Visualization Cookbook Using AVS/Express

International AVS Centre
Manchester Visualization Centre

November 2001

Acknowledgement

This cookbook was written by Priya Dey, a student studying Computer Science at the University of Manchester. This book was created using Microsoft Word 2000 during summer 2001, while working as a summer placement student at the Manchester Visualization Centre.

I would like to thank the UK AVS and UNIRAS User Group for funding and making this project possible. Thanks are also due to my supervisors, Tobias Schiebeck and Matt Cooper, for their assistance with the cookbook, and Yien Kwok and the other staff at Manchester Visualization Centre for their continued help throughout the placement.

Contents

Figures

1 Introduction

AVS, Advanced Visual Systems[1], provides a visual development tool, which allows the user to build reusable modules for visualizing, analysing, manipulating and displaying data. AVS/Express is a modular, extensible system with hundreds of predefined components for visualizing data.

This cookbook aims to show how to tackle some specific visualization tasks using AVS/Express. The examples used will cover different scientific subjects, using medical, meteorological and engineering data. This cookbook describes how to get from the raw data to the rendered image.

The cookbook is written for users who are already familiar with AVS/Express. An introductory course on how to use AVS/Express can be found at www.iavsc.org/training [2].

Chapter 2 illustrates visualization techniques using medical data as an example application. Chapter 3 shows an example application using meteorological data.

This cookbook contains coloured images, and is therefore best viewed in colour.

Conventions used:

- Red underlined text signifies a command or button, which can be selected from the screen.
- **Bold blue** text signifies an AVS/Express module name.
- *Italic* text is used to signify the name of a file, or text the user must enter into the computer.
- The figures in Chapter 2 are preceded by the Roman numerals II, and the figures in Chapter 3 are preceded by III.
- A number in square brackets, e.g. [1], denotes a reference. A list of references is given in References.
- All files used or created are stored in the current working directory.

[1] The website for AVS can be found at www.avs.com [1]

2 Visualizing Medical Data

This chapter illustrates an example application using medical data. This describes how to get from the raw data to the rendered image.

The data represents images of a human head. An MRI (Magnetic Resonance Imaging) scanner was used to scan part of the head in 25 slices. Each slice is 3mm thick, so the total length of the head included in the scanned slices is 75 mm. There are 256 by 256 pixels per slice, and each pixel has a size of 1mm by 1mm. The brain has an acoustic neuroma, which is a benign tumour on the sheathing cells of the acoustic nerve. This tumour is known to be 30mm by 30mm by 30 mm, and is found between slices 7 and 16. Five sequences were used in scanning the brain: T1 vol (t1), T1 vol and contrast (t1c), Proton density (pd), T2 (t2) and Inversion recovery (ir). So there are five data files to process, of a form called Analyze, which is an image file format used in biomedical imaging [6]. Each data file has binary data in a file with the extension *.img*, and a corresponding header file, containing extra information about the data. The image data files are called *dg_t1_25.img*, *dg_t1c_25.img*, *dg_pd_25.img*, *dg_t2_25.img* and *dg_ir_25.img*. The corresponding header files have the same filename except that the extension is *.hdr*, and are called *dg_t1_25.hdr*, *dg_t1c_25.hdr*, *dg_pd_25.hdr*, *dg_t2_25.hdr* and *dg_ir_25.hdr*.

2.1 Reading Data

Data can be imported using a field reader. AVS/Express has a module called **Read_Field**, in the DataIO library, (Appendix A Figure A-1), which is the natural choice of module for reading in binary image data. **Read_Field** reads in an AVS field file and converts it to an AVS/Express field. A field file contains metadata with specific information describing the structure and type of data, so that AVS/Express can understand it[2].

The values to put in the field file can be gained from a description of the data. The size of these images is 256 by 256 pixels, therefore dimension 1 (*dim1*) is 256 and dimension 2 (*dim2*) is 256. There are 25 slices, so dimension 3 (*dim3*) is 25. There are three dimensions, so *ndim* equals 3, and *nspace* = 3, because there are three dimensions in co-ordinate space.

There is one data component per element, so *veclen* =1, and the data is 16 bits, so the data type is *short*. The *field* mapping method is *uniform*, because the data is in a regular array, and the dimensions of the computational and co-ordinate space are the same. The *label* given is the name of the data component, for example *t1c*, but this is optional. The *file* name is *dg_t1c_25.img*, and the *filetype* is *binary*. There are no bytes to skip before the data is found, so *skip* = 0, and the *stride* is 1 because the data components are sequential in the file, so a step of one (one short = 16 bits) is needed to reach the next data component.

A field file was made for each sequence. Below is the field file for the data called t1c, found in the *dg_t1c_25.img* file:

```
# AVS field file
# medical data
ndim = 3       # 3 dimensions in computational space
dim1 = 256     # size of dimension 1
```

[2] A description of the format of a field file can be found at www.iavsc.org/training [2]

```
dim2 = 256
dim3 = 25
nspace = 3      # 3 dimensions in co-ordinate space
veclen = 1      # 1 data component per element
data = short    # type of data components
field = uniform # mapping method
label = t1c
variable 1 file=dg_t1c_25.img filetype=binary skip=0 stride=1
```

The field file is created in a text editor such as Nedit or Emacs, and saved with a *.fld* extension, e.g. *t1c.fld*. The field files for the other data files, t1, t2, pd and ir, can be found in Appendix B.

2.2 Visualizing Data

AVS/Express has libraries of predefined modules, which can be joined together to form a network. A network was made to process the data, using the Network Editor (Appendix A Figure A-1).[3]

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, in order to visualize the data.

orthoslice: produces a slice of a structured field perpendicular to a selected co-ordinate axis. This was used because it allows each of the 25 slices to be viewed individually[4].

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

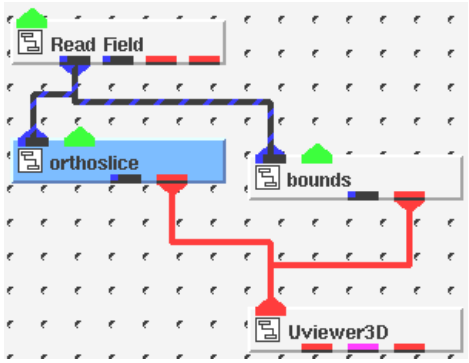

Figure II-1: **orthoslice** Network

[3] For instructions on how to find modules, refer to Appendix A Figures A-4 to A-5.

[4] The **Orthogonal_slicer** or **slice_plane** module could have been used instead of **orthoslice**. **Orthogonal_slicer** has an extra output, which can be connected to the **Uviewer3D**. This shows the bounds of the slice, and can be toggled on and off. Showing the slice boundaries could be useful for showing the original size of the slice, if the image were being cropped, but in this case is not necessary. The **slice_plane** module allows the slice plane to be in any position, not restricted to the co-ordinate axes.

The network in Figure II-1 is used to produce images from the five data files (Section 2.1).

2.3 Altering Images

The network is used to produce an image, which can be manipulated to the desired format. Simple transformations such as scale, translate and rotate can be performed using the icons on the toolbar of the Dataviewer Pad window (Appendix A Figure A-3). The parameters of each module can be altered. From the Editors menu, select Modules, then the module name, and its parameters will be displayed, which can be changed by using the sliding bars to change values, or entering digits. Figure II-2 shows an image produced by the network in Figure II-1. The image has been rotated to show that it is three-dimensional.

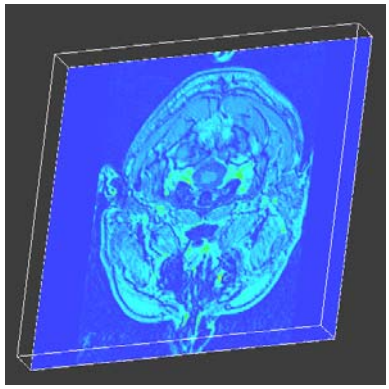

Figure II-2: t1c slice 0, rotated

The **orthoslice** module takes slices through a field. The position of the slice can be changed to view different cross-sections through the field. From the Editors menu, select Modules, then select **orthoslice**, and alter the sliding bars (Appendix A Figures A-3 and A-6). The axis is the coordinate axis the slice is perpendicular to. 0 is the X-axis, 1 is the Y-axis and 2 is the Z-axis. Choose axis 2 to get a cross-sectional view looking down from the top of the head. The plane was altered to change the slice shown. Figures II-3 to II-16 show different slice planes.

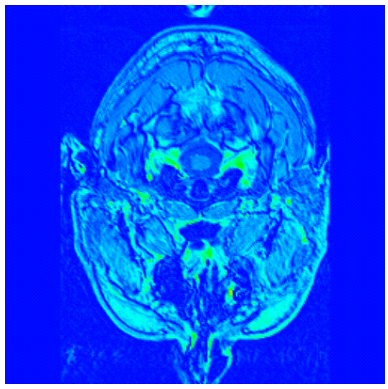

Figure II-3: t1c slice 0

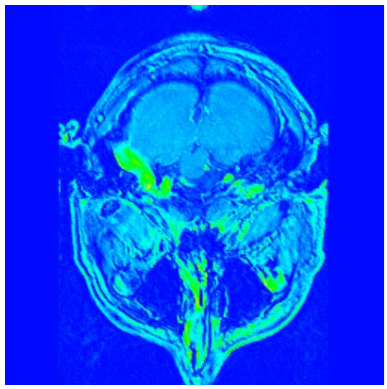

Figure II-4: t1c slice 6

From the information given along with the data files, the tumour was known to be located in slices 7 to 16 (Figures II-5 to II-14). Looking through all the slices confirmed that the tumour was only visible in slice planes 7 to 16.

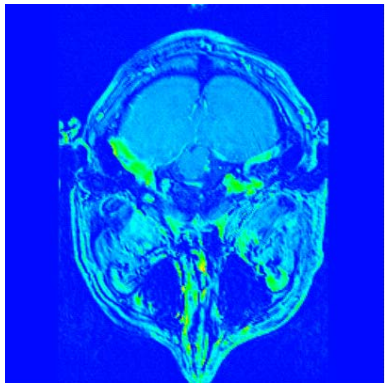

Figure II-5: t1c slice 7

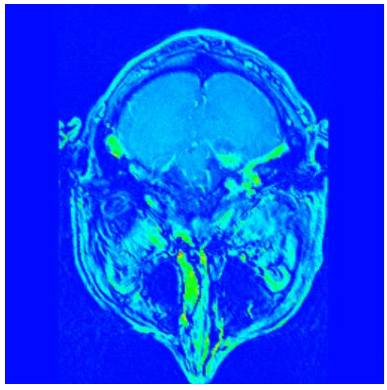

Figure II-6: t1c slice 8

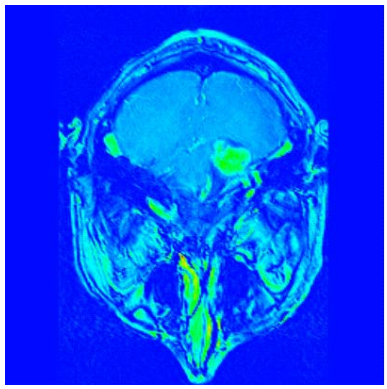

Figure II-7: t1c slice 9

The images show different colours at different data values. The points with the lowest data values are coloured blue, and as the data values increase, the colours change from blue to cyan, green, yellow, orange and red. The tumour tissue has higher data values than the other tissues, so the tumour is coloured green, while the surrounding tissues are cyan and blue.

The tumour is clearly visible in slice 9 (Figure II-7). It looks largest in slice planes 10 to 14 (Figures II-8 to II-12).

