of modules connected together, each module carrying out a specific task. In the Network Editor, the top left hand section is called the main menu, and pressing a button here brings up suboptions. You should find on starting that the main menu option Network Tools is active. Next to the menu options is the module pallette, containing all AVS's individual modules, accessible by scrolling up and down the top of each column.

To use a sample network, place the mouse cursor over the Read Network suboption and click the left mouse button once. A file browser appears showing directories into which you can move by clicking the left mouse button once. Move into the examples directory and click once over `bluntfin1.net`. The file browser disappears and the network builds automatically. The network reads 3D data representing the flow of air over an aerofoil and displays the pressure in the window labelled AVS Display Pixmap. The image can be manipulated by placing the cursor in the window and pressing the middle mouse button whilst moving the mouse. For more information on manipulating scenes produced from AVS networks you should consult the AVS Tutorial Guide Ch 2: Geometry Viewer Tutorial. This tutorial covers the use of the Geometry Viewer module which is similar in functionality to the coupling of Render Geometry and Display Pixmap used in `bluntfin1.net`.

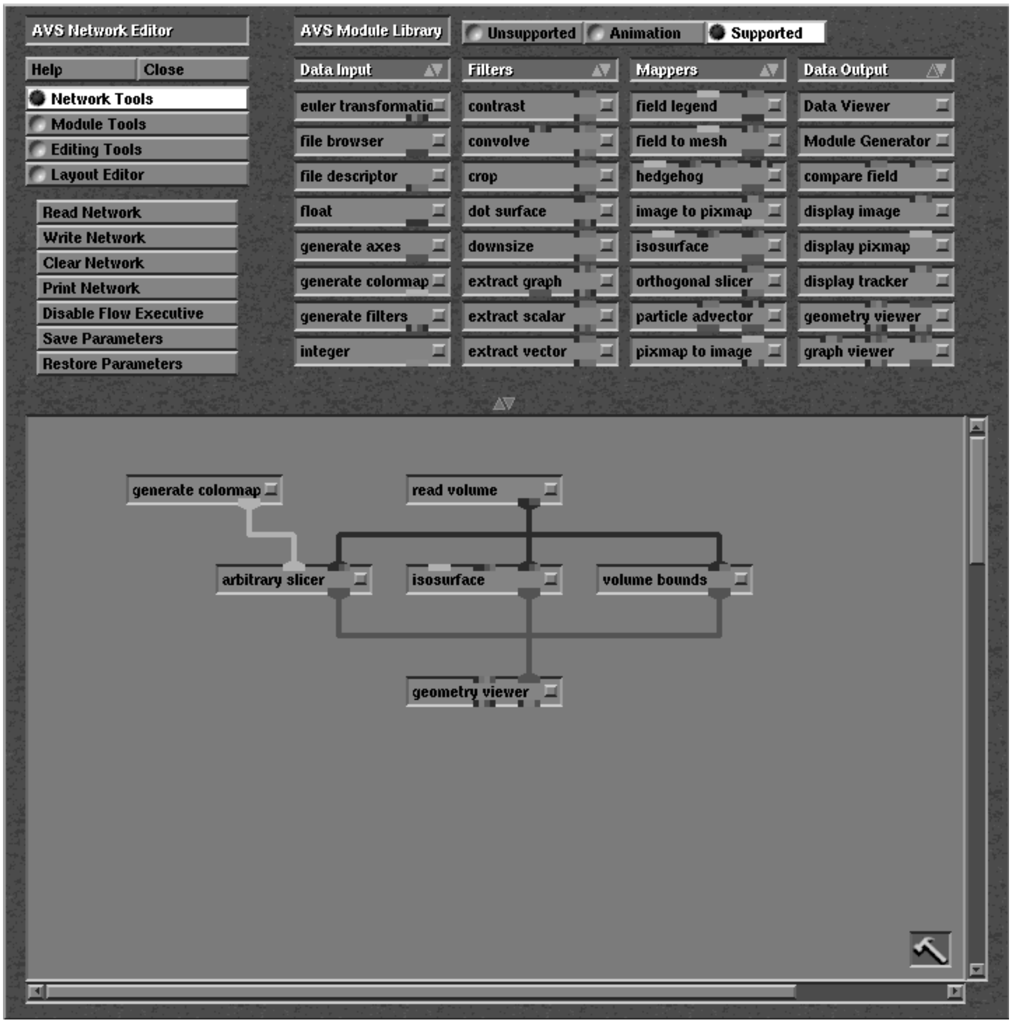

Figure 1: The AVS Network Editor

2.2.3 Build your own network

- First you need to clear the bluntfin network: select the suboption “Clear Network” and confirm.
- Scroll down the data input column until you see Read Image, place the cursor over the panel, press the left button and keeping it pressed, drag the module onto the blank workspace below. Release the mouse button and the module appears on the workspace: this combined operation is called module instantiation.
- In the same way, instantiate the modules Crop from the Filters column and Display Image from Data Output. You now have three modules in the workspace.
- To connect them together, move the cursor onto the coloured output port of Read Image,

press the middle mouse button and while keeping it pressed, move to the input port of Crop. Release the button and the connection will appear in blue. Do the same between the output port of Crop and the input port of Display Image.

- Move to the Network Control panel and click on Read Image. Read Image's file browser appears and you should use the scroll bar and mouse to move into the directory `Image`, and select the file `mandrill.x`. The module Read Image now imports the data in the file `mandrill.x`, passes it to the Crop module to reduce the size of the data and thence to Display Image which shows the results, behind the Network Editor.
- To see the image, press "Close" on the main menu. To see the parameters of the Crop module, click on "Crop" in the Network Control Panel. To set a new value on any of the dials simply press with the left mouse button on the pointer, drag it to the required position and release the mouse button.
- To return to the Network Editor press the "Display Network Editor" button in the Network Control Panel. Before exiting AVS you should clear the network as before and then press "Exit" on the Network Control Panel. Then on the main AVS panel press "Exit AVS" and confirm.

2.2.4 Carrying on

The data you have used so far was AVS's own internal data format. You will want to import your own data into the system and for this you will need to use the AVS Data Interchange Application (ADIA). More information on ADIA is available in the AVS Applications Guide [1] Ch. 1.

To find out more about individual modules you can run other demos in AVS. These are explained in the AVS Tutorial Guide [6], Ch. 1

Online help on AVS can be accessed from within the system using the help buttons in both the Network Editor main window and the Network Control Panel. Paper manuals comprise the AVS Users Guide [8], Developers Guide [3], Tutorial Guide [6], Module Reference [4], Applications Guide [1], AVS5 Update manual [7], Chemistry Developers Toolkit Guide [2] and AVS Technical Overview [5].

2.2.5 AVS Training Materials

What do the materials contain?

The training materials contain slides and notes for two courses, the AVS Introductory [43] and Advanced course [42]. These were initially 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.

There are also a number of data files and modules which support the practical exercises described in the course notes.

How do I obtain the materials?

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`) or the International AVS Center (`avs.ncsc.org`).

For example to obtain the materials from the University of Manchester you would first type the following:

```
ftp ftp.mcc.ac.uk
```

When you are connected to the server you should login in as anonymous and supply your email address as the password. To access the training materials you must move to the subdirectory pub/cgu/avs/avs_course and set the transfer to binary mode before getting the files:

```
ftp> cd pub/cgu/avs/avs_course
ftp> binary
ftp> mget *
```

2.3 IBM Data Explorer

A good way to start with Data Explorer (known as dx) is to follow the tutorial described in Appendix A of the User Guide [26].

The tutorial is in a number of separate parts - each can be followed independently of the others. It makes use of some files which should have been installed with the system. If these cannot be found, it is possible that the path names are different on your system - consult your system manager.

2.3.1 Starting up the system

At the UNIX prompt, type

```
dx
```

You should see the message below as dx is started:

```
Starting DX user interface
```

2.3.2 Building a Network

Appendix A.2 in the User Guide contains a tutorial which shows how to construct a simple network that reads in data, generates an isosurface, and displays it. It also shows how to save the network to file, and deals with the control of parameters (in this case the isolevel).

One thing it does not tell you is how to correct an error in wiring up the network: to remove a connection, click on the *destination* port and drag away the connection before releasing the mouse button. Figure 2 shows the construction of a simple network.

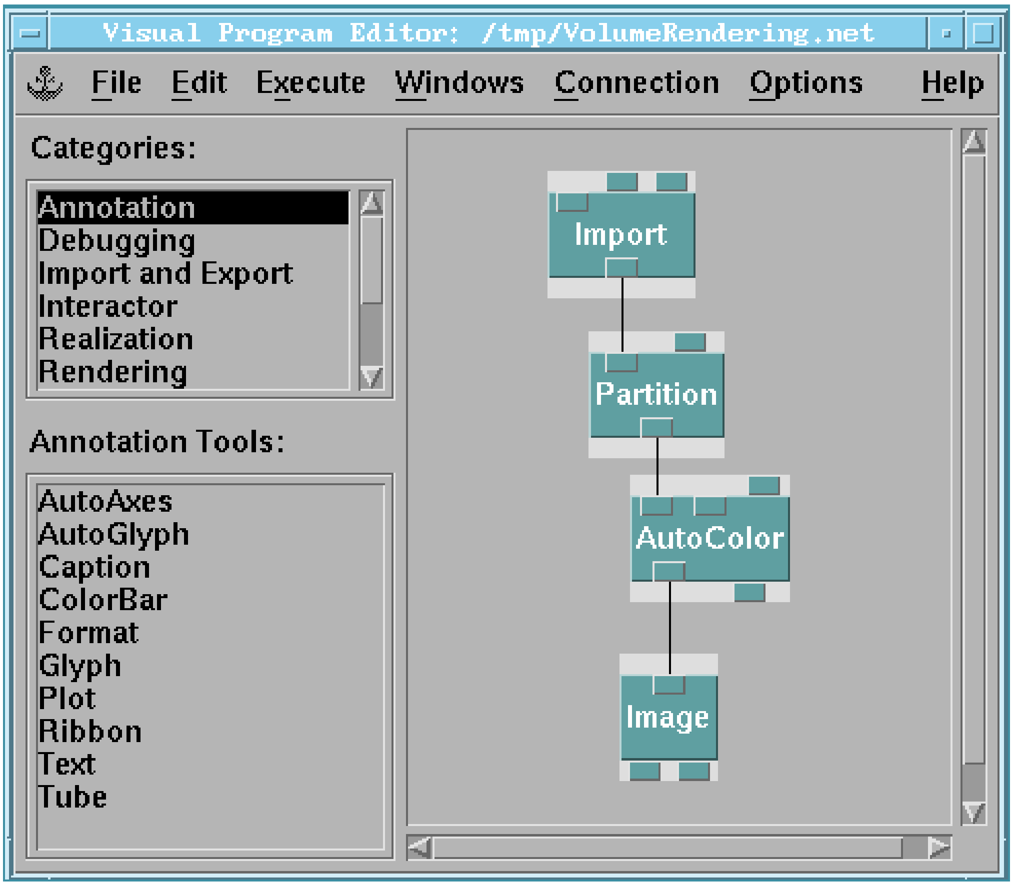

Figure 2: IBM Data Explorer Network

2.3.3 Running Example Programs

Appendix A.1 reads a previously defined network from disk, and explains how to execute it.

2.3.4 Data Input

There is no tutorial on data input and so the new user must refer to the appropriate manual pages. Chapter 4 of the User Guide deals with data import - it is well provided with examples in increasing order of complexity.

2.3.5 Finishing off

To leave dx, click on “File” on the menu bar of the VPE (the Visual Programming Environment) window and select “Quit”.

2.4 IRIS Explorer

The manual IRIS Explorer User's Guide [28] has a very good chapter concerned with "Getting Started" for the first time user and an overview of the IRIS Explorer system can be found in pages xx et seq (before Chapter 1 in the User's Guide). The User's Guide also contains chapters on how to use the Map Editor to build IRIS Explorer Maps. The following sections provide suitable references into this manual where more detailed information can be found.

2.4.1 Starting up the System

To start IRIS Explorer you simply type:

```
explorer
```

If IRIS Explorer does not start up you may need some additional environment variables defined (see Appendix A of the User's Guide).