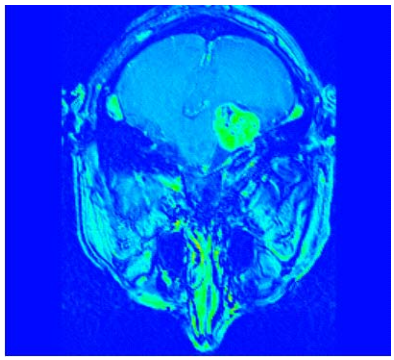

Figure II-8: t1c slice 10

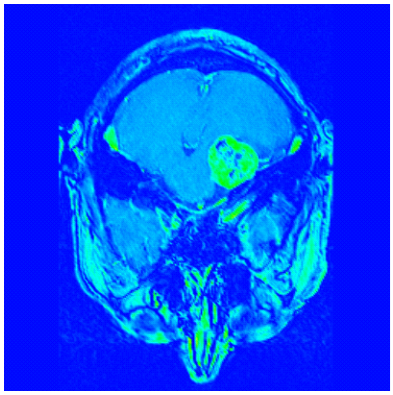

Figure II-9: t1c slice 11

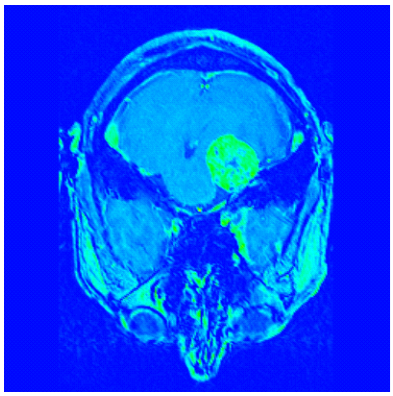

Figure II-10: t1c slice 12

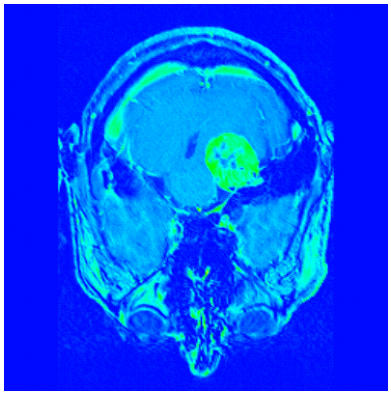

Figure II-11: t1c slice 13

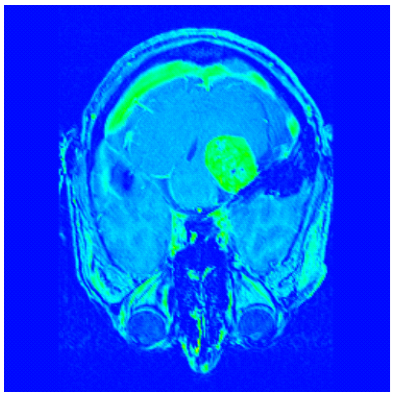

Figure II-12: t1c slice 14

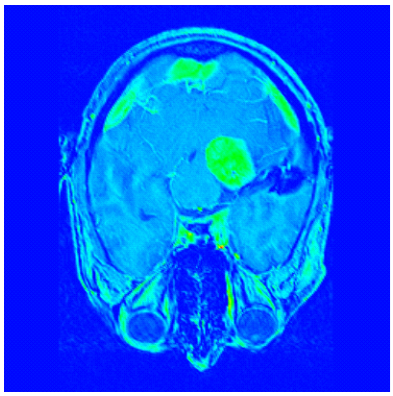

Figure II-13: t1c slice 15

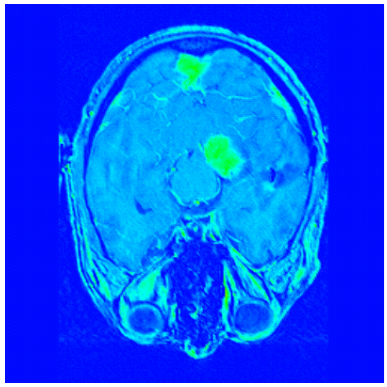

Figure II-14: t1c slice 16

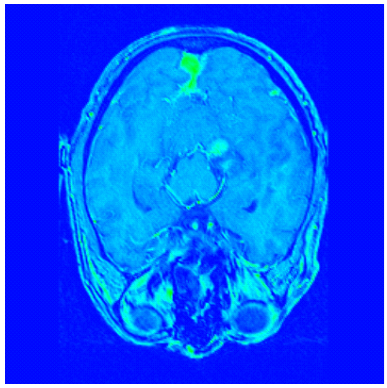

Figure II-15: t1c slice 17

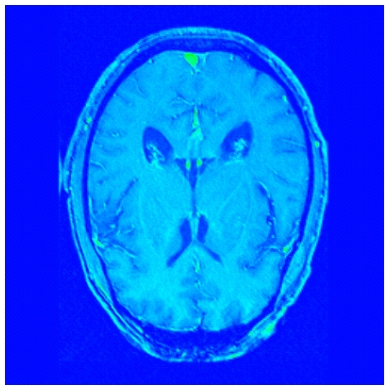

Figure II-16: t1c slice 24

Figure II-16 shows slice 24, the last slice. There is no sign of the tumour visible in this slice.

Slice 10 (Figure II-8) shows the tumour well, in a large size, so this slice was looked at in the other data files – using the field files for the t1, t2, pd and ir data, listed in Appendix B. All of the data files show the tumour, in the same position, which confirms that there is a tumour present.

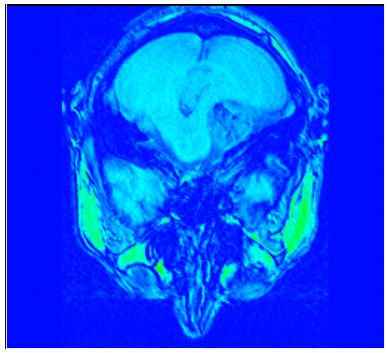

Figure II-17: t1 slice 10

In Figure II-17, t1 slice 10, the tumour is pale blue, the same colour as the surrounding tissue, whereas t1c shows the tumour in green.

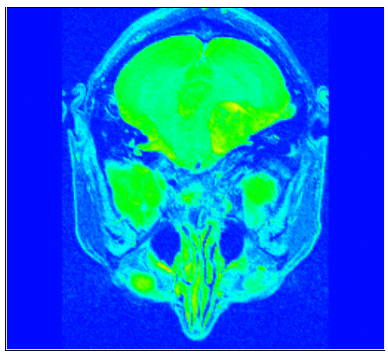

Figure II-18: pd slice 10

In Figure II-18, the tumour is just visible in yellow and green.

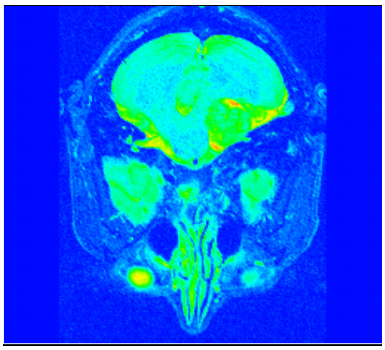

Figure II-19: t2 slice 10

In Figure II-19, the tumour is visible in green, with small red areas at the edges.

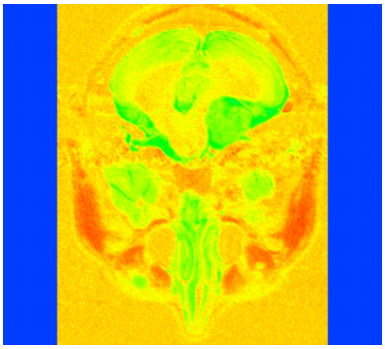

Figure II-20: ir slice 10

In Figure II-20, produced by the ir data file, the colours are very different to those produced by other data files. Most of the head tissues are orange, whereas for other data files the tissues are blue. The tumour is visible in yellow and green.

The next task is to crop the image. The tumour is the interesting part, so it would be useful cut out irrelevant detail. This can be achieved using the **crop** module, from the Filters library (Appendix A Figure A-1).

It is also desired to show the real size of the tumour. The thickness of the slices is actually 3mm, which has not yet been shown. This was achieved by using **readAnalyze**, a module from the International AVS Centre (IAC) library, downloaded from the IAC[5] website. **readAnalyze** is a user macro, which reads data files of the Analyze form, which have binary data in a file and a corresponding header file, with information about the data file (Section 2). **readAnalyze** outputs a uniform field representing the volume of data. It reads a header file and prints information to standard output, then reads the corresponding image file and converts it to uniform field. The header file specifies the spacing of the data in each dimension, and gives a z-spacing of 3. This gives the true thickness of the slices.

[5] Instructions on how to download the module are found at www.iavsc.org/repository. The URL for the module is www.iavsc.org/repository/express.

The **readAnalyze** reader understands the format of the header file, whereas **Read_Field** does not. To achieve the correct z-spacing using **Read_Field** would require the co-ordinates of the data to be altered, because **Read_Field** does not allow the dimensions to have different spacing. Figure II-22 was produced using the network from Figure II-1, which used **Read_Field** to read in the data. Figure II-23 was produced using **readAnalyze** instead of **Read_Field**, as in the network shown in Figure II-21. The **bounds** module shows that the extent of the data in Figure II-23 is 3 times the extent of the data in Figure II-22. This is because **Read_Field** used 1 pixel per data component, whereas **readAnalyze** used 3 pixels per data component, to give the slice its true thickness of 3mm. Section 2.3.3.1 compares the effects of **Read_Field** and **readAnalyze** in volume rendering.

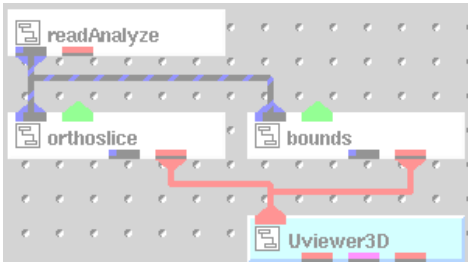

Figure II-21: **readAnalyze** network

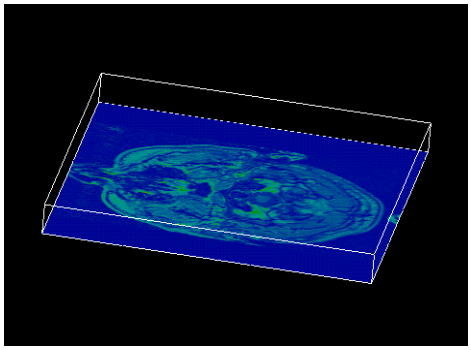

Figure II-22: image produced by **Read_Field**

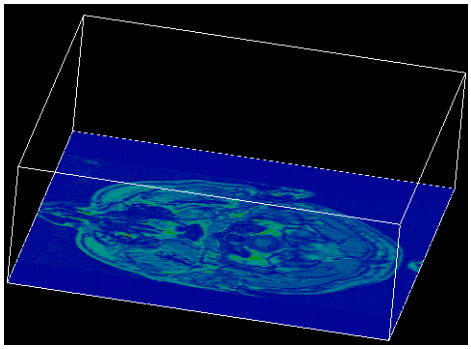

Figure II-23: image produced by **readAnalyze**

2.3.1 Using **readAnalyze, crop** and **isosurface**

Another network was made to process the data.

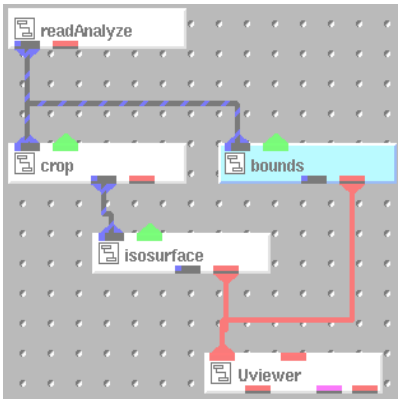

Figure II-24: Network using **crop** and **isosurface**

- **readAnalyze**: reads header file and image file and creates a uniform field. This was used because it can show the true volume of the tumour, which **Read_Field** cannot do. This is because **Read_Field** reads the dimensions of the data given in the field file, which in this case are 256 x 256 x 25, whereas **readAnalyze** reads the header file (Section 2), which gives the true volume as 256 x 256 x 75.

- **crop**: extracts a subset of a uniform field. This was used to cut out uninteresting parts of the image, so the tumour alone can be viewed.

- **isosurface**: creates a surface of constant value. This will show all points with the same data values, so will show parts of the head which are of a similar tissue type to the tumour.

- **bounds**: generates a bounding box of a 3D field. This shows the extent of the volume of the field.

- **Uviewer3D**: creates a 3D image in the Dataviewer Scene window (Appendix A Figure A-2).

It is important to connect **readAnalyze** to **crop**, and then connect **crop** to **isosurface**, so that the isosurface shown will be of the cropped section, not the whole image. The output of **crop** is the input of **isosurface**. The **bounds** module is connected from **readAnalyze** to **Uviewer3D**, so that the extent of all the data is shown, not just the cropped section. To show the bounds of the cropped section, another bounds module could be connected from the output of **crop** to **Uviewer3D**.

Isosurface shows all points with the same data value. It creates a surface of constant value, and is used to give a graphic depiction of the locations of a particular data value in a 3D field. Different tissue types such as bone, and muscle, show up at different isolevels. In order to find the isolevel of the tumour, various isolevels were tried on all of the data (Appendix A Figure A-8).

Figure II-25 shows isolevel 840 throughout all of the data. This showed many tissues in blue, and the tumour was just visible, but hard to see.

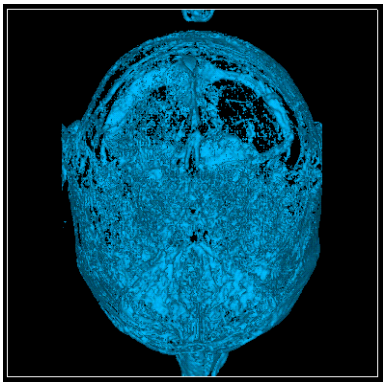

Figure II-25: Isolevel 840

Isolevels 1185, 1434 and 1485 showed many tissues in blue-green, as shown in Figures II-26, II-27 and II-28.

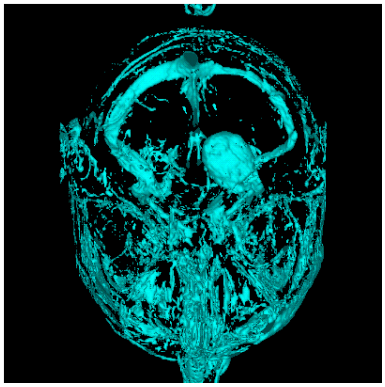

Figure II-26: Isolevel 1185

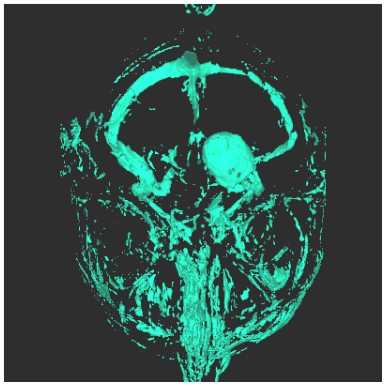

Figure II-27: Isolevel 1434

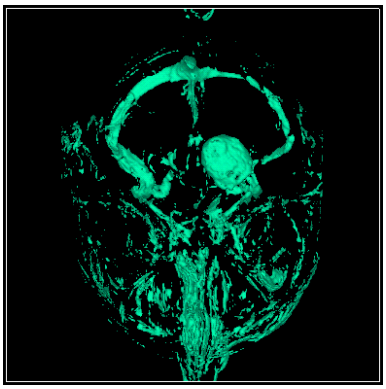

Figure II-28: Isolevel 1485

The tumour shows at isolevel 1709 (Figure II-29), and so do other tissues throughout the head, which suggests that the tumour is the same tissue type as these other tissues.