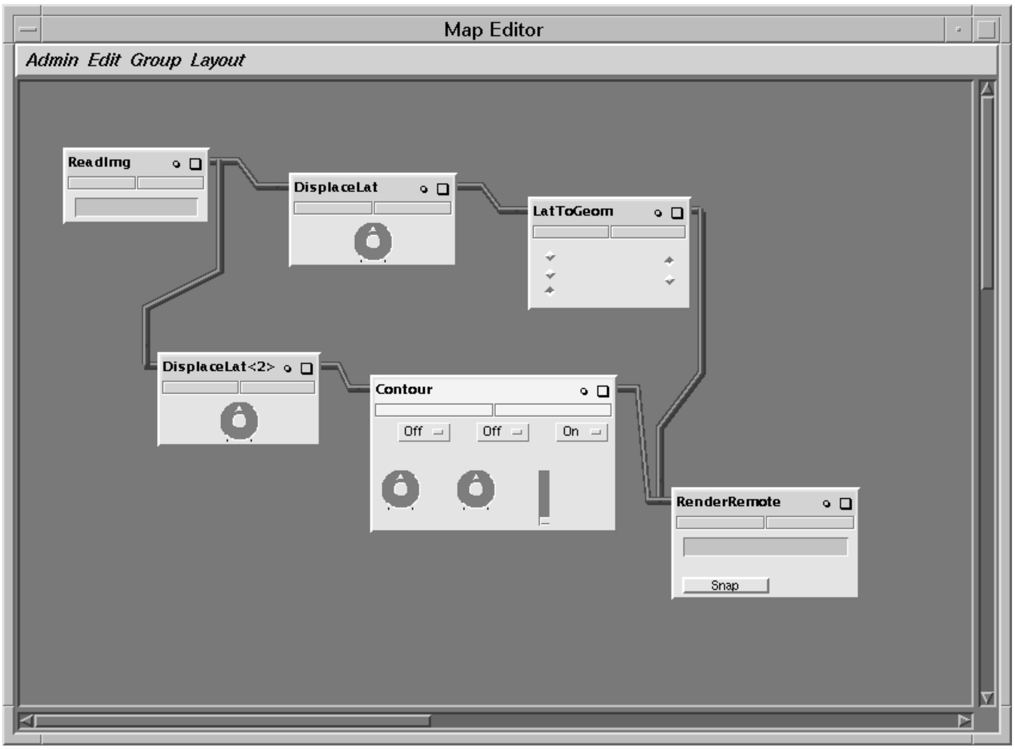

Figure 3: IRIS Explorer Map Editor

2.4.2 Running the Examples

This information can be found at the start of Chapter 1 in the IRIS Explorer User's Guide. In a standard installation the examples can all be found under `/usr/explorer/maps` and may be invoked from the Map Editor or directly when starting IRIS Explorer:

```
explorer -map mapname.map
```

2.4.3 Building your own Map

This is fully described in Chapter 2 and 3 of the IRIS Explorer User's Guide "Working with the Map Editor". An example of the Map Editor is shown in figure 3.

2.4.4 Data Input

Data is imported into IRIS Explorer using the DataScribe tool which is fully described in Chapter 7 of the IRIS Explorer User's Guide. As well as step-by-step instructions, there are samples in:

```
/usr/explorer/scribe
```

and data in:

```
/usr/explorer/data/scribe
```

2.4.5 Finishing off

To quit from IRIS Explorer you must select "Quit" from the Admin menu in the IRIS Explorer Map Editor window.

2.5 Khoros

The manual Khoros Getting Started Manual [33] should be consulted by the first time user. The following sections provide suitable references into this manual where more detailed information can be found.

2.5.1 Starting up the system

Before using any of the utilities in Khoros you must first configure your startup environment. This can be achieved manually or by running the utility `kconfigure`. Both of these techniques is described in Chapter 2 of the Getting Started manual. Once this has been done successfully you can try one of the Khoros applications e.g.,

```
kman kman
putimage -i image:ball
```

2.5.2 Building a Network

All information processing and visualization programs in Khoros are available via the visual programming language, cantata. Cantata is a graphically expressed, data flow visual language which provides a visual programming environment within the Khoros system.

There is a complete manual [39] dedicated to the Khoros application cantata and an example is shown in figure 4.

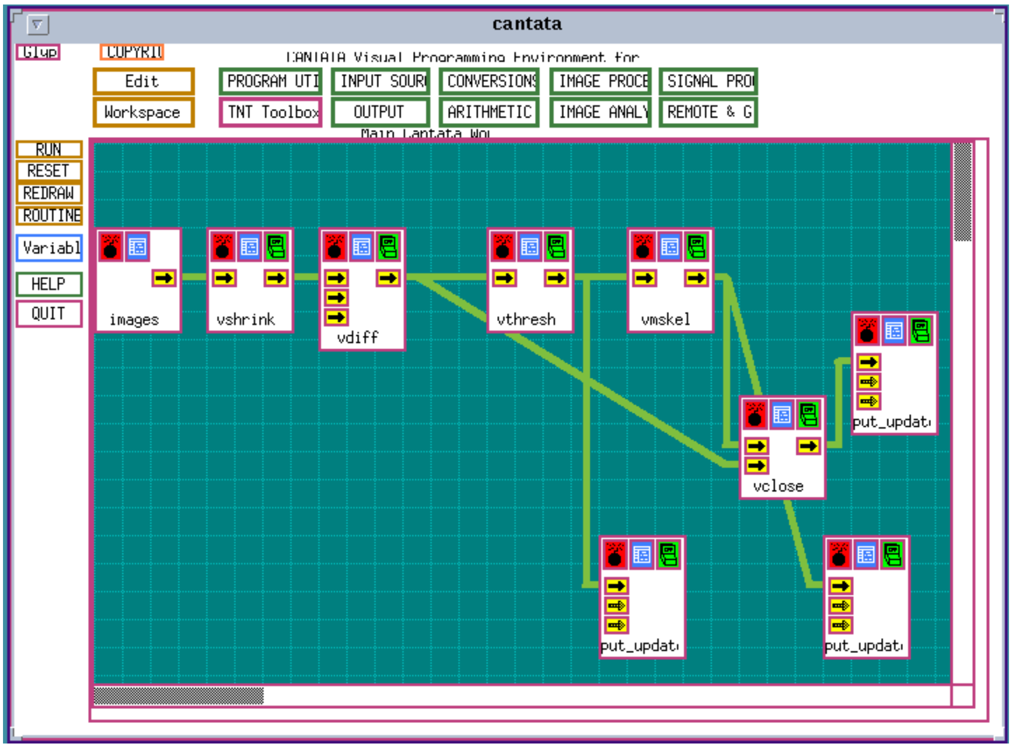

Figure 4: A Khoros Application in Cantata

2.5.3 Running Example Programs

There are a number of different sample Cantata workspaces available in the Sampledata Toolbox. These may be invoked from the command line by running

```
Cantata -restore <workspace alias>
```

with any one of the following aliases:

```
workspaces:ColorArith1
workspaces:ComplexArith
workspaces:Express
workspaces:FilterEdge
workspaces:FilterMedian
workspaces:Formats
workspaces:Fourier
workspaces:Geometry
workspaces:GeometryElevation
workspaces:Render3dFFT
workspaces:Signal
workspaces:SimpleArith
workspaces:Wavelets
```

```
workspaces:5DHead
```

2.5.4 Data Input

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. More information can be found in [36].

2.6 PV-WAVE

PV-WAVE CL is a particularly easy package to get started with, primarily due to having an excellent on-line help and demo system and good introductory documentation. The PV-WAVE Command Language Applications Guide and Overview have been combined into the PV-WAVE tutorial.

Once PV-WAVE CL has been started the supplied demo gallery is particularly comprehensive and useful, and the CL code for these examples is availble to the user. An example from a PV-WAVE demonstration is shown in figure 5.

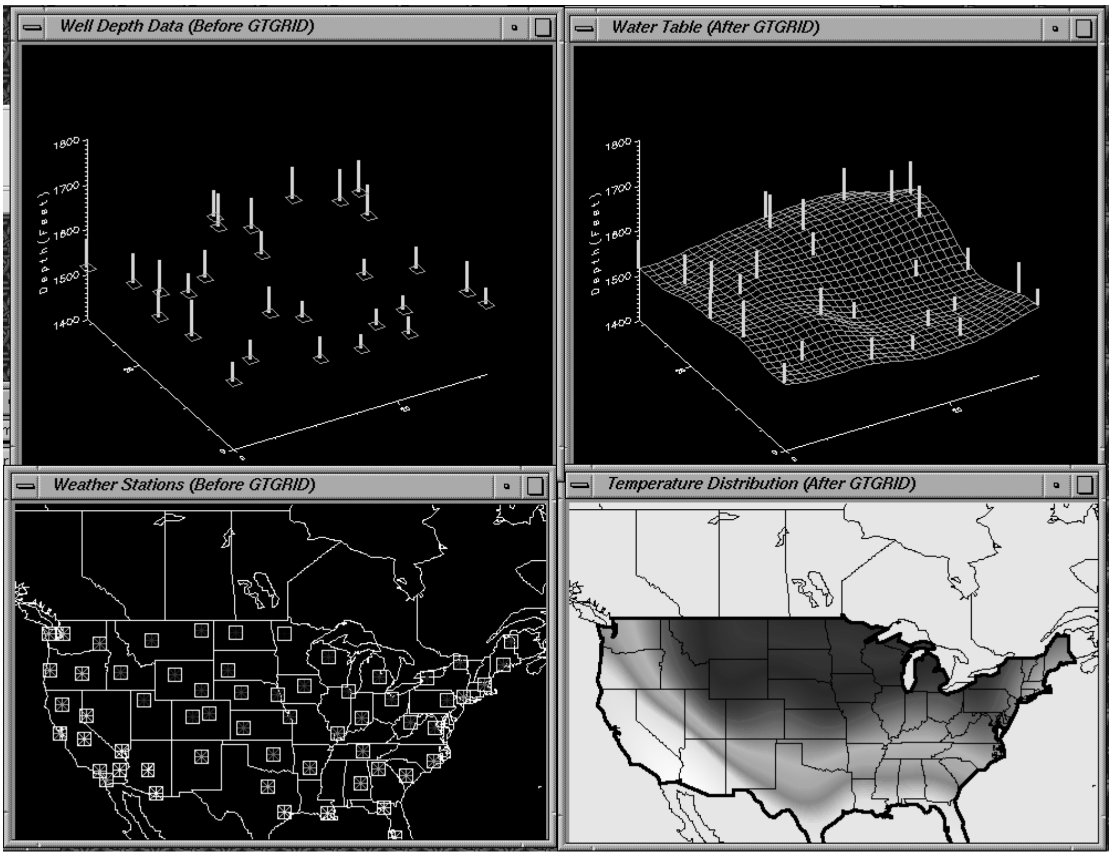

Figure 5: An example from PV-WAVE

There are three getting started manuals provided, although some are not obviously named. These are:

- Getting Started with PV-WAVE Point and Click Motif Version [54]
- PV-WAVE CL Applications Guide - Free Code and Sample Applications [52]
- PV-WAVE Command Language Overview [53]

The latter contains a very useful section, “An Interactive Session with PV-WAVE CL”, which introduces the basic concepts of the command language via a simple tutorial.

Detailed platform-specific installation information is given on the CD-ROM inlay and the electronic documentation. Script files are generated during the installation to setup the user’s environment, so the site dependencies in the manuals are reduced.

Chapter 3: Functionality

3.1 Introduction

In this chapter we summarise the functionality of the visualization systems.

We consider this under the following major headings:

- Data Models: what are the fundamentals of the data that can be handled by each system?
- Algorithms: what visualization operations can be applied to the data, creating a graphical abstraction?
- Presentation: how can the graphical abstraction be presented and manipulated?

3.2 Data Models

It is common for publicity about a visualization system to highlight the 3D or 4D plotting capabilities that it provides. This can be misleading to the potential user for a number of reasons:

- The potential user is primarily interested in the characteristics of the data; the plotting capabilities are a means to the end of analysing the data.
- Emphasising 3D or 4D plotting capabilities may mislead the potential user into believing that the kind of data handled is also 3D or 4D when that is often not the case.

Here we use an approach which starts with the data to be visualized.

3.2.1 General introduction

After importing, the visualization system uses its native form to manipulate a dataset - the external form is not used except to access and import the original data again. The capabilities of a visualization system are limited by the data model offered by this native form. A very powerful data model allows a wide range of visualization possibilities. However it is also possible for this potential not to be realised either in the supported product or by 3rd party software - we review the actual coverage in the sections on Algorithms and Presentation.

Fuller descriptions of the principles of data models are given elsewhere ([11],[63],[18],[17]). Here we confine ourselves to a few practical issues and describe how these are addressed in the visualization systems. We cover type, dimensions and organization of the data.

The scientific investigator who needs visualization is concerned with the values of certain properties defined on a grid.

In simple cases there is a single property, but in many cases the investigator is interested in several properties on the same grid. So the fluid dynamics expert could be concerned with pressure, temperature and flow. This is the dependent data which may represent measurements or the results of computations. In some problems there could be many dependent variables.