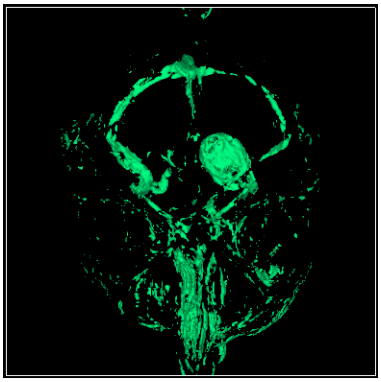

Figure II-29: Isolevel 1709

Isolevel 1737 distinctly showed the tumour (Figure II-30).

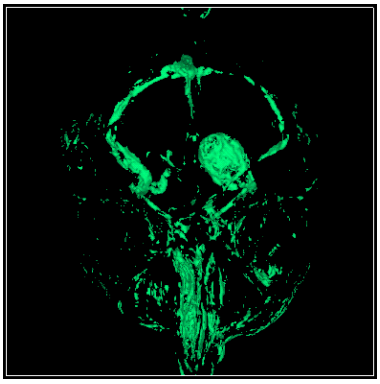

Figure II-30: Isolevel 1737

The tumour showed up well at isolevel 1792 (Figure II-31).

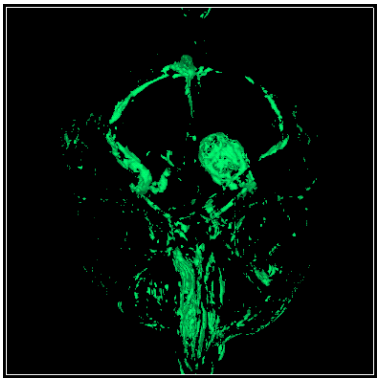

Figure II-31: Isolevel 1792

Isolevel 2040 (Figure II-32) showed the tumour, and other tissues, less clearly than isolevel 1792.

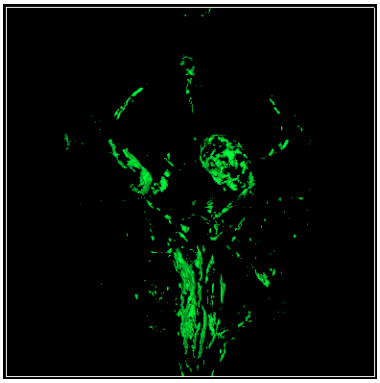

Figure II-32: Isolevel 2040

The parameters of **crop** can be changed, to cut out irrelevant parts of the image. The image was cropped to just show the tumour. First it was cropped in the i-direction, by adjusting the minimum and maximum i-value sliding bars (Appendix A Figure A-7). The minimum sliding bar was increased until the tissues to the left of the tumour were invisible, but the left edge of the tumour was still visible. Then the maximum value was decreased until the right side of the tumour was just visible. This process was repeated for the j minimum and maximum sliding bars, so that the top and bottom edges of the tumour were just visible. The i-values and j-values of the tumour were found to be between 125 and 180. Cropping the k minimum and maximum values changes the planes shown. In Figures II-33 to II-38, the image was cropped to between 7 and 16, because that is where the tumour is located.

Scaling and translating an image of the tumour isosurface gives a close up view. In the network shown in Figure II-33, the **isosurface** module was replaced with an **isovolume** module, in order to compare the effects of **isosurface** (Figure II-34) with **isovolume** (Figure II-35).

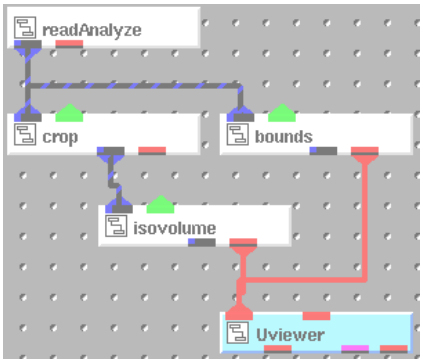

Figure II-33: Network using **crop** and **isovolume**

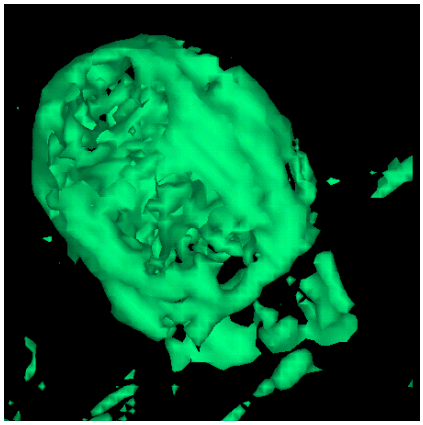

Figure II-34: Isosurface of tumour

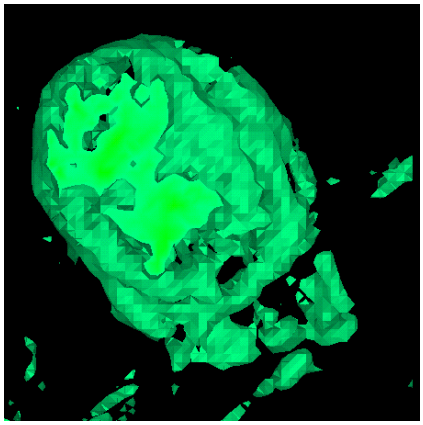

Figure II-35: Isovolume of tumour

The **isovolume** module creates a volume of the data inside or outside an isosurface. Here it has created a 3D field of all the tissue inside the tumour.

Slices 12 and 13 were looked at in more detail, because when **orthoslice** was used, in Figures II-10 and II-11, these slices showed the tumour in a larger size than other slices, and the green tumour had blue tissues clearly visible inside it. The tumour was coloured green and other tissues were coloured blue because they had different data values, which were

coloured according to the default datamap (Section 2.3.2). This suggested that there were different tissue types inside the tumour.

Isovolume slices 12 and 13, scaled up, are shown in Figure II-36. This shows the tumour as hollow inside, which means that the tissue inside does not have the same isolevel as the outside of the tumour. Different tissues have different isolevels, so this confirms that there is a different tissue type inside.

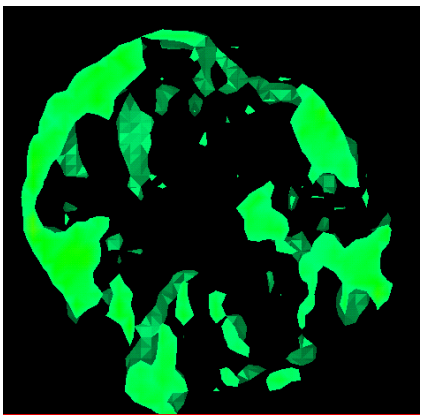

Figure II-36: **isovolume** slices 12 and 13

The effect of cropping an orthoslice was compared with that of cropping an isosurface. The network shown in Figure II-37 is like the network shown in Figure II-33, but uses **orthoslice** instead of **isovolume**.

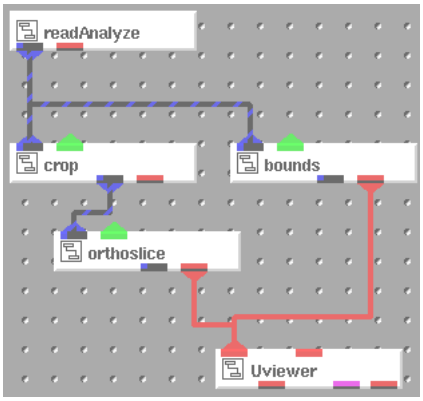

Figure II-37: Network using **crop** and **orthoslice**

In Figures II-38 and II-39, there are two images produced by the network shown in Figure II-37. The k values of **crop** are set to 11 - 12, so only two slices can be viewed (Appendix A Figure A-7). The **orthoslice** plane number 0 gives slice 11, and plane number 1 gives slice 12 (Appendix A Figure A-6).

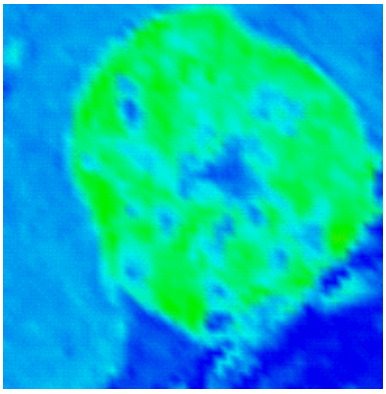

Figure II-38: **orthoslice** 12

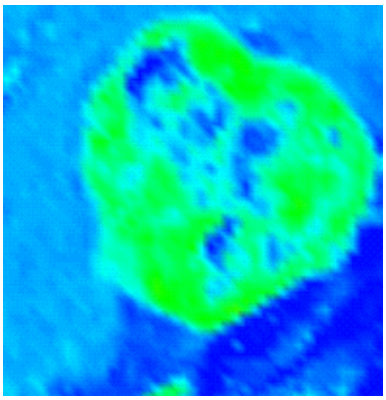

Figure II-39: **orthoslice** 11

Comparing Figure II-38 with Figure II-36, **isovolume** shows just one tissue type - the tumour tissue is shown in green, and the surrounding tissues are not shown because they have different data values, and so have a different isolevel. Using **isovolume**, more than one slice can be seen together – in Figure II-36, slices 12 and 13 are shown together. Using **orthoslice** shows all the tissue types in just one slice. The tumour tissue is shown in green, and all the surrounding tissues are also shown, in different shades of cyan and blue, because the different tissue types have different data values.

2.3.2 Changing the Datamap

The images shown so far have been coloured according to default values, defined by the datamap. The range of colours in the images can be altered, by changing the datamap. This could be useful, for making the tumour a different colour from surrounding tissues, to show that the tumour has a different tissue type, and so that the tumour stands out more. To change the datamap:

- Click on the Select Object… button at the bottom of the Dataviewer Pad window (Appendix A Figure A-3).
- From the Object Selector window, which now appears (Appendix A Figure A-9), select **orthoslice** from the list of items, and click on OK.
- Select Editors from the Dataviewer Pad window, and select Datamap (Appendix A Figure A-10). The Datamap Editor will now appear in the Dataviewer Pad window (Appendix A Figure A-11).

- From the Options menu, select Edit Color (Figure II-40). From the Color Range Mapping menu, choose Constant (Figure II-41). This will reduce the number of colours in the image to two – red and blue. This will make the tumour more distinct, because it will be a completely different colour to the surrounding tissues. Linear would give a continuous range of colours, producing a large variation of colours, as in Figure II-39.

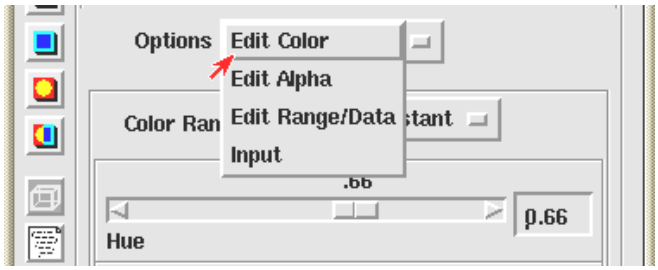

Figure II-40: Editing Colour of Datamap

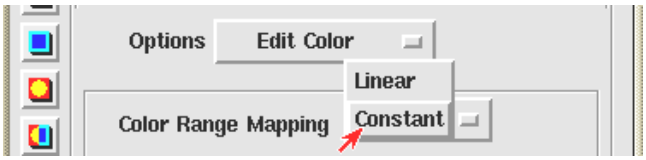

Figure II-41: Editing Colour Range Mapping of Datamap

- From the Options menu, select Edit Range/Data (Appendix A Figure A-12). Click on the Sub-range Values button, and adjust the sliding bars for Range Min and Range Max.

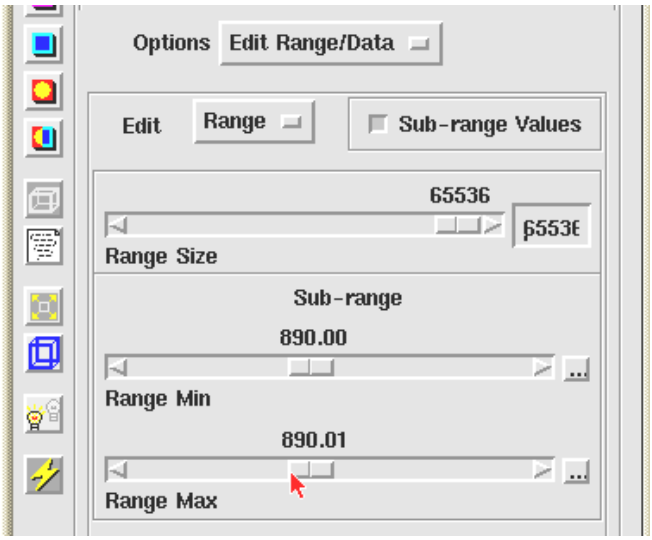

Figure II-42: Editing Range of Datamap

The tumour shows up red on a blue background. Varying minimum and maximum sub-range values alters which tissues are shown. Initially the minimum sliding bar was set at zero, and the maximum bar at 0.01, and both were gradually increased simultaneously until the tumour was visible. Then the maximum bar was decreased, keeping the minimum at one value lower than the maximum bar, until the tumour was visible. The difference between the maximum and

minimum sub-range values was kept at 0.01, in order to pinpoint the exact values at which the tumour was visible. On slice 14, the tumour is distinctly visible between values 890 and 890.01, as shown in Figure II-43.