Visualization systems normally deal with discrete data. Although it is possible to deal directly with mathematical expressions, this is a subject not well dealt with in general purpose visualization systems. Even if the investigator's mathematical model involves a continuum, for computation or measurement reasons the data has usually been sampled. Hence we refer to a grid, which consists of a set of nodes - the independent data - used to define where the dependent data is sampled.

The coordinate system of the grid often consists of cartesian spatial dimensions and also time. A collection of flow data is a good example of this, using either 2 or 3 spatial dimensions.

However the coordinate system need be neither cartesian nor spatial. Chemical reaction rates may depend on the concentration of the constituent substances, the presence or absence of a catalyst and other quantities such as temperature. For the investigator, these are the independent variables and the values of those quantities, at which chemical reaction rates have been measured or calculated, represent a grid. For some problems, there could be many independent variables.

3.2.2 AVS

This section outlines the basic range of data types which are provided for importing the following classes of data. The information provided is correct for the current version of AVS (AVS5) but an additional section has been added to provide details on the changes that will be provided by the next release of AVS (AVS6) in early 1995. AVS5 provides a number of datatypes:

- Scalars;
- Fields;
- Unstructured Cell Data (UCD);
- Geometry;
- Colourmaps;
- Molecule Data Type (MDT);
- User defined data;

Simple data types (character strings, integers, real, boolean)

The simple scalar types (character strings, integers, real, boolean) are all supported as AVS datatypes.

General 1D/2D/3D arrays of data

The field datatype allows the importing of general arrays of data elements (N dimensional array of M elements). The datatype allows the dimensions of the arrays to be described with an optional mapping/transformation being applied to the data elements of the array as they were being imported. These mapping types were split into three distinct types:

- *uniform*: the mapping is direct and implicit as no transformation of the data elements is applied;

- *rectilinear*: a mapping is applied to the dimensions of the arrays so an additional vector of coordinates is required for each dimension of the array during the mapping stage. This mapping could for example cater for logarithmic axis;
- *irregular*: an explicit mapping is applied to each data element as it is imported into the system. The mapping can also raise the dimensions of the data e.g., mapping a 2D array of elements into 3D space. During the mapping stage an explicit coordinate value is needed for every data element.

Node and Cell based data

The Unstructured Cell Data (UCD) type is aimed at providing support for associating data with discrete geometric structures. The data type consists of nodes and cells to form the overall UCD structure:

- *nodes*: a number of nodal positions in 2D or 3D space;
- *cells*: a number of logical connections are defined between nodes to form cells which hence form the overall model. AVS supports the cell types: point, line, triangle, quadrilateral, pyramid, tetrahedral, hexahedral.

Data can then be optionally associated at node positions, part way along a connection between two nodes (mid-edge) or with a complete cell or UCD structure.

Geometric data

The AVS Geometry data type is provided to support the display of 2D/3D graphical objects. The primitive datatypes provided are: disjoint lines, triangle and quadrilateral meshes, polyhedron definitions and spheres. There is also support for the definition of light sources, texture mapping and control over the camera parameters to perform clipping, depth cueing and perspective viewing of scenes.

Chemistry application data

The Molecule Data Type (MDT) is provided to support chemistry applications and consists of a number of objects arranged in a hierarchical structure:

- CHEMmolecule: root object, contains name and data;
- CHEMatoms: data for component atom;
- CHEMchemunits: molecular substructures;
- CHEMquantums: quantum chemistry;
- some support for the addition of user-defined data fields.

Colourmaps

The AVS colourmap data type is an arbitrary sized one-dimensional array of 4D vectors. The vector components are normalised floating point values which represent hue, saturation, brightness and opacity.

User defined data

The AVS system provides a variant of defining structures in the C programming language which encapsulates a number of the scalar data types into a user defined structure. The structure can also contain arrays of these scalar types.

Errors and undefined values

Currently there is no real support for errors or undefined values but this is provided for in AVS6.

AVS6

This section details the changes to the data types which will appear with the release of AVS6 in Q2 1995. In AVS5 different data types were required for different classes of data (i.e., 2/3D data, finite elements, geometric). AVS6 now has one unified data type which can be used to represent these classes of data within the same data structure. This removes the need for different visualization modules to perform the same function on different types of data e.g., there is now only one isosurface module for both field and unstructured cell data.

Another improvement is the facility to extend this base data structure with user defined fields. For example you could add patient attribute information to image scans and the modules within AVS would still recognise and process this extended data type as an image.

Other features include support for cylindrical, polar and spherical coordinate systems and the provision for users to specify NULL (undefined) data values. These underlying data types will also be supported by the visualization modules in AVS6.

With AVS6, there will be a number of new features to improve the handling of large data sets. These features include:

- data reference instead of data flow;
- direct rendering of large data sets;
- data chunking for processing large data sets a section at a time.

3.2.3 IBM Data Explorer

Data Explorer presents a unified data model based on the concept of a field. This concept brings together the grid positions used to define where the data is sampled, the connectivity and the property data. From the point of view of the user, a module may be defined on any kind of data where it is reasonable to do so - there is no artificial distinction between types of data.

Simple data types (character strings, integers, real, boolean)

DX does not really treat these as data types. Some modules do process all these data types - as standard C items - but are generally parameters to determine how to process data. A DX object can be a string, but better support is provided for strings and numbers that are associated *attributes* of a more complicated data structure (e.g. to associate the data with a name or number).

General 1D/2D/3D arrays of data

The easiest way to import array data into DX is to use the prompter tool. Figure 6 shows the interface to import an ascii file containing data values on a uniform 3D grid. The data file consists of data values only, and has a float value per line.

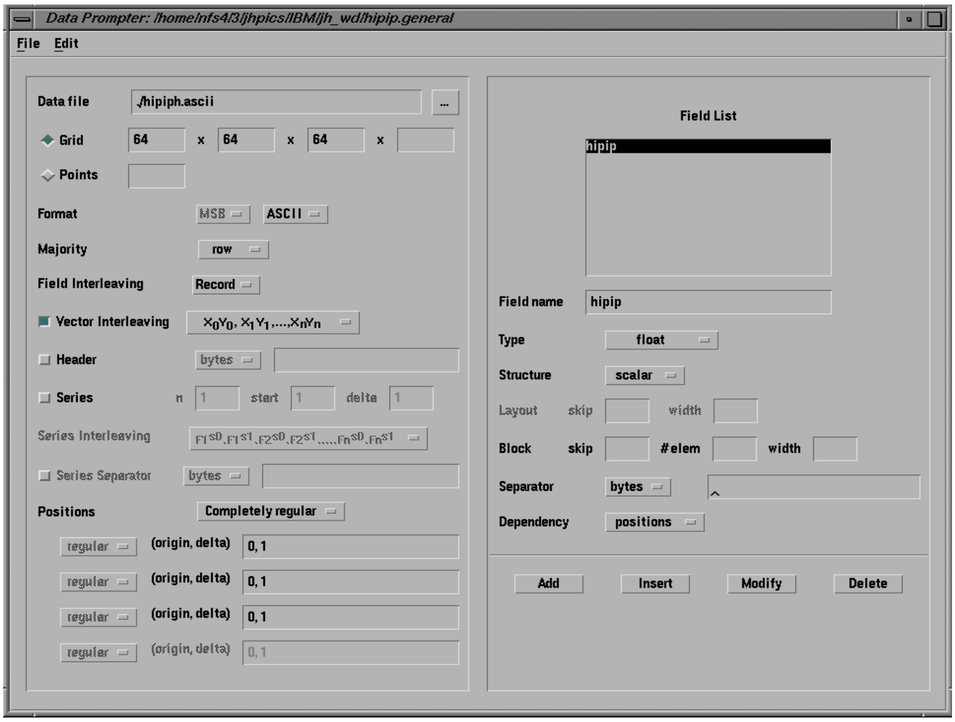

Figure 6: The DX prompter interface for data on a 3D uniform grid.

This generated the following general array format header:

```
file =./hipiph.ascii
grid = 64 x 64 x 64
format = ascii
interleaving = series-vector
majority = row
field = hipip
structure = scalar
type = float
dependency = positions
positions = 0, 1, 0, 1, 0, 1
end
```

The import module converts the data into DX objects, in this case, a Field with 5 components:

1. *data,* a generic array containing the data values.
2. *positions*, as it is a uniform grid this contains the start value for x y and z and the delta (spacing) between each value in x y and z.

3. *connections* describes the connections and element type for the data.
4. *box* defines the corners of a bounding box.
5. *data statistic* contains statistics on the data.

which converts to the following DX format when imported with the import module and exported with the export module (the actual data is now binary and not shown here):

```
object 1 class array type float rank 0 items 262144 msb ieee data 0
attribute "dep" string "positions"
#
object 2 class gridpositions counts 64 64 64
origin 0 0 0
delta 1 0 0
delta 0 1 0
delta 0 0 1
attribute "dep" string "positions"
#
object 3 class gridconnections counts 64 64 64
attribute "element type" string "cubes"
attribute "ref" string "positions"
#
object "hipip" class field
component "data" value 1
component "positions" value 2
component "connections" value 3
attribute "name" string "hipip"
#
end
```

Node and Cell based data

Both node and cell based data are supported. To define node data, state the data is dependent ("dep") on the positions and for cell based say it is dependent on connections. Current cell types include: line, triangle, tetrahedron and cube (the latter is a general purpose shape defined by 8 vertices). Currently no mid edge data is supported.

It is possible to define a DX header to import node and or cell-based data into DX. For example the following lines have been extracted from the standard example dataset in `/usr/lpp/dx/samples/data/irregular.dx` data file:

```
# The irregular positions, which are 24 three-dimensional points.
object 1 class array type float rank 1 shape 3 items 24 data follows
0 0 0
0 0 1
0 0 2
0 2 0
0 2 1
0 2 2
1 0.841471 0
```

```
.............
3 2.14112 2
# The irregular connections, which are 30 tetrahedra
object 2 class array type int rank 1 shape 4 items 30 data follows
10 3 4 1
3 10 9 6
10 1 7 6
.........
17 20 23 22
attribute “element type” string “tetrahedra”
attribute “ref” string “positions”
# The data, which is a one-to-one correspondence with the positions
object 3 class array type float rank 0 items 24 data follows
1 3.4 5 2 3.4
5.1 0.3 4.5 1 2.3
4.1 2.1 6 8 9.1
2.3 4.5 5 3 4.3
1.2 1.2 3 3.2
attribute “dep” string “positions”
# the field, with three components: “positions”, “connections”, and
# “data”
object “irregular positions irregular connections” class field
component “positions” value 1
component “connections” value 2
component “data” value 3
end
```

In this example the data is included with the description but does not have to be.

User defined data

DX has a rich collection of data objects, it is also easy for the user to group a number of objects into a more complex structure (although the description header may be difficult to read).

Errors and undefined values

It is possible to specify the data is invalid, by including a component *invalid positions* which lists the locations, or *invalid components* to list invalid *cells* (where cells can be any cell, e.g. an FE cell or a cell from a uniform grid). This means holes can appear in both regular and irregular grids, or allows the user to define regions where another grid applies in a particular region (e.g. to replace parts of a grid with areas of greater detail). The standard example network `/usr/lpp/dx/samples/programs/InvalidData.net` shows that invalid data is not rendered - e.g. holes appear in the shaded object, contour lines do not pass through invalid points and streamlines stop at invalid points.

When using DX it is also possible to set a *valid* range of values for use by the autocolour/autogreyscale and specify colours to be used for out of range data (this could be used to highlight unsuspected invalid data). Also the Include module can aid the viewing of invalid data and data ranges. This module allows the user to select only those points which lie within or outside a given range. Non selected points can be marked as invalid (Cull).

Any comments

DX also has good support for *series* of data, such as time series. It can also handle the B-rep model for CSG as it is possible to have faces, loops and edges components in an object (e.g. the CSG examples in the public domain).

Although facilities for importing data by using a header description are very powerful in Data Explorer, it is also possible to generate filters (e.g. the FLUENT filter in the public domain) or import modules (e.g. the PHOENICS import module from the public domain). To develop code to import data both as filters and modules see Chapter 6, "Modules that Import Data", and Chapter 10, "The Data Model", of the Programmers Reference manual.

3.2.4 IRIS Explorer

IRIS Explorer - like AVS - also presents a distinction between major data types. As well as geometry, IRIS Explorer provides lattices, pyramids and parameters.

This section outlines the basic range of data types which are provided for importing the following classes of data. The Explorer data types covered are:-

- Parameter
- Lattice
- Pyramid - Finite Element and Molecule Pyramids.
- Geometry
- Pick
- User defined types

Simple data types (character strings, integers, real, boolean)

The Parameter data type handles all of these explicitly, with the exception of boolean (although those values can of course be handled as some other data type).

General multidimensional arrays of data

This form of data is handled by Explorer's Lattice data type. The Lattice is sub-divided into 3 parts:-

1. nDim and dims where nDim is the number of dimensions and dims is an array holding the extent of each dimension.
2. cxData which is a structure containing (1), nDataVar which is the number of data variables, a primitive type definition and an array holding the data. (optional)
3. cxCoord which is a structure containing (1), the coordinate type and an appropriate type of

array holding the coordinates. (optional)

The values of nDim and dims must be the same for each section. There are 5 primitive types:- byte, short, long, float and double.

Lattices have 3 coordinate types:-

- *Uniform*: these can be multidimensional with nDataVar data variables of any primitive type at each node, but with no explicit coordinates. The shape is defined by the number of nodes in each direction with a uniform cell size. Explorer uses a bounding box to set the size and aspect ratio of Lattice coordinates.
- *Perimeter*: these can be multidimensional with nDataVar data variables of any primitive type at each node, and a list of coordinates sufficient to specify an irregularly spaced rectangular structure. It has a 1, 2 or 3 coordinate dimensions depending whether it is a 1D, 2D or 3D Lattice.
- *Curvilinear*: these can be multidimensional with nDataVar data variables of any primitive type at each node, and with M coordinate variables to describe the position of each data point. The dimensionality of the Lattice gives an implied connectivity to the data.

Colour maps are just specific instances of a 1D uniform lattice with either 1 data variable (grey scale), 3 data variables (RGB) or 4 data variables (RGB & opacity).

Node and Cell based data

The Explorer Pyramid data type holds two types of data: irregular or unstructured data and molecular modeling data. A pyramid consists of 3 main parts:

- the several layers of pyramidal data; for example, points, lines and faces. These values are collected in cxLattice structures.
- the relationship between these lattices, described by cxConnection. The cxLattice and cxConnection together form a layer.
- optional references to predefined pyramid elements which are stored in a dictionary, cxPyramidReference.

Irregular or Unstructured Data: In IRIS Explorer, the Pyramid data type is primarily used for finite element modelling and creating irregular grids and most Explorer pyramid modules handle only this kind of data. The basic layers to create a Tetrahedral grid include:

- the base lattice which holds the coordinate and data at each node.
- layer 0 (1D) which lists the connections between nodes to form edges.
- layer 1 (2D) which lists the connections between edges from layer 0 to form faces.
- layer 2 (3D) which lists the connections between the faces from layer 1 to form tetrahedral elements.
- layer 3 which lists the connections between the tetrahedrons of layer 2 to form complex structures composed of tetrahedral elements.

Molecular Modeling

Chemistry pyramids are used to construct objects according to information pertaining to molecule structures. This pyramid structure is more narrowly defined than that for finite elements and represents (in the 3D layer, layer 2) not a volume, but a ball-and-stick construction. There are a number of modules associated with this type such as BallStick which takes a molecule pyramid and outputs it, in Geometry, as a ball and stick construction, and ReadPDB which links in to the Brookhaven Protein DataBase and can search for named molecules and builds Chemistry Pyramids.

A number of chemistry modules have been written for IRIS Explorer at Imperial College; these are bundled with version 2.2 in the unsupported modules category. These include modules for reading data from a variety of chemistry packages (see section 4.4.1), for the creation of geometry representing molecules as ball and stick and ribbon constructions, and for the animation of molecular vibration modes.

Geometry data type

As noted above in section 3.4.7, geometry in IRIS Explorer is implemented using Inventor, an object-oriented 3D toolkit. Geometry objects which can be created via the IRIS Explorer geometry API include points, lines, polygons, spheres, cones, cylinders, triangle meshes, NURBS patches, octree volumes, and text; this interface also offers control over colours, lighting and geometric transformations. It is possible to also create and manipulate the geometry via the Inventor API.

Geometry is read into and written out of IRIS Explorer as an Inventor object, which makes it available to other Inventor applications such as SGI's Showcase presentation package. The Open Inventor file format is being promulgated by SGI as a de-facto standard for 3D scenes, and the number of Inventor applications, readers and translators is apparently on the increase.

Pick Data

This is a specialised data type that is used only in 2D and 3D modules. It is used to enable the user to pick, or select, a particular location in an image (2D) or rendering (3D) module display window and obtain information about it. For example, you may want to query part of a picture of an isosurface. You could find out the coordinates of a spot on the surface or the data value at that point. It is also possible to use the pick data type to create an object or move an existing object to a new location in a rendering module display window. As the user clicks on the position in the window the information goes to an upstream module which creates the new geometry and sends it to the Render module.

User defined data

IRIS Explorer provides a language, called the Explorer Typing Language (ETL) which can be used to build new user-defined data types that can be passed between modules in the same way as other IRIS Explorer data types. ETL was used to create the standard Explorer root types such as Lattice and Pyramid etc., which contain subtypes such as cxData and cxCoord. A user-defined root type can reference these Explorer subtypes in its definition. ETL is syntacticly very similar to C.

Once the user has defined the data type using the ETL, IRIS Explorer compiles it. This consists of the automatic translation of the ETL file into C and creates a library of accessor functions for the different components of the type, together with the appropriate header files. Information about the new type is automatically picked up by tools such as the Module Builder (used in the creation of new modules - see below) and the Map Editor (used in the creation of new applications). The user can then write their own modules to process data using the new type, manipulated via the new API which has been automatically generated by IRIS Explorer in the compilation of the type.

Errors and undefined values

Having read in an Explorer Lattice there are several ways to deal with erroneous values. Using ScaleLatNode it is possible to define a min and max for the data and then either Clamp, stretch or threshold the erroneous values. Clamping fixes the value to either the min or the max value, stretching rescales the data to fit the parameter min and max and thresholding produces a binary array where 1.0 is inserted for a correct value and 0.0 for an error. This can be used later to take different courses of actions for correct and error values.

3.2.5 Khoros

Khoros actually provides two data models, a polymorphic data model and a geometry data model. These data models are implemented within the Khoros data services libraries. All Khoros data processing and visualization routines are written to operate on these data models via data services. The low-level functionality of data services give these operators the ability to operate on data independent of data type, size, and file format.

Polymorphic Data Model

The polymorphic model is so named because it is capable of storing data from several different domains. By capitalizing on the commonality of data interpretation across these different domains, the polymorphic model facilitates interoperability of data manipulation routines. In other words, processing routines which use the polymorphic data model will be able to process data objects containing anything from signals to images and from to volumes to animations.

The polymorphic model consists of data which exists in three-dimensional space and one-dimensional time. You can picture the model most easily as a time-series of volumes in space. This time-series of volumes is represented by five different data segments. Each segment of data has a specific meaning dictating how it should be interpreted. Specifically, these five segments are value, location, time, mask, and map. All of these segments are optional; a data object may contain any combination of them and still conform to the polymorphic model.

The value segment is the primary data segment, consisting of data element vectors organized implicitly into a time-series of volumes. The value data may be given explicit positioning in space and time with the location and time segments. The remaining two segments are provided for convenience. The mask segment is used to mark the validity of each point of value data. The map segment is provided as an extension to the value data; the value data can be used to index into the map data. Figure 7 provides an overview of the Khoros Polymorphic Data Model.

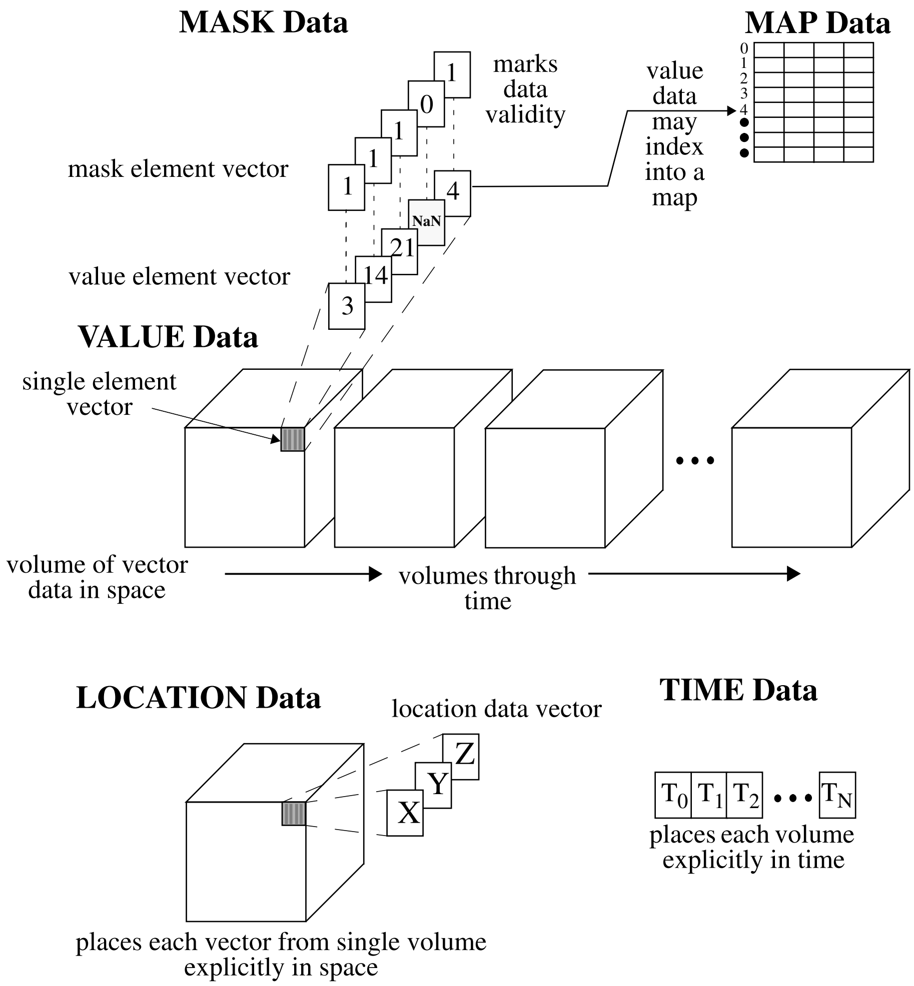

Figure 7: An overview of the Khoros Polymorphic Data Model

Value Segment

The value data segment is the primary storage segment in the polymorphic data model. Most of the data manipulation routines are specifically geared toward processing the data stored in this segment. In an imaging context, the individual pixel RGB values would be stored in here. In a signal context, regularly sampled signal amplitudes would be stored here.

The value segment consists of a time-series of volumes where each volume is composed of element vectors. Each element vector is composed of a number of value points. The size of the value segment is determined by the width, height, and depth of the volume, by the number of volumes through time, and by the number of points in the element vector. This makes the value segment, and the polymorphic data model, inherently five-dimensional.

Location Segment

The value points in the value segment are stored implicitly in a regularly gridded fashion. Explicit location information can be added using the location segment. If the value data is irregularly sampled in space, the explicit location of each sample can be stored here. Specifically, the information stored in this segment serves to position each the value data in explicit space. Note that the location data only explicitly positions a single volume; the position then holds for each volume through time.

The location segment consists of a volume of location vectors. The width, height, and depth of the volume are identical to the volume size of the value segment. Different location grid types are also supported. A curvilinear grid allows for explicit locations to be specified for each vector in the value data. A rectilinear grid allows for explicit locations to be given for the width, height, and depth axes. A uniform grid allows for explicit location corner markers to be specified.

Time Data

Explicit time information can be added using the time segment. If each volume of value data is irregularly sampled in time, an explicit timestamp for each volume can be stored here. This is useful in animations where each frame of the animation occurs at a different time.

The time segment consists of a linear array of timestamps. The number of timestamps matches the time size of the value segment.

Mask Data

The mask segment is available for flagging invalid values in the value segment. If a processing routine produces values, such as NaN or Infinity, these values can be flagged in the mask data so that later routines can avoid processing them. A mask point of zero is used to mark invalid value points, while a mask point of one is used to mark valid value points. The mask segment identically mirrors the value segment in size; there is one mask point for each value point.

Map Data

In cases where the value data contains redundant vectors that are duplicated in different positions, the map segment may be used. The value vectors are replaced with values which index into the map; the map then contains the actual data vectors. In this sense, the map is an extension of the value segment.

The map segment consists of a number of width-height planes. The values from the value segment map into the map height indices. The map vector runs along the map width. A simple map would consist of just a single width-height plane; a more complicated map would have a width-height plane for every depth, time, and element plane in the value segment. This provides a great deal of mapping flexibility. For example, every plane in a volume or every image in an animation could have a separate map.

Geometry Data Model

The geometry data model supports the storage and retrieval of a number of standard geometric primitives, such as spheres, triangles, and lines. Other non-geometric primitives such as octmeshes and textures are also supported.