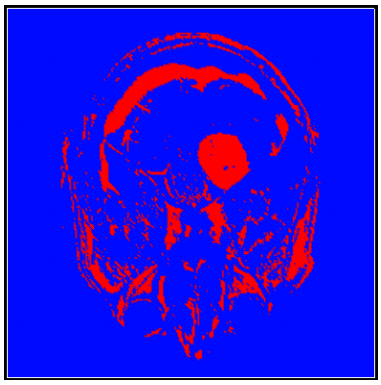

Figure II-43: Slice 14, datamap sub-range 890 to 890.01

Between values 1968 and 1968.01, the tumour is only just visible, as shown in Figure II-44.

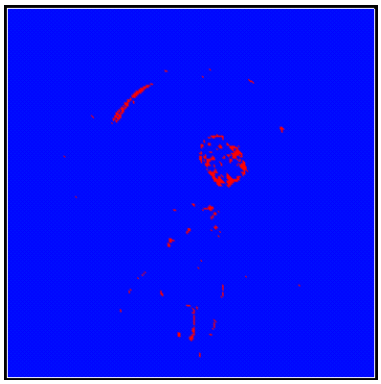

Figure II-44: Slice 14, datamap sub-range 1968 to 1968.01

Between values 1476 and 1476.01, the tumour is clearly visible, as shown in Figure II-45.

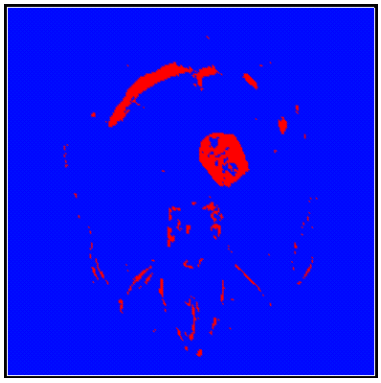

Figure II-45: Slice 14, datamap sub-range 1476 to 1476.01

Figure II-46 shows that the tumour is also clearly visible between values 1751 and 1751.01.

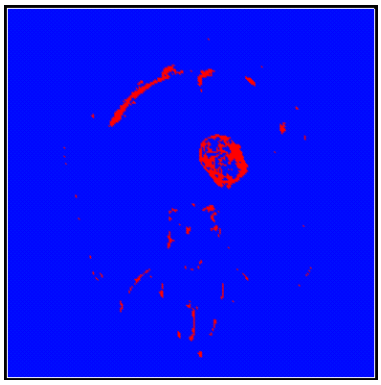

Figure II-46: Slice 14, datamap sub-range 1751 to 1751.01

The tumour is distinctly visible between values 890 and 1968. The optimal values for the t1c data are 1476 – 1751 (the same values as isolevel), showing the tumour in red, and most other tissues in blue. On slice 14, with sub-range values set to minimum 1711.08, maximum 1711.09, gaps are visible inside the tumour, as shown in Figure II-47.

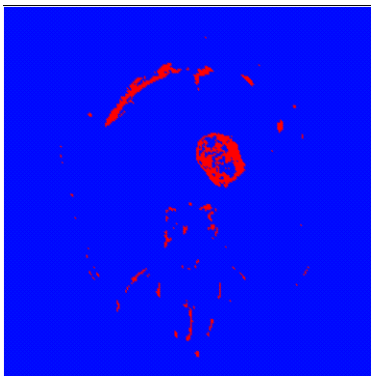

Figure II-47: Slice 14, datamap sub-range 1711.08 to 1711.09

In Figure II-48, slice 11 shows the tumour, and shows less of the other head tissues shown by slice 14.

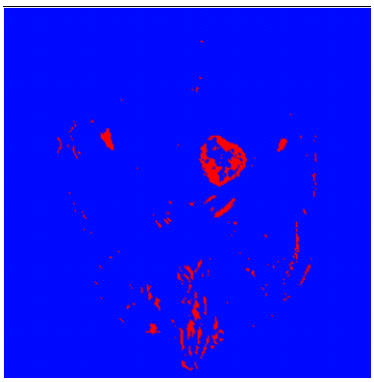

Figure II-48: Slice 11, sub-range 1476.21 to 1476.22

2.3.3 Volume rendering

In order to render the volume of the tumour, another network was created, shown in Figure II-49. This uses the **volume_render** module from the Mappers library (Appendix A Figure A-1), which directly renders a volume.

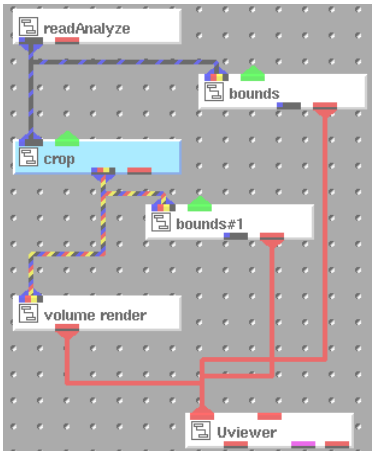

Figure II-49: Network using **volume_render**

The other modules used are:

readAnalyze: reads a header file and creates a uniform field.

Bounds: One **bounds** module was connected from **readAnalyze** to the **Uviewer3D**, to show the extent of the data. Another **bounds** module was connected from the output of **crop** to the **Uviewer3D**, to show the extent of the cropped area.

crop: used to cut out irrelevant detail.

Uviewer3D: used to display the image in 3D.

In order to use **volume_render**, the software renderer must be used:

- Go to the Editors menu on the Dataviewer Pad (Appendix A Figure A-10), and choose View
- From the View menu, choose General (Appendix A Figure A-13).
- Choose the renderer to be Software, instead of OpenGL (Appendix A Figure A-14). This allows a ray-tracing volume-rendering technique to be used [5].

From the Editors menu choose Modules (Appendix A Figure A-10), and choose **readAnalyze**, to select a header file (Appendix A Figure A-15). In the following examples, the header file *t1c.hdr* was used. A solid green cube appeared, filling the bounds of the data. This was not useful, so the parameters of **volume_render** needed to be changed to show the head tissues.

To alter parameters of **volume_render**:

- Go to the Editors menu, and choose Modules (Appendix A Figure A-10).

- Choose the **volume_render** module. The parameters of **volume_render** now appear (Appendix A Figure A-16).
- Choose Surface to be Gouraud Shading[6] (Figure II-50).

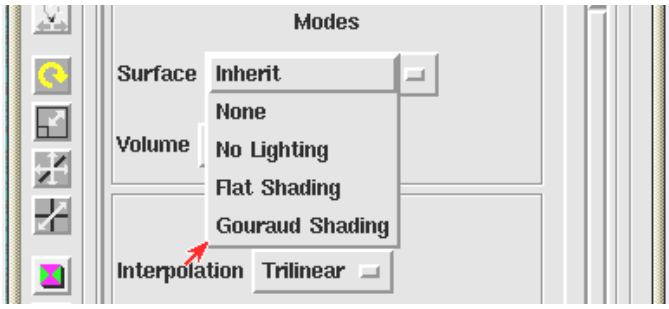

Figure II-50: Selecting Surface

- Change Volume to Ray tracer (Figure II-51). This is the only volume-rendering technique that can be used with the software renderer [5].

Figure II-51: Selecting Volume

- The Range Position was initially set to -0.50, which was outside the data values of the head tissues, which is why a solid green cube was displayed. The Range Position was increased to 496, by moving the sliding bar (Figure II-52).

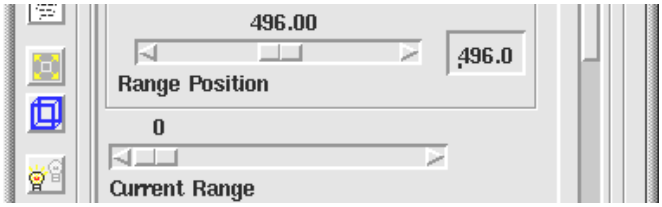

Figure II-52: Altering Range Position

- This displayed most of the head tissues, including the exterior layers of the head, as shown in Figure II-53.

[6] Flat Shading, Inherit or No Lighting will give very similar results, but Gouraud Shading gives the highest quality rendering because it uses both lighting and interpolation techniques. None gives a black screen.

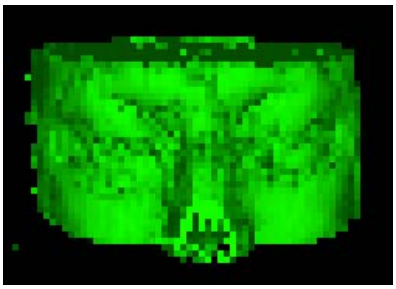

Figure II-53: Range Position 496

In this picture the head has been rotated 90 degrees to show it facing forwards. Switching off the Fat Ray[7] button gives better resolution, as shown in Figure II-54.

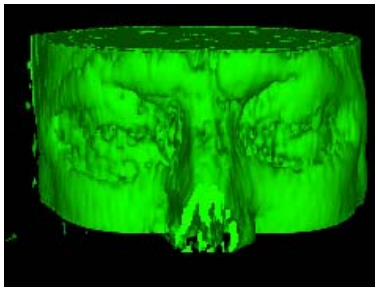

Figure II-54: Range Position 496, fat ray off

Pressing the Reset/Normalize/Center icon on the Dataviewer Pad toolbar (Appendix A Figure A-3), gives a view looking down from the top of the head. Increasing the Range Position to 992 (Figure II-55) shows the internal tissues of the head, including the tumour.

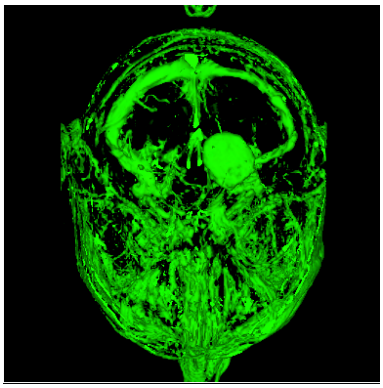

Figure II-55: Range Position 992

Figure II-56 shows Range Position 1489, with the image cropped. As before (Section 2.3.1) the image was cropped in the i-direction to 125-180, and in the j-direction to 125-180, to just show the tumour.

[7] Fat Ray renders each data element across multiple pixels, which renders the image more quickly, but with lower resolution [5].

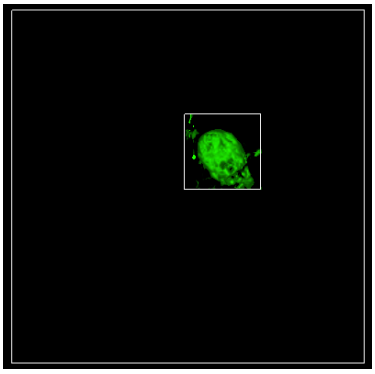

Figure II-56: Range Position 1489

The image was then scaled and rotated, using the icons on the Dataviewer Pad (Appendix A Figure A-3). Figure II-57 shows a close up view of the tumour.

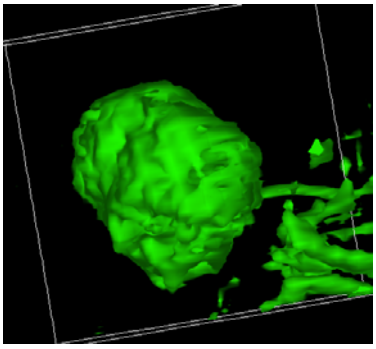

Figure II-57: Range Position 1709

The image was cropped in the k-direction to view slices 10 and 11, because the slices had previously been found to show the tumour well (Figure II-58). The tumour looks hollow inside, because the tissues inside have different data values, which do not show up at this range position.

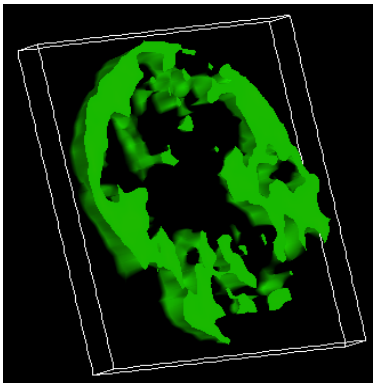

Figure II-58: Range Position 1700, slices 10 and 11

In Figure II-59 the image has been cropped in the k-direction to show slices 7 to 16, which shows the whole volume of the tumour.

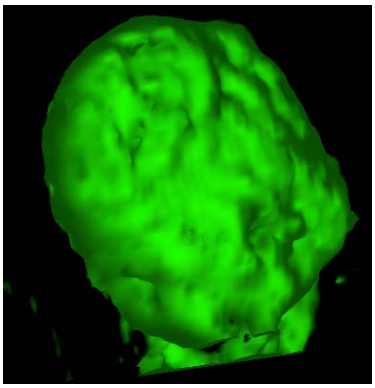

Figure II-59: Range Position 1500, slices 7 to 16

Figure II-60 shows half of the tumour, in slices 8 to 13. This shows the inner surfaces of the tumour.

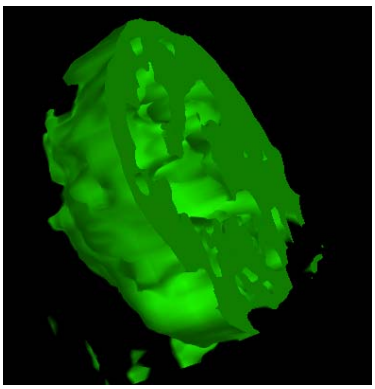

Figure II-60: Range Position 1500, slices 8 to 13

At range position 1985, the tumour shows up less clearly, as shown in Figure II-61:

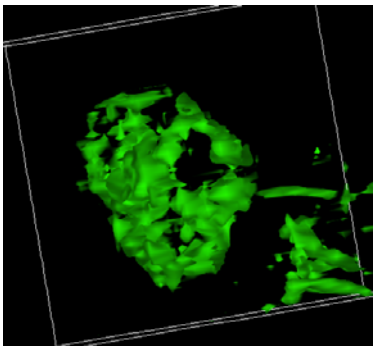

Figure II-61: Range Position 1985

The **volume_render** module helps to show the shape of an object. Figures II-62 and II-63 show that the tumour is bigger in slices 10 -13 than in slices 14 -16.