The geometry data model is centered around a primitive list. This list is able to store any combination of geometric primitives such as spheres or polylines. Each geometric primitive consists one or more different types of data, suchas location data and color data. The types of data required depend on the primitive; all primitives have location and most have color while only some have radii or normals. Most data is explicitly provided, although colors may be provided indirectly via a colourmap. Quadmesh and octmesh primitives, which are not illustrated here, are also available. These mesh primitives are overlaid on top of the polymorphic data model. Thus, from the point of view of the polymorphic data model, a quadmesh will appear to be an image, and an octmesh will appear to be a volume.

General 1D/2D/3D Arrays of Data

The polymorphic data model can be used to represent up to five dimensional arrays of data complete with auxiliary information such as explicit location and time data, validity data, and map data.

Node and Cell Based Data

Explicit nodes are provided for vertices or cells in the Geometry data model, however the connectivity is implied by the geometry primitive which the data represents. There is no support for arbitrary connectivity.

The polymorphic data model allows an explicit location to be assigned to every data vector in the Value segment, thus sparsely distributed node data can be represented. Connectivity between the nodes can be specified only through the implicit organization of the Value data. There is no support for explicit connectivity between arbitrary nodes.

User Defined Data

The data services provides library calls for manipulating a generic abstract data object containing arbitrary segments of any dimensionality. While new data models could be constructed using this, it is not recommended as no existing processing operators would be able to operate on the data. In general, if the data can be expressed in a five-dimensional space, it is best to use the polymorphic data model.

Errors and undefined values

The Mask data can be used to specify the validity of the data (undefined data). There is no explicit mention of support for errors within data values but the application could simply allocate a portion of the data vector to support this feature. Even a combination of the mask and value data could be used for indicating if data has an error component associated with it.

3.2.6 PV-WAVE

In principle, PV-WAVE's data model could be as general as its command language. This includes arrays and structures, which in combination allows a very flexible data definition. In practice the data is confined to what can reasonably be handled by the plotting procedures. To give an example, although it is possible to defined unstructured data using PV-WAVE's data types, no procedures to plot such data are available. More detailed information can be found in Chapter 4: Data Import section 4.6.1. There are also some procedures to create, manipulate and display polygon and vertex data within PV-WAVE and these are described in the Reference Volumes 1 and 2.

3.3 Algorithms

3.3.1 Introduction

This section deals with the algorithms which can be used to visualize a particular data set. It is structured according to the following scheme:

- first, we characterise the nature of the data - this effectively gives us classes of problems, such as scalar fields in 3D etc., (see section 3.3.2);
- next, we review briefly interpolation techniques - this gives us a means of creating a model from the data that is defined everywhere (see section 3.3.3);
- then, for each of the classes, we describe some of the possible visualization techniques (see section 3.3.4 to section 3.3.7);
- finally, we look at the availability of the different techniques in each system (see section 3.3.8 to section 3.3.12).

3.3.2 Classification

In visualization, we begin with a set of data. This will come either from observation (as in satellite recording or medical scanner) or from simulation of some physical process (as in CFD). Either way, the data is a sample from some underlying field - which we do not know, but which we are looking to visualization to help us understand. Thus a fundamental step in visualization is the creation of an empirical model, guided by the scientific investigator's understanding of the underlying field.

When selecting a visualization algorithm, it is useful to first consider the nature of the underlying field. What are the dimensions of the independent variables? What are the datatypes of the dependent variables, and so on.

Indeed we shall use the nature of the underlying field as our means of classifying the visualization algorithms. We use a subset of the classification derived at the AGOCG Visualization Workshop in 1991 [11].

The letter E represents the entity to be visualized, and a subscript denotes the dimension of the independent variables. Thus E_3 represents an entity defined over a 3D region. Time is treated specially - so in the above example, if the entity varies over time, we write $E_{3;t}$.

The type of dependent variable can be scalar (S), vector of dimension k (V_k) or tensor (T_{kk}), and this is written as a superscript.

So a vector field over 3D is written as: $E_3^{V_3}$.

3.3.3 Interpolation

Interpolation is the process by which the empirical model is created from the data.

There are a variety of methods and a good reference is the book by Lancaster and Salkauskas [44]. Here we give only a brief summary.

Consider first just E_1^S. We shall be given a set of points $(x_i, f_i), i = 1, 2, \ldots, N$. The interpolation problem is to construct a model F(x) which matches the data at the given points.

The simplest method is *nearest neighbour*, in which the value of F is taken to be the f-value of the nearest datapoint to x. Notice two features: it is very quick; but F(x) has a discontinuity midway between each datapoint.

Continuity can be improved by *linear interpolation*, in which F(x) is a linear function (that is, a straight line) between data points. This requires some computation so is slower, but F(x) is now continuous. Note however the slope of F(x) is not continuous, so if we know the underlying entity is smooth, then our recreation of it may be misleading in this sense.

Slope continuity (continuity of the first derivative of F(x)) can be achieved, but at the expense of more computation. A usual method is to estimate the slopes at the data points (for example, as the average of the slopes of the lines to adjacent data points) and then to fit a cubic function in each interval between data points. There are a variety of techniques for doing this *piecewise cubic interpolation* [12].

Second derivative continuity can be achieved, at even more computational expense, using *cubic splines*.

There are analogs of these methods in 2D and 3D. Consider first data on a rectilinear grid. The nearest neighbour extends in an obvious manner - it is quick, but discontinuous as in 1D.

Linear interpolation extends to *bilinear (2D)* or *trilinear (3D)*. Bilinear interpolation proceeds thus: find the grid square of interest; use linear interpolation in x for both extreme values of y; use these two calculated values in a further linear interpolation step in the y-direction. Trilinear is an obvious extension to 3D. These give continuity of function value, at extra computation expense. Hill [21] describes computational aspects of linear, bilinear and trilinear interpolation.

It is important to note that the bilinear interpolant in 2D is deceptively complex to visualize: it is a curved surface with contour lines which are hyperbolic. Similarly, the trilinear interpolant in 3D is likewise complex: surfaces of constant value are hyperbolic in nature. For this reason it can be useful to split the rectangles into triangles, or cuboids into tetrahedra, and fit simpler interpolants in each triangle or tetrahedron. These have straight line contours, or planar surfaces of constant value, respectively.

Piecewise cubic interpolation extends to *piecewise bicubic* and *piecewise tricubic*; these provide first derivative continuity, but are relatively rare (especially tricubic) on account of the computation involved.

Suppose now the data does not lie on a rectilinear grid. A variety of techniques have been suggested for scattered data, and a good recent review is by Foley and Nielson [16]. A very simple and reliable method is *multiquadric* (MQ) interpolation. The MQ interpolant is continuous in all derivatives, and is defined (in 2D) as:

$$F(x, y) = \sum_{1}^{N} a_i B_i(x, y)$$

where

$$B_i(x, y) = \left(R^2 + d^2\right)^{0.5}$$

with R a constant and d the distance of (x,y) from the ith data point (x_i,y_i). The coefficients a_i are found by solving the N linear equations:

$$f_i = F(x_i, y_i)$$

The extension to 3D (and indeed higher dimensions) is straightforward.

Another good method is the *quadratic Shepard's method*. This has the form:

$$F(x, y) = \sum_{1}^{N} w_i(x, y) L_i(x, y)$$

where $L_i(x,y)$ is a quadratic function constructed to be a good approximation to the underlying function near the corresponding data point (x_i,y_i), and $w_i(x,y)$ is a weighting based on the distance of the interpolation point (x,y) from the corresponding data point. Again the extension to 3D is straightforward.

Another approach is to construct, in 2D a triangulation of the data, or in 3D a tetrahedral decomposition. There are well known algorithms for this: the Delaunay triangulation has optimal properties in terms of avoiding long, skinny triangles - traditionally thought to be a good thing. (Note however that views on this are changing - data dependent triangulations that align the triangles with features of the data, rather than across them, can give superior results in practice - see the paper by Dyn et al [14])

The triangulation is useful because local interpolants can be fitted within each triangle. Again these can be linear to give function value continuity, or higher order to give slope continuity.

There are many other techniques for interpolating scattered data, and the reader is referred to [16] for detail. Renka [60] gives an efficient implementation of the quadratic Shepard method.

3.3.4 Algorithms for scalar data over 3D

E_3^S

A large class of applications require the visualization of a scalar field over a 3D region. For example, data from medical scanners, temperature or pressure data from CFD.

There are essentially two approaches:

- *Surface Extraction:* In this approach, a 2D surface of interest is extracted. The surface can be extracted as a 2D planar *slice* through the 3D region, with 2D algorithms being used to display the data. Alternatively, the surface can be extracted as an *isosurface*, comprising all points with the same value of the scalar field. The slice therefore extracts in the space of the independent variable; the isosurface extracts in the space of the dependent variable.

  Of course, these techniques display only a 2D subset of the data. However, the third dimension can be visualized through animation - in the case of slicing, the cross-section can moved in time through the 3D region; in the case of isosurfacing, the isolevel can be moved in time from minimum to maximum.

  Note that in the case of an isosurface, it is possible to visualize a second scalar field over the extracted surface, by assigning colour values to the surface corresponding to the second scalar field.
- *Volume Rendering*: In this approach, the entire 3D volume is visualized. This is achieved by mapping the scalar value to colour and opacity, to form a coloured jelly-like *material* which can then be rendered. This allows the interior of the volume data to be inspected, and varying the classification of data by colour and opacity enables different features of the data to be extracted. For example, in an extreme case, an isosurface can be extracted by classifying all values greater than a threshold to be opaque.

We now look at these approaches in greater detail:

Surface Extraction

Slicing

This is relatively elementary: the slice can be positioned anywhere in the volume, a special case being the *orthogonal slice* which is perpendicular to one axis. A typical visualization technique on the slice is the image display, in which each pixel is coloured according to the scalar value at the corresponding position on the slice; but other 2D techniques are possible (e.g. contour lines). The 2D algorithms are described in section 3.3.6.

Isosurfacing

The extraction of isosurfaces has been studied in detail by researchers over the last few years. An excellent review article has recently appeared [49].

The best known algorithm is *marching cubes* [47]. This assumes data defined on a set of cubical cells, and constructs a polygonal approximation to the isosurface on a cell-by-cell basis. The extraction uses linear interpolation along edges of the cube cells, and a simple strategy to determine the surface across faces and in the interior. This strategy needs some care however, as topological ambiguities can occur, and as a consequence there can be holes in the resulting surface. Ning and Bloomenthal [49] explain how this can occur, and the remedies which can be taken. Certainly a good algorithm will contain some robust disambiguation strategy.

One such strategy is to decompose each cell into a number of tetrahedra. If this is done carefully - again see Ning and Bloomenthal - then linear interpolation along edges is sufficient to uniquely define a piecewise linear interpolant within each tetrahedron, and hence an unambiguous surface over the entire volume. This is known as *marching tetrahedra*.

Other strategies are possible, and some are more efficient than marching tetrahedra because they generate fewer triangles.

If the data are not given on a structured mesh, then there are techniques to construct a tetrahedral mesh from the data

Marching tetrahedra can then be used on this mesh.

A common feature of all algorithms is the generation of a set of triangles which approximate the isosurface. These are passed to a geometry renderer for display. For lighting and shading, it is important to calculate surface normals at the triangle vertices, and different strategies are possible:

- Calculate the gradient of the scalar field (which is in the direction of the surface normal) at the data points - for cubical cells, central differences can be used. Linear interpolation along edges will give estimates of the normal at the triangle vertices.
- Calculate the normal at a vertex as the average of the normals to the triangles which share that vertex.

Either strategy will generate normals which can be used for Gouraud or Phong shading to create a visually smooth isosurface.

As mentioned earlier, the colour of the isosurface may be assigned by some transfer function of a second scalar field.

Volume Rendering

In volume rendering, the intent is to display a representation of the entire 3D volume data - rather than extract a surface. The data is modelled as a coloured jelly substance, of varying opacity or transparency. It involves the following two initial steps:

- *Colour Classification*: The range of scalar data values are mapped to a range of colours, using some transfer function.
- *Opacity Classification*: The range of scalar values are mapped to a range of opacity values, using again some transfer function.

This classification step is under the control of the user, and is the key to successful visualization. As an example, consider a medical application where the scalar data values represent density of tissue. Then a certain range of data values will be known to correspond to bone, say, other ranges to skin, fat etc., and these can be mapped to specific colour and opacity values. Other data values may be *fuzzy*, for example, maybe skin, maybe fat, with certain probabilities; these are mapped to colour and opacity values intermediate between those for definite skin and fat. This ability to handle fuzzy classification is an important aspect of volume rendering. There are two major approaches to rendering the jelly substance: direct ray casting and splatting which are described in section 3.4.2.

3.3.5 Algorithms for vector field over 3D

This is the type: $E_3^{V_3}$. The field is defined by position only, that is, it is time-invariant or steady.