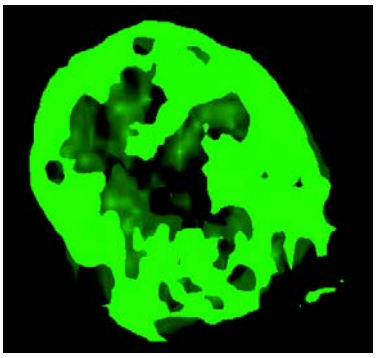

Figure II-62: Range Position 1500, slices 10 to 13

The cross-sectional area shown in Figure II-52, for slices 10 to 13, is bigger than in Figure II-63, for slices 14 to 16. This is because the centre of the tumour is located between slices 10 and 13, so this is where the tumour is largest.

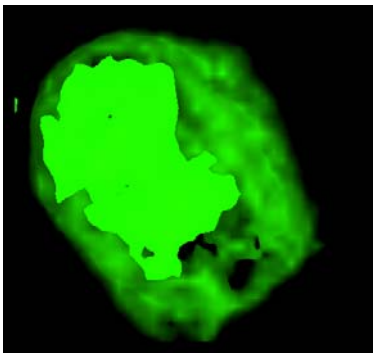

Figure II-63: Range Position 1500, slices 14 to 16

The transparency of the images can be altered, by changing the alpha values. The **volume_render** module has the parameters Alpha Range Model, Minimum Alpha and Maximum Alpha (Figure II-64).

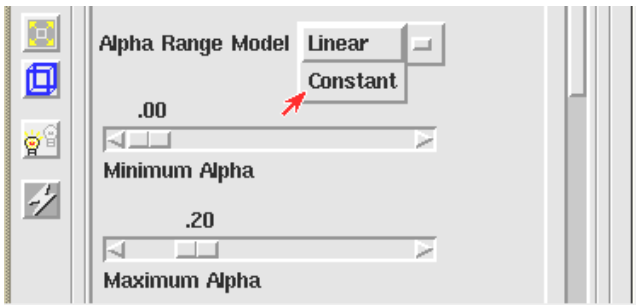

Figure II-64: Altering Alpha Values

The Alpha Range Model can be either Constant or Linear. Both give similar results. The Minimum Alpha and Maximum Alpha were set to 0 by default, which produced the images in Figures II-53 to II-63. Increasing Maximum Alpha to 0.5 reduced the transparency of the tissues surrounding the tumour. The effect is shown in Figure II-65, which shows slices 10 to 13, with green tissues visible around the tumour.

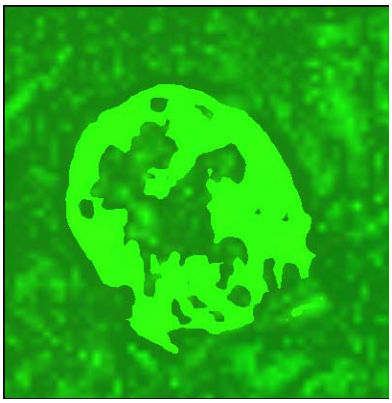

Figure II-65: Maximum Alpha value 0.5

Increasing Maximum Alpha to 1 made all of the tissues in the image completely opaque. This produced a solid green volume, as shown in Figure II-66.

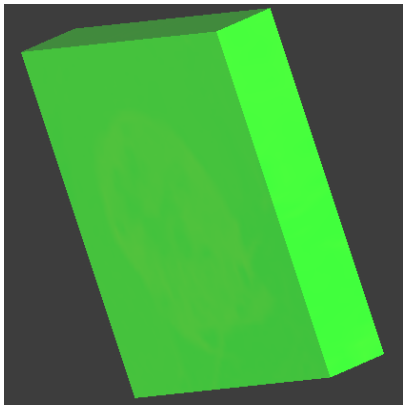

Figure II-66: Maximum Alpha value 1

The clearest images were produced by the default Maximum Alpha value of 0, which had the highest transparency.

2.3.3.1 Volume Rendering Using **Read_Field**

The **readAnalyze** module was used in volume rendering because it shows the true volume of the tumour (Section 2.3). Figure II-68 shows what would happen if **Read_Field** had been used instead, as in the network shown in Figure II-67.

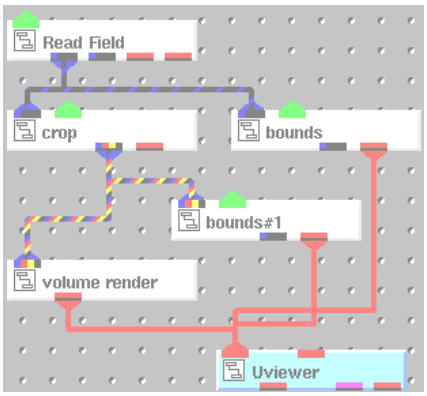

Figure II-67: Network using **Read_Field** and **volume_render**

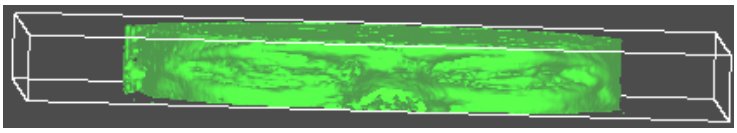

Figure II-68: Image produced using **Read_Field** and **volume_render**

In Figure II-68, the head looks squashed, because its true volume cannot be shown using **Read_Field**. The height of the head should be three times larger, as shown Figure II-54, which was achieved using **readAnalyze**.

2.4 Conclusion

This chapter described an example application of visualization of medical data, using AVS/Express as the visualization tool. The data represented images of a human head, created using an MRI scanner to scan part of the head in 25 slices. There were five data files to process, in Analyze form, which had binary data in a file, and a corresponding header file.

It was necessary to import the data into AVS/Express, in order to visualize it. A field file was made for each data file, describing the structure and type of data, and the **Read_Field** module was used to read the field files into AVS/Express (Section 2.2). A problem with using **Read_Field** was that it could not show the true thickness of the slices. Each slice was 3mm thick, and required three pixels per data element, but **Read_Field** used only one pixel. A solution was to use **readAnalyze** instead (Section 2.3.1). The **readAnalyze** module reads the header files, which contain extra information about the data. The module **readAnalyze** gave better results because it showed the true volume of the head slices.

Slices through the head could be viewed using **orthoslice** (Section 2.3). Looking through the slices in all five data files (Section 2) confirmed that the tumour is present in slices 7 to 16. The tumour appeared largest in slices 10 to 14.

The **isosurface** module showed all tissues with the same data values (Section 2.3.1). An isosurface is a surface of constant value, and gives a graphic depiction of the locations of a particular data value in a 3D field. Different tissue types such as bone, and muscle, show up at different isolevels. In order to find the isolevel of the tumour, various isolevels were tried on all of the data. Using the **isosurface** module, the isolevel was adjusted until the tumour was visible. Any other tissues showing at the same isolevel had the same data values, and it was concluded that the tumour had the same tissue type as these tissues.

The **crop** module was used to cut out irrelevant detail (Section 2.3.1). The sliding bars for the three dimensions were adjusted until only the tumour was visible (Appendix A Figure A-7). The tumour was found to be located between pixel values 125 and 180 in the first and second dimensions, and between 7 and 16 in the third dimension. The **bounds** module was useful for showing the extent of the volume of the data, and could also be connected to the output of crop, to show the extent of the cropped region. By altering the datamap, the colour of the tumour could be changed to red, and other tissues to blue, which made the tumour stand out more, and showed that the tumour was of a different tissue type to the surrounding tissues (Section 2.3.2).

Volume rendering produces images of three-dimensional volumetric data [5]. Volume rendering was useful for showing the true volume of the tumour (Section 2.3.3). The **volume_render** module directly renders a volume. The image could be cropped, scaled and rotated to obtain different views of the tumour. Looking through cross-sectional slices showed that the tumour is bigger in slices 10 to 13 than in 14 to 16. This is because the centre of the tumour is between slices 10 and 13, so this is where the tumour is largest.

The volume-rendering network used in Section 2.3.3, shown below in Figure II-69, has the most useful features of all the networks used in this chapter. The **readAnalyze** and **volume_render** modules showed the true volume of the tumour. The range of data values rendered by **volume_render** could be adjusted to show the tumour distinctly from surrounding tissues, which were of different tissue types. The **crop** module allowed uninteresting parts of the image to be cut out. The use of two **bounds** modules showed the extent of the cropped section, as well as the full extent of the data.

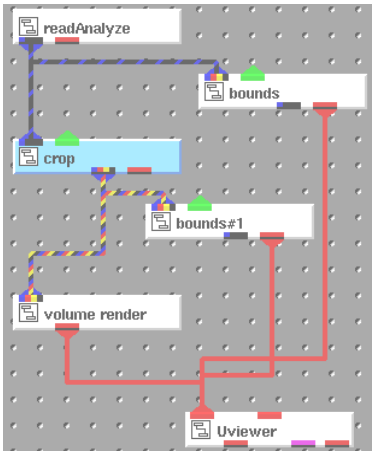

Figure II-69: Volume-rendering Network

3 Visualizing Meteorological Data

This chapter illustrates an example application using meteorological data. This describes how to get from the raw data to the rendered image.

The data is global ocean data, scanned in 20 slices. Each slice has a length and width of 288 by 143 pixels. The data has five components, labelled *u*, *v*, *w*, *temp*, and *mag*. The *u*, *v* and *w* components give the three directions of the velocity of water flow in the oceans. The magnitude of the velocity is *mag*, and the temperature is *temp*. The data contains null values, which represent the land.

An interesting feature of the data is the Gulf Stream. This is a warm current of the North Atlantic Ocean, flowing in a generally northeastern direction from the Straits of Florida to the Grand Banks, east and south of Newfoundland. The term is often extended to include the North Atlantic Drift, which flows from the Grand Banks to the shores of Western Europe, Scandinavia, and the islands of the Arctic Ocean. The Gulf Stream is of great meteorological importance because of its moderating effects on the climate of Western Europe [7].

The data is in two files, *oceanB1994.dat*, and *oceanB.xyz*.

3.1 Reading Data

Data can be imported using a field reader. AVS/Express has a module called **Read_Field**, in the DataIO library, (Appendix A Figure A-1), which is the natural choice of module for reading in binary image data. **Read_Field** reads in an AVS field file and converts it to an AVS/Express field. A field file contains metadata with specific information describing the structure and type of data, so that AVS/Express can understand it[8].

The values to put in the field file can be gained from a description of the data. There are 20 slices, and each slice has a size of 288 by 143, so the sizes of the dimensions of the data are 288 by 143 by 20, given by *dim1* = 288, *dim2* = 143 and *dim3* = 20. *ndim* = 3, because there are 3 dimensions in computational space, and *nspace* = 3, because there are 3 dimensions in co-ordinate space.

veclen = 5, because there are 5 data components per data element. These are labelled *u*, *v*, *w*, *temp*, and *mag*. The type of *data* is *float* (a floating-point number). The mapping method, given in *field*, is *rectilinear*. This means that each dimension in the data has an explicit co-ordinate mapping, the spacing along the co-ordinate axes can be non-uniform, and the dimensions of the computational and co-ordinate space are the same.

The five variables *u*, *v*, *w*, *temp*, and *mag*, are all found in the same file, *oceanB1994.dat*, which has a binary *filetype*. The data components for the five variables are interleaved in the following format:

```
u v w temp mag u v w temp mag u v w temp mag u v w temp mag …
```

All the first data components, for all the variables, are followed by all of the second data components. Each variable has a *stride* of 5, because there is a step of 5 floats between data components for a variable.

[8] A description of the format of a field file can be found at www.iavsc.org/training [2]

The *skip* value tells the field reader where to find the first data component for a variable. The *skip* value is given in bytes.

The first data component for variable 1 (u) is the first item in the file, so there are no bytes to step over, so its *skip* is 0.

The first data component for variable 2 (v) is located after the first data component for variable 1. Therefore a *skip* of 1 float is needed to reach the first data component for variable 2. 1 float = 4 bytes, so the *skip* is 4 for variable 2.

The *skip* is different for each variable, because the first data component for a variable (n) is preceded by (n – 1) other variables in the file. Each data component is of type float, which is 4 bytes, so ((n - 1) x 4) bytes must be jumped over. Therefore variable 3 has a *skip* of 8, variable 4 has a *skip* of 12, and variable 5 has a *skip* of 16.

Each dimension in the data has an explicit co-ordinate mapping[9]. The co-ordinates are found in the file *oceanB.xyz*, which has a binary *filetype*. All of the data for co-ordinate 1 is followed by all of the data for co-ordinate 2, which is followed by all of the data for co-ordinate 3.

For co-ordinate 1, there are no lines to *skip* before the first data element is found, so *skip* = 0. The *stride* is 1 because the data components for co-ordinate 1 are sequential in the file, so a step of 1 (1 float) is needed to reach the next data component for co-ordinate 1.

The first data element for co-ordinate 2 is found after all the co-ordinate 1 data. There are 288 data components for dimension 1, of type float. There are (288 x 4) bytes for co-ordinate 1, so there are 1152 bytes before the data for co-ordinate 2, so *skip*=1152. The *stride* equals 1 because the data components for co-ordinate 2 are sequential in the file, so a step of 1 (1 float) is needed to reach the next data component for co-ordinate 2.