There are a variety of techniques for the display of vector field data. For a good review, see [50] and [22]. We summarise the main techniques here:

- *Arrows*: A glyph showing the flow speed and direction, usually as an arrow of variable length, is drawn at a given set of points. One may choose to draw the glyph at some or all data points, in which case the data provides sufficient information. Alternatively, one may wish to draw the arrows at user-specified positions, in which case interpolation will be needed.

  There are serious perception problems when this technique is used in 3D. It is more successful when a 2D surface is extracted, and the arrows are shown only for data points on the surface - indeed it is often effective to draw only lines without an arrowhead giving a spiked appearance (hence the term “hedgehogs”). Some success has been obtained in 3D by using 3D glyphs.
- *Particle Advection*: This is based on a traditional experimental technique for flow visualization. A light-emitting particle is released into the field and its progress is filmed over a specified time interval. In experimental flow visualization, this is commonly called a *pathline*.

  In practice, this visualization technique is implemented as follows. An initial particle position is specified: say (x_0, y_0, z_0), at time $t = 0$. Then its progression is governed by the system of Ordinary Differential Equations (ODEs):

  $$\frac{dx}{dt} = v_x \qquad \frac{dy}{dt} = v_y \qquad \frac{dz}{dt} = v_z$$

  with the initial values:

  $$x(0) = x_0 \qquad y(0) = y_0 \qquad z(0) = z_0$$

  Because the flow is steady, the velocities v_x, v_y and v_z are dependent only on position, not on time. The solution of these equations is done numerically, and the quality of the ODE solver will determine the accuracy of the resulting path. Euler’s method is simple and quick, but generally inaccurate. Better results will be gained by a Runge-Kutta method. There are a family of R-K methods, with differing orders of accuracy. Second order R-K is commonly used.

The solution will also need interpolation to calculate the velocity at the points required by the ODE solver, and the different algorithms described earlier can be used.

The presentation can be as the animated movement of the particle, or the trace of the particle path can be rendered. In this latter case, the path is identical to a stream line as explained below.

- *Stream lines*: These are lines which are everywhere tangential to the velocity field. The speed of flow is also indicated by the relative closeness of the streamlines. The method is better in 2D than 3D, some depth cueing being needed to assist perception.

  For steady flows, particle paths and stream lines are identical, and so the computation process is identical. The user will typically provide a set of starting positions from which the stream lines are generated.
- *Stream ribbons*: This is a variation in which a pair of adjacent stream lines are considered to be the edges of a ribbon, and rendered as such. This is useful to display twist in a flow field [23].
- *Stream surfaces*: This is another variation, where a number of adjacent streamlines are connected into a polygonal surface [24].
- *Time lines*: This is based on the experimental technique of releasing a line of hydrogen bubbles into the flow at a given time. The position of this line at a number of successive time intervals is then displayed - as an advancing front.

  In practice, this can be computed by selecting initial positions on a line in the flow. These are imagined as the start points of particle paths, and the positions of the particles at regular time intervals are recorded - interpolation through the positions at any time gives the corresponding time line.
- *Streak lines*: In experimental flow visualization, this is defined as a line composed of those particles which have passed through a specified location in a specified period of time. In computational visualization, the term is used somewhat loosely. Sometimes it is used to refer to the release of a *line of particles* whose position is tracked through the flow; in a steady flow, these particle tracers will follow stream lines. On other occasions, it is used to indicate particle paths through a time-varying field. When we look at different systems, we shall try to indicate the meaning of the term in the context of the system.
- *Topology Methods*: This approach identifies the critical points where velocity vector is zero - and classifies them as sources, sinks, etc. The connection of critical points divides the space into regions of common flow properties. This is described in [20]. This has been implemented in the FAST [9] visualization environment developed at NASA Ames Research Center, but to our knowledge is not available in other visualization systems.

It should be noted that it is often useful to calculate scalar quantities from the vector field, for example:

- *magnitude of velocity*

- *magnitude of vorticity*

These scalar quantities are of type E_3^S and can be visualized as such.

3.3.6 Scalar field over 2D

There are three approaches:

- *Line extraction*: In this approach, a slice is taken in a 1D subspace. The extraction can be in the space of the independent variables, giving us a cross-section through the 2D region, and a 1D technique can then be used - for example, a graph. Animation can provide a view of the entire 2D region, by sweeping the slice through the space. The slice can be orthogonal to an axis, or at an arbitrary angle.

  Alternatively, the line can be extracted in the space of the dependent variable, giving a line (an isoline or contour line) along which the scalar field has constant value. Several isolines can be drawn, within a range of values of the scalar function, so as to give a view of the whole region. A variation is to shade with constant colour between isolines.
- *Surface drawing*: In this approach, the scalar value is mapped to a third spatial dimension, giving a surface in 3D space which can then be rendered as a geometric object. Note this also allows a second scalar field to be displayed, by colouring the surface according to the corresponding value of the other field.
- *Image display*: In this approach, the 2D region is mapped to an area on the display, and each pixel coloured according to the value of the scalar function - using some suitable transfer function.

These approaches are described in more detail:

Line Extraction

- *Slicing*

  This is elementary: typically values along the slice direction are extracted, and a simple graph is drawn. The 1D interpolation techniques mentioned earlier can be used to *fill in* between the extracted samples.
- *Contouring*

  There is a vast literature on contouring which has been a popular 2D visualization technique for geographers and other scientists for many years. A good review paper is by Sabin [61].
- If the data is defined on a regular grid, then there is a simple method (the analogue of marching cubes in 3D) in which linear interpolation is used to estimate the intersection of isolines and grid lines - with some strategy for determining the lines within grid cells. Ambiguities can again occur, so care is needed (see Sutcliffe [62]).

  Methods based on linear interpolation will have slope discontinuities as the isolines move between grid cells. For smoother lines, one needs a method based on bicubic interpolation -

see Preusser [51].

For scattered data, one approach is to create a triangulation of the data points. Within any triangle, the intersection of the isoline with the edges can be found. Joining the intersections with straight lines gives the isoline of a linear interpolant within the triangle.

An alternative approach for scattered data is to use an interpolant to estimate the underlying field on a grid, and then use a gridded contouring method. It is often useful to enhance the appearance by shading the regions between isolines - indeed the isolines may be removed altogether. As with volume rendering, one needs a colour transfer function to associate colour values with values of the underlying field.

Surface Drawing

Another traditional graphics technique for 2D data is the *carpet plot* which shows 2D scalar data as a surface in 3D space - the value being mapped to the height axis. In earlier days this was drawn as a projection of a network of 1D curves parallel to the x and y axes, giving the carpet like effect. Modern implementations will draw this as a smoothly shaded surface.

It is possible to show a second scalar field by draping a colour shaded contour map over the surface - that is, one scalar variable is represented by height, the other by colour.

Image Display

This is a very simple technique in which the sample region is mapped to a corresponding region on the device, and the colour of each pixel is determined by the associated value, or interpolated value, at that point. Again a colour transfer function will achieve this.

3.3.7 Scalar field over 1D

This we mention largely for completeness: the conventional approach is to draw a graph, relying on one of the interpolation methods described in section 3.3.3 to fill in between data points.

3.3.8 AVS

Scalar field over 3D

Note: AVS has separate modules for different datatypes - ucd data is treated by a separate set of modules from AVS field data - field data can be uniform, rectilinear or irregular grids.

- *Surface Extraction*

  Slicing: The orthogonal slicer module takes a slice through a 3D scalar field, perpendicular to one of the axes.

  The arbitrary slicer module takes a slice at any orientation. The thresholded slicer module is similar, but any values outside a range are mapped to zero in the output slice.

For ucd data, the slicing module is ucd rslice.

- *Isosurface*

  The isosurface module generates an isosurface from a 3D scalar field. The particular algorithm used is not described in the manual. Likewise the way that gradients are calculated is not described.

  The ucd iso module does the same for ucd data.

  The surface may be coloured by a second scalar field.

- *Volume Render*

  The tracer module creates a ray cast image from volume data. There are two interpolation methods: *voxel approximation* which is similar to nearest neighbour in that the voxels are deemed to have constant colour and opacity, but the value is taken from the upper left corner of the cell; and *trilinear* interpolation.

  The cube module is a general tool, adapted from SunVision, which has four options: texture - which shows texture-mapped exterior surfaces of the volume; maximum - which just projects the maximum voxel along each ray; ray cast and create surfaces - which render surfaces at different density levels by ray casting.

  The xray module is a fast volume renderer, giving orthographic views only of the data.

  The volume render module - the exact technique being used is not clear from manual - allows the rendering of volume and surface data in combination.

Vector field over 3D

Note: The AVS modules described below apply to static vector fields.

- *Arrows*

  The hedgehog module shows arrows at locations in the 3D volume - for AVS node data. These locations may be generated first by the sampler module, and can be: single point; points on line or circle; points on plane; points in volume; data points themselves.

  The arrows can be drawn with or without arrowheads.

  Note: A scalar field can be viewed in conjunction with the hedgehog by colouring the arrows according to the value of the scalar field at that point.

  The ucd hog module does the same for data defined in the AVS ucd data structure.

- *Particle advection*

The particle advector module releases a grid of particles into the field. As with hedgehog, the initial sample of points can be generated by the sampler module. The particles can be displayed as a tracer of specified length, and batches can be released.

The ODEs can be solved either by Euler's method, or Runge-Kutta (order of R-K method not specified in manual). Interpolation method is not specified.

There is no standard UCD Particle Advector module but a public domain module ucd_particle makes use of the ucd_streamline module to display particles along each streamline segment.

- *Stream lines*

The stream lines module generates stream lines for a vector field. The user can specify the initial sample of points from which the stream lines are drawn.

The ODEs can be solved either by Euler's method, or Runge-Kutta (order of R-K method not specified in manual). Interpolation method is not specified.

The ucd streamline module is similar for ucd data. The ODE method has choice between Euler, 2nd order R-K and 3rd order R-K.

- *Stream surfaces and ribbons*

The stream lines module can optionally create a surface connecting the stream lines.

The ucd streamline module can optionally create ribbons of specified width.

Scalar field over 2D

- *Contouring*

The contour to geom module constructs isolines from a 2D scalar field for a specified threshold value. The particular technique used is not described.

The ucd isolines module creates isolines on the exterior boundary of a UCD structure.

- *Image Display*

The colorizer module converts data at each point of a scalar field to a colour.

Scalar field over 1D

AVS provides the module AVS/Graph (contained in AVS release 5.02 which is now available) for producing traditional 2D plots from 1D data. The module has been built using some of the Toolmaster-agX libraries and is an example of AVS and UNIRAS integration. The module provides a number of different representations of the data:

- line and curves

- scatter plots
- bar and area charts
- polar and pie charts

The module provides the usual control over parameters and annotation facilities: title, axes, limits, line/curve/marker styles etc.

3.3.9 IBM Data Explorer

Scalar field over 3D

- *Surface Extraction*

  Slicing: The Slice module extracts an orthogonal slice through a volume of data, and this can be passed to a 2D visualization technique.Slice extracts an n-1 dimensional slice from an n dimensional dataset. The nth dimension is removed, i.e. a Slice from a 2D dataset produces a 1D line, a Slice through a 3D dataset produces a 2D grid (with no third dimension).

  Slab does a similar job to Slice except it maintains the nth dimension of the data and allows the slab thickness to be controlled.

  MapToPlane allows users to define non-orthogonal slices by controlling both the position of and normal to the plane

  Isosurface: The Isosurface module creates an isosurface from a *scalar field*. The particular algorithm used is an implementation of marching tetrahedra [41].

  The gradient vector (for shading) can be generated internally (again how it does it is not specified), or it can be supplied explicitly. The Gradient module will calculate the gradient of the scalar field, and this can feed into the Isosurface for shading.

- *Volume Render*

  The Render module creates a volume rendering of volume data (alternative modules are Display and Image. The data is of the "scalar field" datatype. The algorithm used is specified in [48].

  Note that the module can handle a combination of volume data and surfaces.

Vector field over 3D

Static Field

- *Particle Advection, Streamlines, Stream Ribbons*

  The Streamline module produces streamlines for static vector fields. It traces the flow of a particle released from a defined point. The tracing is done by proceeding step-by-step, each step being in the direction of the vector field at the current point; this direction is found by interpolation from the vector data. The user has control over the step length. (This would appear equivalent to Euler's method.)

There is a sophisticated set of controls. The starting points can be given as a list of points, or as a geometric entity - which could be an isosurface for example. The lines themselves can be drawn as lines, ribbons (by connecting to the Ribbon module), or 3D tubes (by connecting to the Tube module).

Time Varying Field

- *Streaklines*

The Streaklines module produces streaklines for time-varying vector fields. This works as follows. Successive vector fields advancing in time are passed to the module. The streaklines are traced step-by-step as for streamlines, but the direction is calculated now by interpolation in space and time.