The first data element for co-ordinate 3 is found after all the co-ordinate 2 data. There are 143 data components for dimension 2, of type float. There are (143 x 4) bytes for co-ordinate 2, and 1152 bytes before the data for co-ordinate 2, so the co-ordinate 3 data starts after (143 x 4 + 1152) bytes, so *skip*=1724. The *stride* equals 1 because the data components for co-ordinate 3 are sequential in the file, so a step of 1 (1 float) is needed to reach the next data component for co-ordinate 3.

The field file is shown below.

```
# AVS field file
#
ndim = 3
dim1 = 288
dim2 = 143
dim3 = 20
nspace = 3
veclen = 5
data = float
field = rectilinear
label = u v w temp mag
variable 1 file=./oceanB1994.dat filetype=binary skip=0 stride=5
variable 2 file=./oceanB1994.dat filetype=binary skip=4 stride=5
variable 3 file=./oceanB1994.dat filetype=binary skip=8 stride=5
variable 4 file=./oceanB1994.dat filetype=binary skip=12 stride=5
```

[9] An explicit co-ordinate mapping is required because the data is not regular. The co-ordinates in the third dimension are not evenly spaced.

```
variable 5 file=./oceanB1994.dat filetype=binary skip=16 stride=5
coord 1 file=./oceanB.xyz filetype=binary skip=0 stride=1
coord 2 file=./oceanB.xyz filetype=binary skip=1152 stride=1
coord 3 file=./oceanB.xyz filetype=binary skip=1724 stride=1
```

The field file is created in a text editor such as Nedit or Emacs, and saved with a *.fld* extension, e.g. *oceanB1994.fld*.

3.2 Visualizing Data

AVS/Express has libraries of predefined modules, which can be joined together to form a network. A network was made to process the data, using the Network Editor (Appendix A Figure A-1).[10]

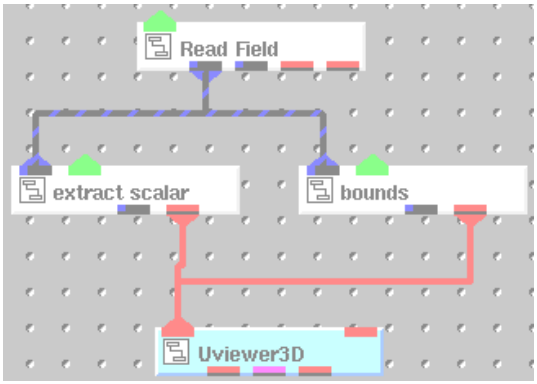

Figure III-1: Network using **extract_scalar**

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, to create an image.

extract_scalar: extracts a single scalar data element from a vector component of a field. For example, selecting the fourth vector element of the field (u, v, w, temp, mag) returns the field (temp). This was used to show one data component individually.

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

The network in Figure III-1 was used to produce images from the data files (Section 3.1).

3.3 Altering Images

The network is used to produce an image, which can be manipulated to the desired format. Simple transformations such as scale, translate and rotate can be performed using the icons on the toolbar of the Dataviewer Pad window (Appendix

[10] For instructions on how to find modules, refer to Appendix A Figures A-4 to A-5.

A Figure A-3). The parameters of each module can be altered. From the Editors menu, select Modules, then the module name, and its parameters will be displayed, which can be changed by using the sliding bars to change values, or entering digits.

The data component parameter of **extract_scalar** was initially set to temp (Appendix A Figure A-17). This produced the image shown in Figure III-2. This shows the top slice of data, with the land in blue, and seas and oceans in red.

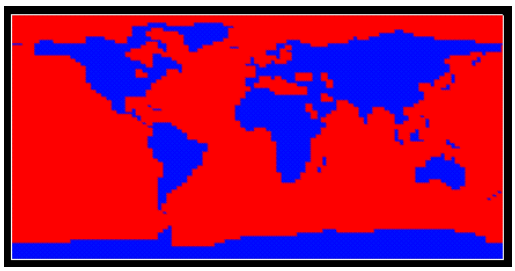

Figure III-2: **extract_scalar** temp

This was rotated to give a side view of all 20 slices (Figure III-3). This shows the depth of the seabed.

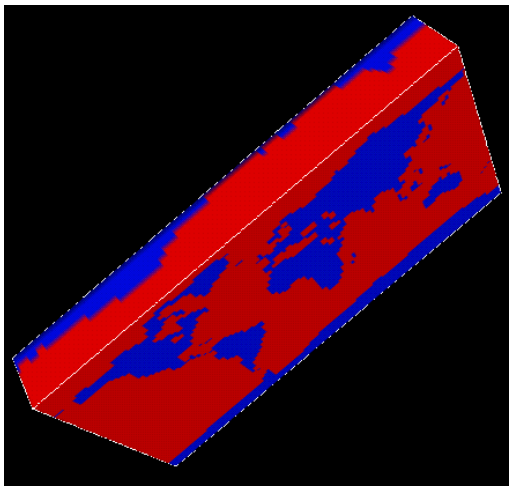

Figure III-3: **extract_scalar** temp, side view

This was rotated to show the bottom of the last slice (Figure III-4). This shows some areas of sea, in red, but is mostly blue land, which is the seabed.

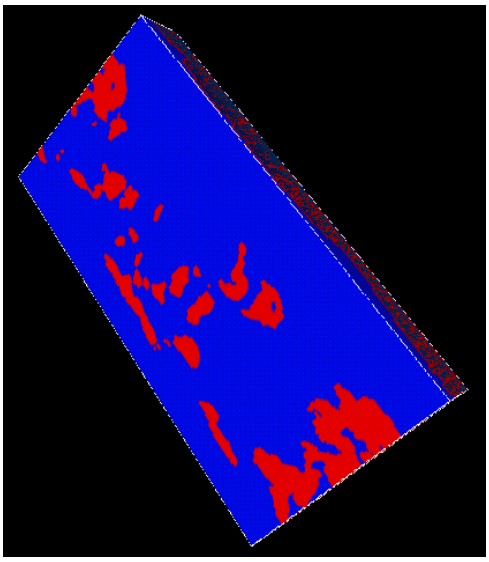

Figure III-4: **extract_scalar** temp, back view

3.3.1 Changing the Datamap

The images shown so far have been coloured according to default values, defined by the datamap. All of the land has been coloured blue, and all of the sea has been coloured red. The range of colours in the images can be altered, by changing the datamap. This would be useful for showing more information, because different data values would have different colours, and the images would represent the data in more detail. The sea is currently all one colour, so the images give no information about the different data values in different regions of the sea. Changing the datamap could show different colours in regions of the sea, where there is variation in data values.

- Click on the Select Object… button at the bottom of the Dataviewer Pad window (Appendix A Figure A-3).
- From the Object Selector window, which now appears (Appendix A Figure A-21), select **extract_scalar** from the list of items, and click on OK.
- Select Editors from the Dataviewer Pad window, and select Datamap (Appendix A Figure A-10). The Datamap Editor will now appear in the Dataviewer Pad window (Appendix A Figure A-11).
- From the Options menu, select Edit Color (Figure III-5). From the Color Range Mapping menu, choose Linear (Figure III-6). This gives a continuous range of colours, so a large variation of colours will be shown. Constant would produce only two colours – red and blue.

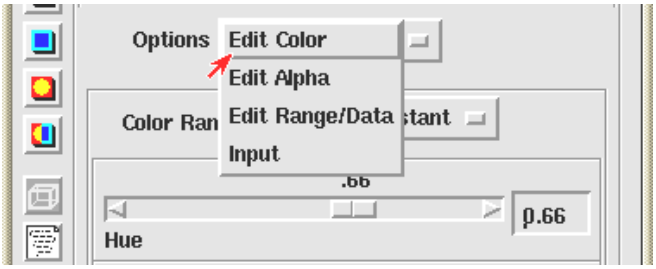

Figure III-5: Editing Colour of Datamap

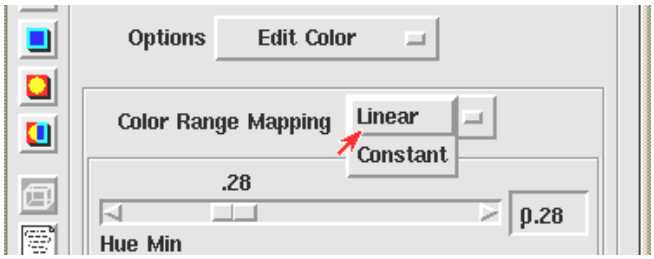

Figure III-6: Editing Colour Range Mapping of Datamap

- From the Options menu, select Edit Range/Data (Appendix A Figure A-12). Click on the Sub-range Values button, and adjust the sliding bars for Range Min and Range Max. Changing the sub-range values alters the range of data values to which the colour range is mapped. The range of data values can be adjusted to pick out the features the user wants to see.

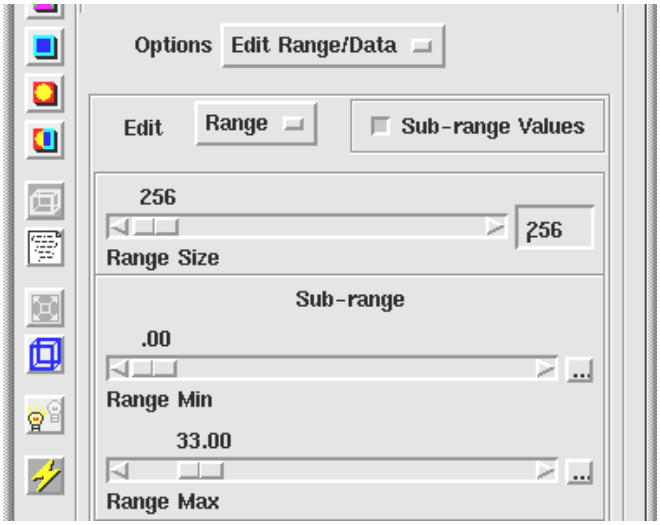

Figure III-7: Editing Range of Datamap

The value of Range Min was set to 0, and Range Max was altered to change the range of colours shown. Range Max value 10 produced the image shown in Figure III-8, for the temp data component. The sea now shows different colours at different data values. The points with the highest data values are coloured red, and as the data values decrease, the colours change from red to orange, yellow, green, cyan and blue. The points with the lowest data values are coloured blue. In Figure III-8 the seas are coloured red, with yellow, green and blue areas towards the top and bottom of the image. This shows that the temperature is lower towards the north and south poles.

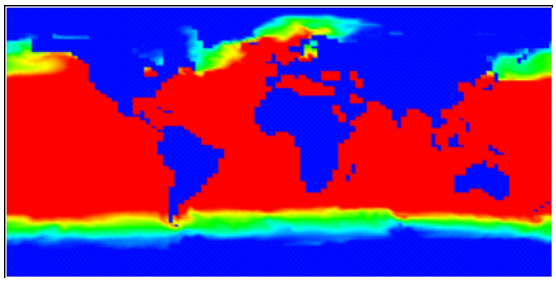

Figure III-8: Range Max 10

Figure III-9 shows the effect of Range Max value 50. This shows the sea in green towards the equator, and blue towards the North and South poles.

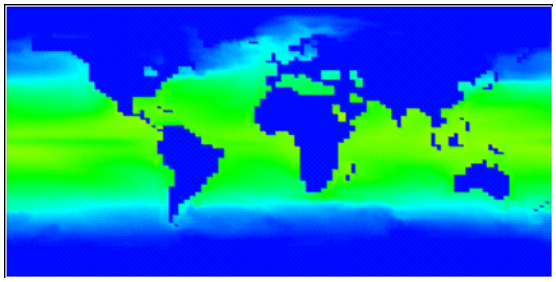

Figure III-9: Range Max 50

- Range Max value 33 produced the image in Figure III-10 for the temp data component. This Range Max value was the best, because it showed the largest range of colours for the different data values in the sea. Figure III-10 shows that the temperature is higher towards the equator.

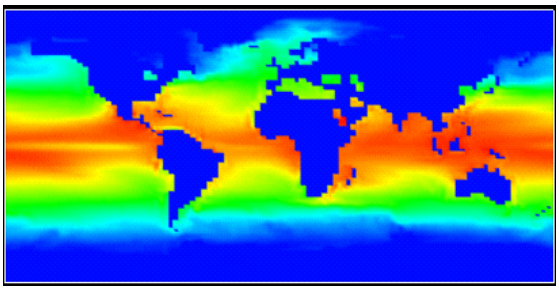

Figure III-10: **extract_scalar** temp-component

- The data component parameter of **extract_scalar** was changed to u, then v, w, and mag (Appendix A Figure A-17). This produced the following images for the different data components (Figures III-11 to III-14). These images show the top slice of data.

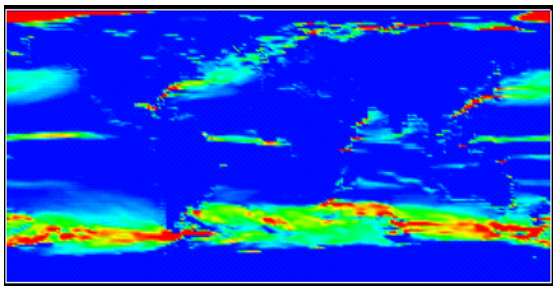

Figure III-11: **extract_scalar** u-component

Figure III-11 shows the Gulf Stream, across the North Atlantic, in red and green, while the surrounding ocean is blue. This shows that the u-component of velocity is higher in the Gulf Stream than in surrounding waters. This is because the Gulf Stream is a current, which flows faster than the surrounding waters, so its velocity in the u-direction is higher.