Scalar field over 2D

- *Contouring*

The Isosurface module will create isolines from 2D data. A number of threshold values can be specified. The technique used is not described.

The Band module, together with the Autocolor module, will create a visualization with coloured regions denoting ranges of values.

- *Surface View*

The Rubber Sheet module can be used to create a surface view.

- *Image Display*

The Autocolor module converts data at each point of a scalar field to a colour.

Scalar field over 1D

IBM Data Explorer provides a Plot module to provide traditional 2D graphics plot of 1D data. The module provides all the facilities to alter parameters and annotation features associated with the plots. These include: axes labels, title, axes style (linear and logarithmic), tickmarks and colour legends.

There are also some other related modules that can be used in conjunction with the Plot module to produce other 2D representations of data.

- Histogram regroups your data into a specified number of bins. The output of Histogram is a new field which are the bars on the histogram
- Connecting Histogram through AutoColor then Plot produces a colored plot of the data dis-

tribution.

3.3.10 IRIS Explorer

Scalar field over 3D

• *Surface Extraction*

Slicing: The Slice module slices a uniform 3D lattice by taking regular samples on a cutting plane, which may be at any orientation. The output is a 2D lattice of the same datatype and number of channels as the input lattice.

The SliceLat module takes a slice through a 3D lattice and outputs the coloured plane as geometry. The lattice cells are intersected with the slice plane, forming polygons. The polygon vertices are coloured by values interpolated from the lattice values.

Other slicing options are provided by Orthoslice, which generates 2D slices which are oriented with the coordinates of the input 3D lattice, and MultiSlice, which gives geometry output from a number of slice planes.

• *Isosurface*

The IsosurfaceLat module generates an isosurface from a scalar 3D lattice. The algorithm used in earlier releases of IRIS Explorer was based on Marching Cubes, although this has been updated in version 2.2.

The gradient vector (for shading) can be generated internally from the gradient of the scalar field (the Smooth parameter “on”). Alternatively the renderer can generate vertex normals from the geometry of the triangular mesh (the Smooth parameter “off”).

The surface may be optionally coloured according to the value of another lattice. The colours used can be controlled by means of a colourmap, which may optionally be passed to the module.

The Contour module generates a set of isosurfaces (i.e. 3D contours) in wireframe. The user has control over the minimum and maximum isolevels, and the number of contours to be calculated. They are generated by calculating contour lines in orthogonal 2D slices of the 3D dataset; the user can control the direction of the slices to be used.

• *Volume Render*

The VolumeToGeom module The VolumeToGeom module uses a volume rendering algorithm to convert a 3D lattice into a geometry. The method used is the so-called hierarchical splatting algorithm of Laur and Hanrahan [45], which fills the space within the volume using screen-oriented planar shapes (or splats). The algorithm uses a small number of large splats in uniform regions of the volume, and fills more detailed areas with a lot of smaller splats. The user has control over the error tolerance associated with the subdivision of the volume, and also over the size and type of the splat, which gives interactive control over rendering time and quality. The geometry produced may be combined with other geometry in the rendering of the final scene (see section 3.4.7), unlike the VolumeRender module (below) which performs direct volume rendering on the input lattice.

The VolumeRender module performs volume rendering on a 3D byte lattice. It offers the choice between two algorithms offering different speed and quality advantages, so the user can trade off between interactivity and quality. The “Transform” algorithm gives interactive rendering speeds for moderately sized datasets, while the “Slicer” algorithm produces high quality images with longer rendering times. In the latter case, the user has interactive control over the number of slices (and so, the quality of the final image).

Vector field over 3D

• *Arrows*

The Vectors module displays a 3D vector field for a 1D, 2D or 3D lattice having 1,2 or 3 channels (i.e. vectors having 1,2 or 3 components). Vectors are located at the lattice coordinate locations and point in the direction of the data vector field. The vectors may be optionally coloured using a scalar lattice of the same dimensionality and size. They may be displayed as lines, tubes (cylinders) or arrows (cylinders plus cones).

The VectorGen module is a simplified version of the same module - here, each vector is represented as a bi-coloured line segment.

• *Particle Advection, Streamlines, and Streaklines*

The Streakline module calculates a streakline through a velocity field. It stops when either the new velocity is 0, or when the streak intersects the bounds of the velocity field. The user has control over how each iteration is calculated and displayed. The full solution can be shown or each iteration as it is calculated. The current particle path is shown in red with the old paths in green.

Particle advection is provided by the module ParticleAdvect which was developed as part of the NCSA Pathfinder project (`http://redrock.ncsa.uiuc.edu/PATHFINDER/pathrel2/explorer/ParticleAdvect/ParticleAdvect.html`)

Scalar field over 2D

• *Contouring*

The contour module generates contour lines for a 2D lattice (or a 3D lattice - see above). For lattices with several channels, the user can select the channel to be contoured. The lines are coloured according to an internal colourmap; this can be optionally overridden by passing another colourmap to the module.

• *Surface View*

The DisplaceLat displaces the coordinates of one input lattice by the data values of the other. The two input lattices can be the same. This can be used to create a surface view of a 2D lattice. The user can control the amount of displacement interactively.

- *Image Display*

  The LatToGeom module creates geometry from a 1D or 2D lattice i.e., it produces lines or sheets, depending on the dimensionality of the input lattice. The user can control whether the geometry is produced as points, lines or (in the case of sheets) polygons. The geometry can be optionally coloured via an input colourmap.

  The DisplayImg module displays 2D lattices as images. IRIS Explorer also contains a number of image processing modules, which may be used to filter or modify the image before display. See section 3.4.7, below, for more details.

Scalar field over 1D

IRIS Explorer currently has two modules which support traditional 2D representations of 1D scalar data. These are:

- Gnuplot: This is an implementation of gnuplot 3.0. Some additions to standard gnuplot have been made to allow the plot/splot commands to extract data and coordinate information from the input lattice.
- Graph: This module plots an X-Y graph of one or more channels of data from a 1-D lattice. The Draw Mode selector allows the user to select between single channel plotting and drawing of all channels.

There is also a Histogram module which can be used in conjunction with the above.

IRIS Explorer release 3.0 will also have NAGGraph with more advanced features than Graph.

3.3.11 Khoros

All the image operators are written tooperate on width-height planes of the polymorphic model. If the depth, time, or elements dimensions of data object are greater than one, the operation is repeated for each width-height plane.

Scalar field over 3D

There is an isosurface operator in the Geometry Toolbox which produces an isosurface constructed of triangles. An orthogonal slicer operator is also available. Mapping operators are provided which can map the scalar values contained in the field into RGB-alpha values.

Vector field over 3D

There are currently no operators which directly address the visualization of vector fields, although it is possible to produce scalar fields from the vector fields using the various arithmetic operators found in the Datamanip Toolbox.

Scalar field over 2D

A two-dimensional field can be produced from scattered location points using the gridding operator found in the Geometry Toolbox. The orthogonal slicer in the geometry toolbox can be used to slice 1D lines from the 2D fields. Two dimensional fields are generally visualized as images or 3D plots, but this is done through various interactive applications.

There are numerous operators in the Image Toolbox which are general image processing operators and more information is given in section 3.4.8.

Xprism can also produce line, mesh, coloured contour and shaded 3D plots. There are also facilities to alter annotation and parameters associated with these plots within Xprism.

Scalar field over 1D

Xprism can produce a number of traditional 2D plots from 1D data. The Xprism application in the Envision Toolbox provides fifteen different 2D plot types and these include:

- line and scatter plots
- bar charts
- discrete plots

The Xprism application also provides full control over fonts, colours, axes, marker types, line types, titles etc.

3.3.12 PV-WAVE

Scalar field over 3D

- *Surface Extraction*

  Isosurface: The SHADEVOLUME procedure calculates an isosurface from data in cubical cell arrangement. The algorithm used is that described in [40]. It is said to be variation on marching cubes. There is also an interactive procedure called WgIsoSurfTool.

  The gradient calculation for shading is not described in the manual.

  Slicing: The interactive procedure WgSliceTool allows the user to slice 3D scalar data.

- *Volume Render*

  The VOLREND function in the Advanced Rendering Library creates a volume rendering from data in cubical cell arrangement.

Vector field over 3D

There is only an “arrow” capability for this visualization datatype.

- *Arrows*

The VECTORFIELD3 function in the Advanced Rendering Library plots vector glyphs at a set of points specified by user.

Scalar field over 2D

- *Contouring*

The CONTOUR procedure draws a contour plot from a uniform 2D grid, with very sophisticated control over the annotation and presentation. There are two methods: one draws all lines per cell then moves on to next cell; the other traces a line through all cells, then moves to next line. No further algorithmic details are given. In addition to the uniform 2D grid used by CONTOUR there are a range of non-linear gridding functions.

The IMAGE_CONT procedure overlays a contour plot onto an image display. Bilinear interpolation is used in image display.

The SHOW3 procedure overlays image, surface view and contours.

- Surface View

The SURFACE procedure draws a surface view, with sophisticated control over the annotation and presentation.

- *Image Display*

See contouring above.

Scalar field over 1D

PV-WAVE has a number of specific view windows for producing 2D plots from 1D data:

- 2D line
- 2D scatter
- 2D histogram

Each view window provides control over numerous annotation and parameters associated with each view type.

3.4 Presentation

3.4.1 Introduction

Presentation is the computer graphics part of Visualization. It includes the rendering (typically to a computer screen) of images, geometric models, volumes and hybrids of these. It also encompasses direct interaction or manipulation, i.e. selecting and moving objects, probing, and so on. This section does not address rendering to other media such as PostScript, video, etc. which is covered elsewhere.

3.4.2 Rendering

Genuine 3D display devices do exist which allow you to walk around and see objects from different angles. However they are large, expensive, experimental and often have low spatial and colour resolution. In most cases Abstract Visualization Objects [19] are rendered to a 2D screen or, for stereoscopic vision, two screens - one for each eye. When mounted into moveable, trackable object such as a boom box or helmet one can once again appear to walk around the objects. This virtual reality is currently a more promising direction than true 3D displays and potentially has great application to visualization e.g., the NASA Virtual Wind Tunnel project [13].

Images

Rendering images typically involves a one-to-one mapping from image pixels to screen pixels. Panning is readily provided for in a windowing environment, and zoom is often provided by bilinear or bicubic interpolation.

Geometry

Rendering geometric models uses classic computer graphics principles, with hidden surface removal, local illumination models, shading, and transformation to a 2D image with perspective. The algorithms for doing this are well established and some may be implemented in hardware.

Volume

To render volume data, it is assumed that classification has already been done. Volume datasets which have had isosurfaces extracted produce a geometric model which is readily rendered as described above. A problem with surface extraction methods is that they do not allow views inside volumes; even if many semi-transparent shells are rendered, detail is lost between the layers. Rendering the volume data without recourse to intermediate geometric objects avoids these problems and also permits weak, fuzzy surfaces or gradual gradients to be rendered. There are two major approaches to rendering volume data: direct ray casting and splatting.

In *Direct Ray Casting* a ray is cast from the eyepoint through each pixel into the volume. Along each ray, regular samples of the colour and opacity are taken. As the volume need not be axis aligned with the viewing plane, this step involves interpolation, and can use any of the techniques described in the earlier section. The final colour for the pixel is obtained by accumulating the colour and opacity values at the samples. A classic paper on this approach is that by Levoy [46]. This differs from ray tracing in that rays are not reflected from the surface of objects; all rays are perpendicular to the image plane.

A variation, optimized for speed rather than accuracy, is *Maximum Intensity Projection*. This is where the volume has axes aligned with the view plane and the highest value along a ray is used as the colour of the pixel - there is no accumulation.

In *Splatting,* rather than project from the image plane into the volume, the footprint of each voxel is projected onto the image plane. This footprint is typically a Gaussian distribution but this may be simplified to a triangle or step function, trading accuracy for rendering speed. The method was first described by Westover [65].

Hybrid

Hybrid methods are used to present scenes containing a mixture of image, geometric and volume elements. For example, a prosthetic hip replacement may be matched up with a CT scan of the patients hip. Problems arise when mixing the different rendering types together, to ensure correct occlusion of objects and to allow picking of interpenetrating geometric and volume objects. In some cases, all the data must be re-rendered if one small part of the model changes. Typical strategies include converting all objects to one type (surface extraction turns volumes into geometric objects, conversely geometric objects can be voxelised) or merging the intermediate results of rendering using some depth sorting method.

3.4.3 Manipulation

Viewpoint selection

The simplest form of direct manipulation is selecting a different view of the scene, either by moving the camera or moving the objects in the scene. On a 2D screen this is typically done with sliders, dials etc. in a dialog box, or (better) by gesturing with the pointing device. A particularly useful analogy used by some systems is a *glass trackball* which conceptually encases the displayed scene and is moved by dragging the mouse. Spaceballs may be used to achieve a similar result. In a 3D environment, gesturing with a dataglove is often used.