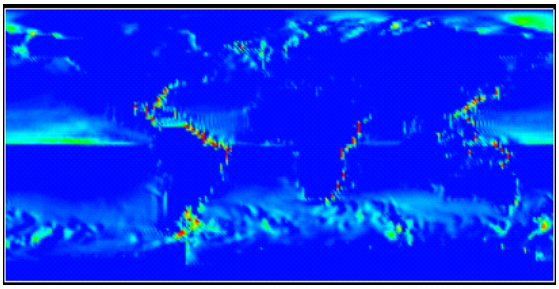

Figure III-12: **extract_scalar** v-component

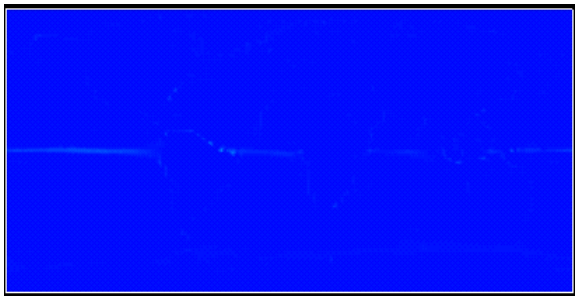

Figure III-13: **extract_scalar** w-component

In Figure III-13 the map is all the same shade of blue, which suggests that the w-component is very similar all over the world.

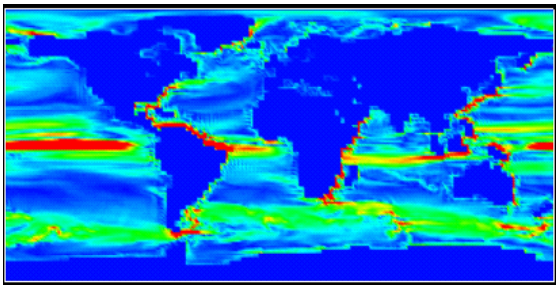

Figure III-14: **extract_scalar** mag-component

Figure III-14 shows that the magnitude of velocity is higher in the Gulf Stream than in the surrounding ocean.

3.3.2 Using **orthoslice**

Another network was made using **orthoslice** (Figure III-15), in order to view slices individually.

The images shown in Figures III-8 to III-14 had regions of both land and sea coloured in the same shade of blue. To avoid this, the **set_null** module was used, to set the data values of the land to null. When the data was created, an impossible data value was used to indicate a null value. The **set_null** module allows that data value to be defined as a null value. The regions of land have null data values, and are not visualized. This leaves the land colourless, so that regions of land will not be confused with regions of sea.

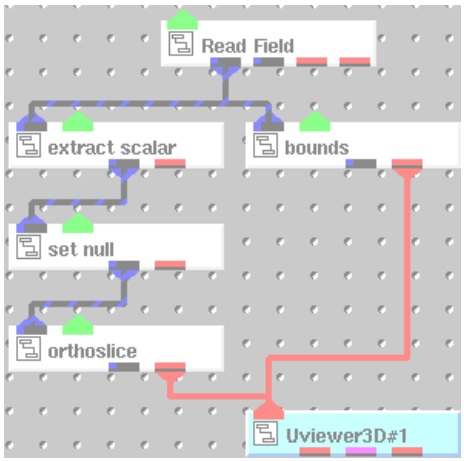

Figure III-15: Network using **orthoslice**

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, to create an image.

extract_scalar: extracts a single scalar data element from a vector component of a field. For example, selecting the fourth vector element of the field (u, v, w, temp, mag) returns the field (temp). This was used to show one data component individually.

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

set_null: alters a field by defining a specified value as a null value and setting the flag that indicates the presence of null data to 1 ("has null data"). This was used to set the data components of the land to null, because the land has no velocity values, and should not be visualized.

orthoslice: produces a slice of a structured field perpendicular to a selected co-ordinate axis. This allows each of the 25 slices to be viewed individually[11].

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

The data component parameter of **extract_scalar** was set to temp (Appendix A Figure A-17). This produced the image in Figure III-16, for the top slice of data.

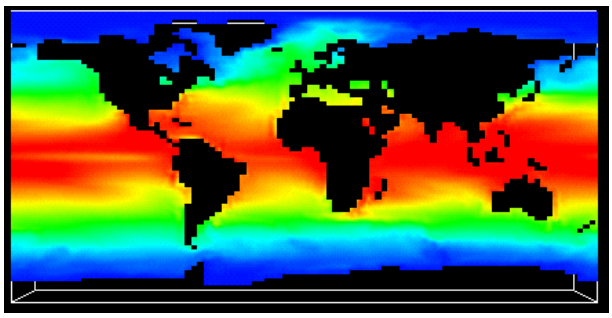

Figure III-16: **orthoslice** axis 2 plane 19

In Figure III-16 the sea is redder towards the equator. This is because the sea nearest to the equator receives the most heat from the sun, so the temperature increases towards the equator. The land is now colourless, and is transparent so the black background shows through, because **set_null** was used to set the data values of the land to null.

The **orthoslice** module takes slices through a field. The position of the slice can be changed to view different cross-sections through the field. From the Editors menu, select Modules, then select **orthoslice**, and alter the sliding bars (Appendix A Figures A-3 and A-6). The axis is the coordinate axis the slice is perpendicular to. 0 is the X-axis, 1 is the Y-axis and 2 is the Z-axis. Choose axis 2 for a view looking down from above the world (showing the Z-plane). The plane was altered to change the slice shown. Figures III-16 to III-25 show different **orthoslice** planes.

[11] The **Orthogonal_slicer** or **slice_plane** module could have been used instead of **orthoslice**. **Orthogonal_slicer** has an extra output, which can be connected to the **Uviewer3D**. This shows the bounds of the slice, and can be toggled on and off. Showing the slice boundaries could be useful for showing the original size of the slice, if the image were being cropped, but in this case is not necessary. The **slice_plane** module allows the slice plane to be in any position, not restricted to the co-ordinate axes.

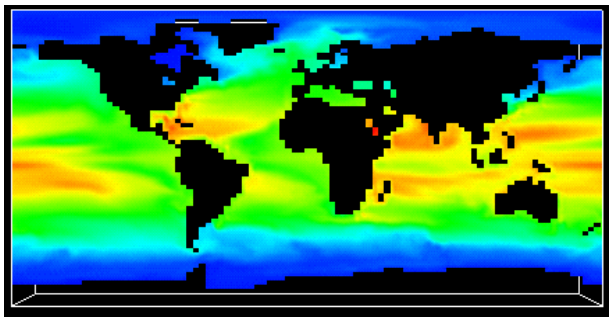

Figure III-17: **orthoslice** axis 2 plane 12

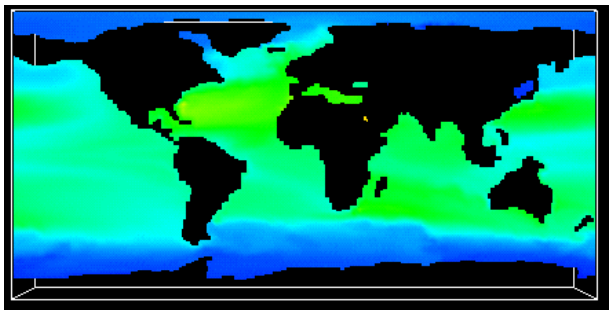

Figure III-18: **orthoslice** axis 2 plane 9

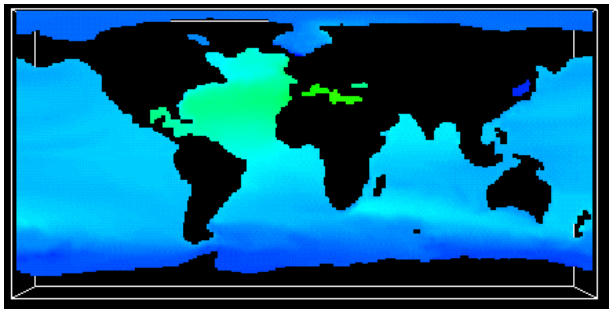

Figure III-19: **orthoslice** axis 2 plane 7

Figure III-19 shows that the temperature is higher in the North Atlantic than in other oceans. This could be due to the Gulf Stream, which is a warm current in the North Atlantic.

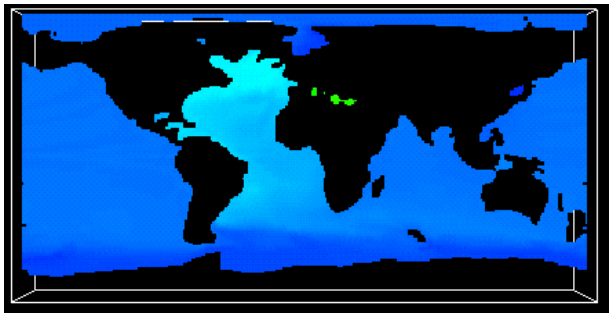

Figure III-20: **orthoslice** axis 2 plane 5

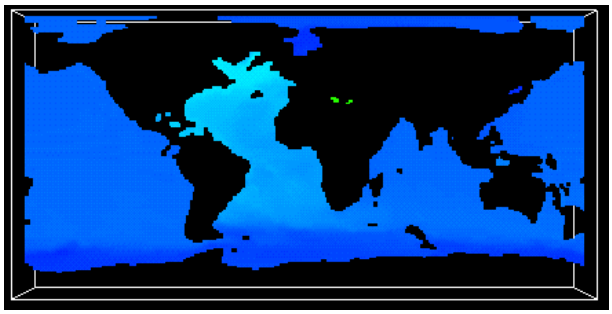

Figure III-21: **orthoslice** axis 2 plane 4

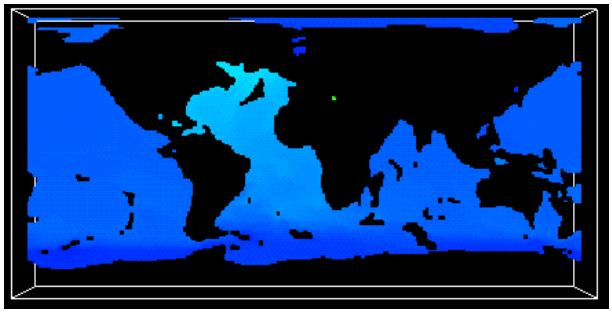

Figure III-22: **orthoslice** axis 2 plane 3

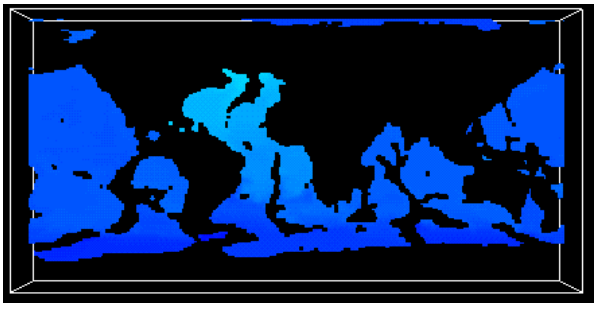

Figure III-23: **orthoslice** axis 2 plane 2

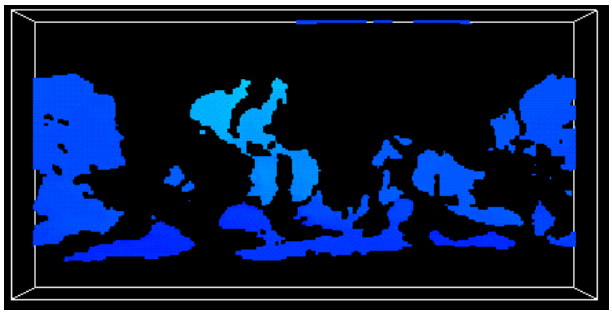

Figure III-24: **orthoslice** axis 2 plane 1

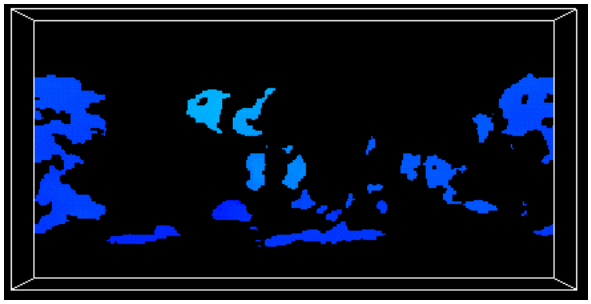

Figure III-25: **orthoslice** axis 2 plane 0

Figure III-25 shows the bottom slice, and the North Atlantic is a paler shade of blue than the other waters. This shows that the Atlantic is the warmest ocean, through all the depths.

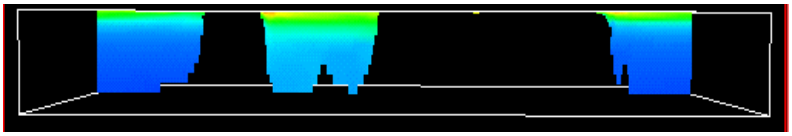

Figure III-26: **orthoslice** axis 1 plane 95, rotated

In Figure III-26 the **orthoslice** axis is set to 1. The image is rotated to the side, so the view is from the South Pole, through all 20 slices.

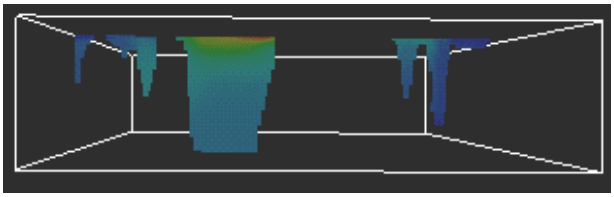

Figure III-27: **orthoslice** axis 0 plane 96

In Figure III-27 the **orthoslice** axis is set to 0. This gives a view from the west, and shows the depth of the ocean, and the temperature through the depths.