Picking

A refinement on moving the whole scene is to select a component of it. This may indicate to the visualization software that a particular object is to be used for the next interactive operation. It represents information flowing backwards through the visualization process, from rendering to mapping or to filtering.

Probing

An object is inserted into the scene where it samples the underlying model and reports back the data values. For example colour may be used to depict pressure on the surface of an object. Colours could of course be compared by eye with the colour scale, but a probe can be positioned anywhere and will give a reading directly in kiloPascals or other suitable units. A probe reports its geometric position in the scene to earlier stages of the visualization process, which then generate the requested data. This need not be numeric; other Abstract Visualization Objects are often produced. For example a probe for a flow field might produce a solid arrow whose direction and length indicated the components of the vector flow field. Other quantities such as the curl or divergence of the field might be probed and represented as twisting of the arrow shaft, or colour of its tip.

3.4.4 Hardware support

High performance graphics workstations have dedicated hardware for accelerating various stages of the geometric rendering pipeline such as hidden line and surface removal, perspective transformation, texture mapping, and local illumination models. Less commonly, they have support for image rendering, such as bicubic zoom, or for volume rendering, such as fast, hardware assisted trilinear interpolation. This is particularly important as direct volume rendering is compute intensive and real-time performance is a much sought-after goal. Support for hybrid rendering is currently rare, particularly if picking is to be supported. General purpose visualization software may not support all the capabilities of a particular platform.

Some workstations have support for presenting stereo images - using either liquid crystal shutters to present alternate images or using twin displays in a headset - and for 3D interaction using devices like spaceballs, datagloves, and the like. These capabilities are currently rather manufacturer specific, so again it may be difficult for general visualization systems to take advantage of them.

3.4.5 Application Visualization System (AVS)

AVS uses three separate components to view different types of object. There is an image viewer, a geometry viewer, and a tracer for volume data, which produces an image. This means that hybrid scenes can only be displayed and interacted with by converting all data to geometric form. 24 bit TrueColor or DirectColor displays are supported. 8 bit displays will use 216 colours from a 6x6x6 RGB cube; while fine for general use, this is severely sub-optimal for greyscale image display with only six grey levels. Typical medical imaging systems use 250 grey levels with the other six colours mapped onto bright primary and secondary colours for annotation

Image display

Collections of images can be displayed. Zoom and stacking are supported; images can be labelled and there is a flipbook animation capability. A variety of dithering techniques can be selected. The unsupported alpha blend module allows the compositing of stacked images with variable transparency but this requires hardware support. Separate modules provide two probes: one gives the colour at a point, the other measures the distance between points.

Geometric objects

The geometry viewer supports both a hardware renderer and a software renderer. The latter supports the full functionality of AVS but may be slow. The hardware renderer will take advantage of the native graphics system - PHIGS, GL, Starbase - but in this case there is no software fallback for missing functionality. If a particular hardware system does not support, for example, transparency or texture mapping, the hardware renderer will display all objects as opaque or plain, respectively. Spaceballs are supported on SGI platforms. Stereo is supported on Silicon Graphics, Evans & Sutherland and Kubota platforms.

The next release of AVS (AVS6) will extend the graphics library support to include PEXlib and OpenGL.

Objects may be translated, rotated and scaled using a glass trackball paradigm. They may be coloured, texture mapped and their other surface properties such as specularity altered. Lights of various colours may be positioned in the scene.

Pick information can be sent to other modules and a variety of other modules provide probes. The probe module reports data values and can use either nearest neighbour or trilinear interpolation.

Volumetric objects and hybrid rendering

These are dealt with in a variety of ways. One option is to use the volume render module, which have variable colour and opacity and may be sent to the geometry viewer. This is one way to do hybrid scenes. There is no control over the lighting although the gradient shade module helps with this, and rendering can be slow. This method really requires hardware support for 3D texture mapping and transparency.

Fast views of axis aligned volumetric data are provided by the xray module which provides options for finding the sum, minimum, mean, median, minimum or maximum value for each pixel.

A third alternative is the cube module from the SunVision toolkit, intended for classified volumetric data. This does not appear well integrated with the other modules. Four rendering methods are supported: texture mapped external surfaces, a maximum intensity projection method which need not be axis aligned, ray casting and surface extraction. The last two methods classify voxels using values set in the edit substances module. Interpolation is nearest neighbour by default but can be set to trilinear. Rotation and translation is by specifying 4x4 matrices although other utility modules can generate these.

Lastly, a pair of modules give direct raycasting. The tracer module does the raycasting and can accept a colourmap for greyscale volumes. The display tracker provides a direct manipulation interface on the resulting image using a glass trackball paradigm. Bilinear zoom can be used on the image without having to repeat the raycasting operation.

Currently there does not appear to be a module to perform splatting.

3.4.6 IBM Data Explorer

DX uses a raft of rendering modules which co-operate to present and interact with objects. There are often alternative ways of doing similar things, depending on the data and interaction required. Unlike other systems, all object types are displayed in a simple and natural way in the same window. This consistency of user interface is a definite benefit.

Three modules are central to the rendering process. Display is the most basic module; it simply displays an image which may be a 2D regular dataset or the output of the Render module. Display does no rendering; it is simply a mechanism to put images on the screen.

Image display and Geometric objects

Image is a DX Macro and contains modules such as: AutoAxes, Render, Display, Camera. The Image tool is the most frequently used method of rendering and display in DX as it supports direct manipulations of the displayed scene such rotation, translation, zoom, navigate etc. Users should avoid passing images e.g. a Landsat image, to the Image tool as it will be rendered as if it where a 2D dataset prior to display. To simply display image data the Display module should be used.

Render is the most powerful module as it renders one or more geometric or voxelised objects and presents them.

Objects and cameras can be translated and rotated. The viewing model is easy to use, being based on a look-to point rather than a camera position. Object properties such as colour, normals, specularity can be modified. Point lights and ambient light are supported. The renderer appears to use Gouraud and Phong local illumination models.

Pick data may be sent to other modules, and a variety of probes are available. These send their 3D position, or a list of positions, to other modules. A measuring probe calculates the area and volume of objects.

Volumetric objects and hybrid rendering

Rendering hybrid scenes is readily performed subject to a few limitations - interpenetrating volumes are not supported, and volumes are not rendered with perspective. However, volumes need not conform to a regular rectangular grid. The documentation stated that volumes were rendered by ‘one of a variety of irregular and regular volume rendering algorithms’ which appeared to mean direct ray casting using a dense emitter model: opacity gives the absorption per unit thickness and the colour relates to light emission per unit thickness: a self-luminous gel. Volumes are composited front to back with geometric objects.

3.4.7 IRIS Explorer

Image display

Images in IRIS Explorer are passed through the system as 2D multichannel lattices. Thus, all modules that accept lattices as input can be used to manipulate images - either by modifying the coordinates part of the lattice (for image cropping, scaling, rotation, etc.) or the data part of the lattice (for image filtering, blurring, edge detection, etc.). Much of the image processing functionality in IRIS Explorer is provided via the ImageVision library, an object-oriented toolkit for the manipulation, processing and display of image data. A special feature here is that ImageVision modules can be chained together to make use of ImageVision’s so-called pull model for passing only the region of interest of an image along the chain. This leads to greater efficiency, especially when dealing with large images.

The DisplayImg module displays 2D lattices as images. The module accepts multiple input lattices; each is displayed as a separate image, and each can be separately managed, manipulated and updated. Lattices of any datatype and any number of channels are allowed - thus, single-channel input is displayed as monochrome, while 3-channel input is displayed as RGB.

Geometric objects

Geometry in IRIS Explorer is implemented using Inventor, an object-oriented 3D toolkit. Earlier releases of IRIS Explorer were based on IRIS Inventor 1.0, which used the IRIS GL for rendering geometry. The latest release of IRIS Explorer (2.2) is based on Open Inventor 2.0, which uses OpenGL for rendering.

The geometry type in IRIS Explorer is an Inventor object. This means that geometry can be shared between IRIS Explorer and other Inventor applications (for example, SGI's Showcase presentation package) for display and manipulation outside IRIS Explorer. Similarly, 3D geometries from other packages can be read into IRIS Explorer once they have been translated into Inventor file format (see chapter 4, below). It also means that modules can be written (see Chapter 6: Incorporating Application Code) which make calls to the Inventor API to create and modify geometry within IRIS Explorer (a simplified geometry API is also supplied with IRIS Explorer which provides some of the same functionality). Finally, IRIS Explorer is able to inherit and make use of much of the functionality of Inventor for 3D geometry creation, manipulation and display, which is very sophisticated.

The main module for geometry display and interaction is Render. This allows a rich set of controls over the scene, including

- Changing camera parameters via a viewing model. By default, this model allows for examining the scene through the 'glass trackball' analogy mentioned above in section 3.4.3, although other viewers are available which mimic flying, walking, etc. through the scene.
- Selecting objects in the scene. The user can then edit the object's appearance (using Inventor controls over components of the lighting model), colour (using Inventor colour controls), and its scale, orientation and location (using Inventor 3D manipulators).
- Picking objects. Here, information about the selected object is written to an IRIS Explorer data type which can be passed to another module for further processing. Open Inventor's 3D picking returns useful information such as the point of intersection, normal at the point of intersection, the nearest vertex, and more.
- Controlling lighting. Besides being able to manipulate the properties of the material of which objects in the scene are composed, the user is able to create and edit various types of lights in the scene - again, making use of the sophisticated Inventor interface to do this.

Finally, it should be noted that the Render module can combine multiple geometries into a single scene, irrespective of their origin. Thus, a user could, for example, display a scene made up of a wireframe box, an isosurface, a volume rendered lattice, an imported 3D model and a slice through a vector field. This is another benefit of the flexibility of the Inventor 3D toolkit which IRIS Explorer uses for its geometry data type

Volumetric objects and hybrid rendering

A volume to geometry module does direct volume rendering by splatting. There is support for hybrid rendering as the output may be fed into the render module.

3.4.8 Khoros

The interactive data presentation routines in the Khoros system can be found in the Envision and Geometry Toolboxes. The Envision Toolbox provides a number of applications for interactively exploring multidimensional data. The data can be visualized as images, surfaces, 2D plots, or 3D plots. The Geometry Toolbox contains an interactive geometry and volume renderer. All data presentation routines interpret data according to the polymorphic and geometry data models.

Image display

There are a number of image display applications which are all in the Envision toolbox. These applications are Editimage, Putimage, Animate, and Spectrum. Editimage is an interactive image display program which provides a zooming capability, colourmap editing, false colouring capabilities, and image value display. Animate is an sequence display tool. Putimage is a non-interactive image display program. Spectrum is an interactive program for exploring multi-dimensional data.

These applications all use the Khoros image visual object. This visual object is capable of displaying image stored in any data type. Data types other than byte are automatically normalized between 0 and 255. A standard 256 entry colourmap can then easily be applied to the normalized image through the image visual object. Complex data types are converted to floating point using real, imaginary, magnitude, or log magnitude conversion before the normalization occurs. The image visual object is capable of displaying to either 8 or 24 bit displays. A private colourmap is used. On an 8 bit display, 24 bit images will be displayed using a fast 332 quantization. Large images may be displayed using a pan icon. The image data is cached such that the entire large image is never all in memory at any one time. This is also true of the animation display with large animations.

An image probing capability is available in Editimage. The data values at and around the pixel indicated by the mouse are displayed. The explicit world-coordinate position of the pixel is also displayed if explicit location data is available.

Geometric objects

The RenderMonster application in the Geometry Toolbox provides the geometry visualization capability to the Khoros system. Implemented as a software renderer, RenderMonster interactively produces either 24 bit true-color rendered images, or 8 bit rendered images. Alpha compositing is used to render solid and semi-transparent objects together.

The Xprism application in the Envision Toolbox provides fifteen different 2D plot types and nine different 3D plottypes. It supports multiple plots per area and multiple plot areas. All details of the axis, tic marks, labels, colors, line styles, and annotations are easily modified by the user. The built-in expression parser allows the user to create complex data arrays interactively.

Geometry Transformations and Viewpoint Selection in RenderMonster can be performed interactively on the rendered image using the mouse. Pressing different mouse buttons on the rendered image and moving the mouse will perform scalings, rotations, and translations. A bounding box is used to show the transformation interactively as it is being performed. When the bounding box has been transformed to the desired position, a new rendering is done.

Khoros runs on any hardware with an X display, regardless of it being 8 bit or 24 bit. No special hardware capabilities are required, nor are they used if present. There are plans however on doing a GL port of RenderMonster.