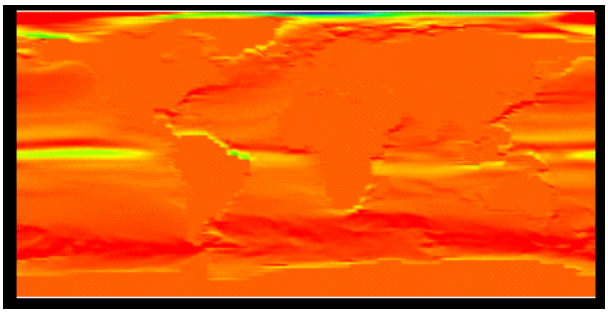

Figure III-28: **orthoslice** axis 2 plane 19 u-component

Figure III-28 shows the u-component of the data. It appears to be higher along the coasts of continents than in the rest of the oceans.

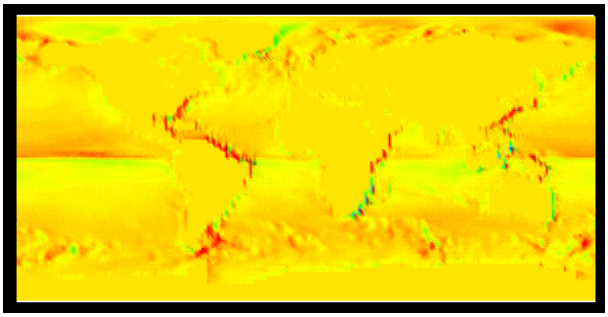

Figure III-29: **orthoslice** axis 2 plane 19 v-component

In Figure III-29, the v-component appears to be higher along the coasts than in the oceans.

3.3.3 Particle advection techniques

Flow fields can be examined by injecting a foreign substance into the flow and observing the effects the flow field has on this substance. Advection is the local change of this foreign substance due to the flow field [5].

Releasing and advecting weightless particles in a flow field can simulate a variety of effects. The particles can be represented as points. A particle's current position can be connected to the position it would be advected to, for a constant flow. Repeating for several time-steps and accumulating the line segments gives a streamline.

AVS/Express has a module called **streamlines,** which generates a set of lines showing the progress of weightless particles, and an **advector** module which simulates the release of weightless particles into a flow field.

3.3.3.1 Streamlines

A network was made using the **streamlines** module (Figure III-30).

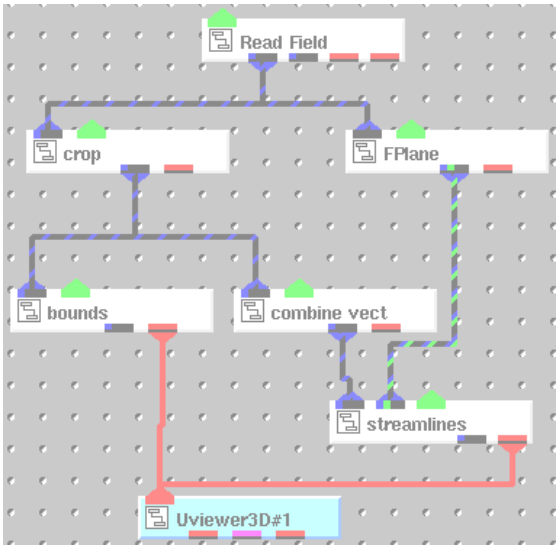

Figure III-30: Network using **streamlines**

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, to create an image.

- **crop**: extracts a subset of a uniform field. This was used to cut out uninteresting parts of the image.

FPlane: creates a 3D plane of variable size. This plane can be transformed in three dimensions, and is useful as a sampling or slicing input to macros such as **streamlines**. **FPlane** was used as a probe, which creates a sample of points to use as starting positions for the streamlines. The **streamlines** module will advance these points through space, creating a set of lines showing the progress of particles through the field.

combine_vect: takes all selected scalar data components and produces a single output vector component.

streamlines: generates a set of lines showing the progress of weightless particles through a vector field.

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

The network was used to produce the following image.

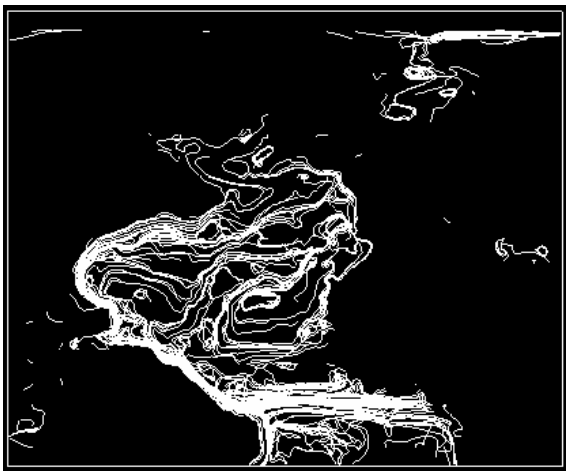

Figure III-31: streamlines image

The streamlines appear throughout the ocean, and are most dense near the east coast of America.

3.3.3.2 Using **glyph**

The **glyph** module places a geometrical object, such as an arrow, at each node of a field, allowing node locations to be easily identified. The geometrical objects (glyphs) can be sized and coloured according to the data at each node. This could be used to show the magnitude and direction of the current velocity at different points in the ocean. A network was made, using **glyph** and **Arrow1**, to draw arrows at points in the field.

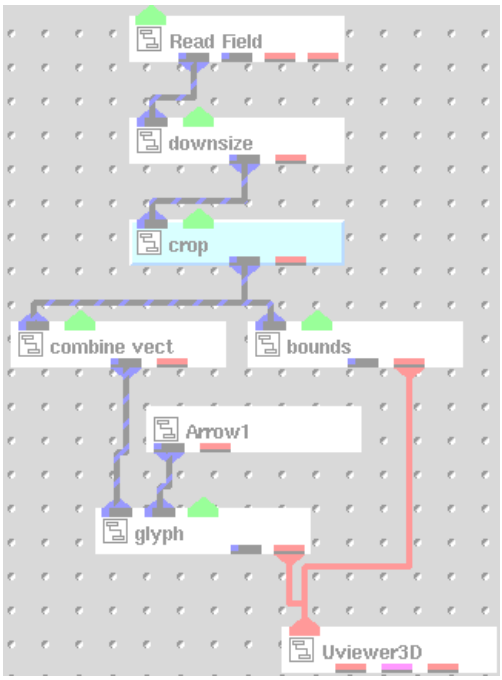

Figure III-32: Network using **glyph**

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, to create an image.

downsize: resamples a field to reduce its size. This was used to reduce the number of arrows created, so that individual arrows would be more clearly visible.

- **crop**: extracts a subset of a uniform field. This was used to cut out uninteresting parts of the image.

combine_vect: takes all selected scalar data components and produces a single output vector component.

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

Arrow1: creates a wireframe arrow-shaped mesh, describing the geometry of the glyph.[12]

glyph: produces geometric objects at points in an input field. This was used with **Arrow1** to produce arrows, sized and coloured according to the magnitude of the data at each point.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

This network produced the following image (Figure III-33). The **downsize** module was used to reduce the size of the sampled field. From the Editors menu, Modules was selected, then **downsize** was selected, and the sliding bars were altered (Appendix A Figure A-18). The I downsize factor alters the sample size in the first dimension, and the J downsize factor alters the sample size in the second dimension. Pressing the Integer Sliders button restricts the factors to integer values. The I downsize factor and J downsize factor were set to 2, to halve the size of the sampled field in both dimensions. The parameters of **combine_vect** were set to combine components u, v and w into a vector. A veclen of 3 was chosen, because the vector would have a length of 3 components, and the buttons for u, v and w were selected (Appendix A Figure A-19).

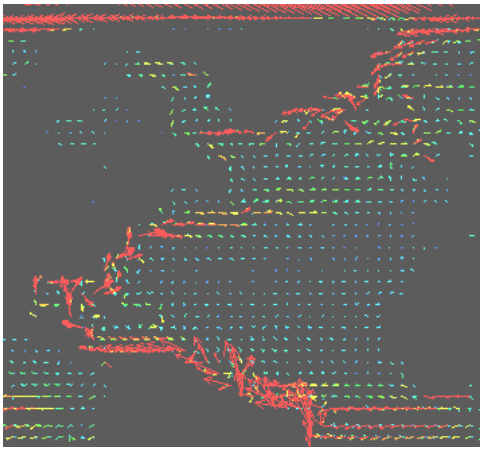

Figure III-33: image using **glyph** and **Arrow1**

The datamap was changed to make the arrows a different range of colours (Section 3.3.1). There are many large red arrows by the east coasts of North and South America, showing that the velocity is highest near the coasts.

[12] Arrow2, Arrow3 or Arrow4 could have been used instead of Arrow1. Each draws a different style of arrow. Arrow2 creates a solid arrow-shaped mesh, Arrow3 creates a 3D wireframe arrow-shaped mesh, and Arrow4 creates a 3D arrow-shaped mesh with a solid 3D head.

3.3.3.3 Advection

A network was made using the **advector** module, which shows progress of particles through a velocity field (Figure III-34).

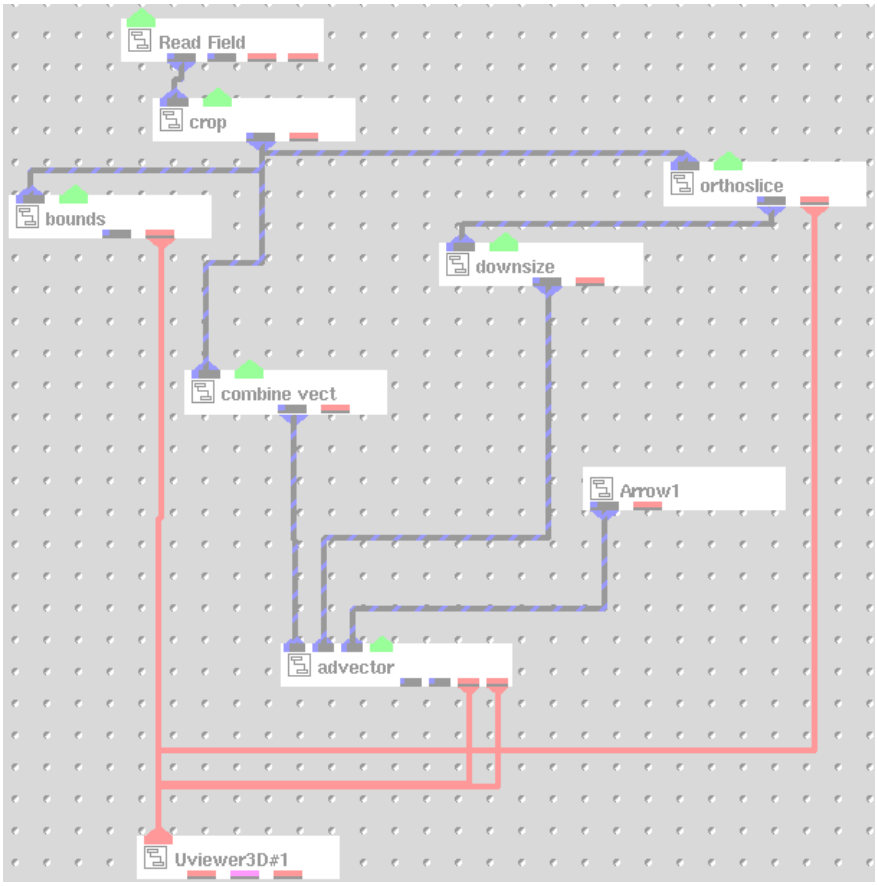

Figure III-34: Network using **advector**

Read_Field: reads a field file and converts it to an AVS/Express field. This was used to read information from the field file, to create an image.

- **crop**: extracts a subset of a uniform field. This was used to cut out parts of the image, to show only the interesting part of the data.

combine_vect: takes all selected scalar data components and produces a single output vector component.

bounds: generates a bounding box of a 3D field. This is useful for showing the extent of an object.

orthoslice: produces a slice of a structured field perpendicular to a selected co-ordinate axis. This was used as an input to **advector**.[13]

[13] Fplane was not suitable as an input to advector, because it did not pass the correct co-ordinate data to advector, so it did not allow arrows to be produced to represent the particles.

downsize: resamples a field to reduce its size.

advector: releases weightless particles into a velocity field.

Arrow1: creates a wireframe arrow-shaped mesh.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

The following image was produced.

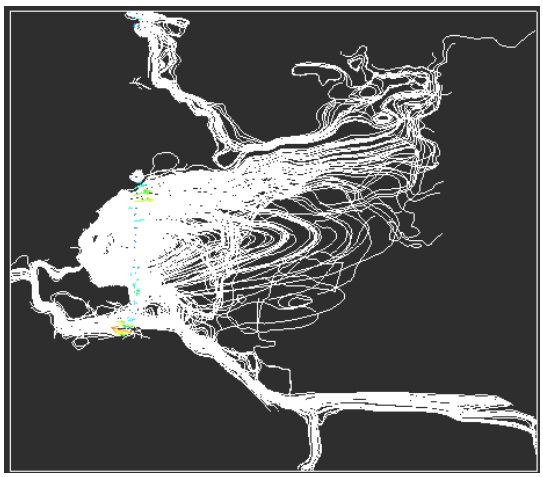

Figure III-35: **downsize** 1

Figure III-35 shows that **advector** can produce streamlines. Here the streamlines were too dense, so the downsize factor was increased from 1 to 4 in the I and J dimensions (Section 3.3.3.2). This produced Figure III-36.

Figure III-36: **downsize** 4, Glyph Scale 1

There were now less streamlines, so individual streamlines could be viewed more easily. The arrows were hardly visible, so they were scaled up, by altering the Glyph Scale parameter of **advector** (Appendix A Figure A-20).

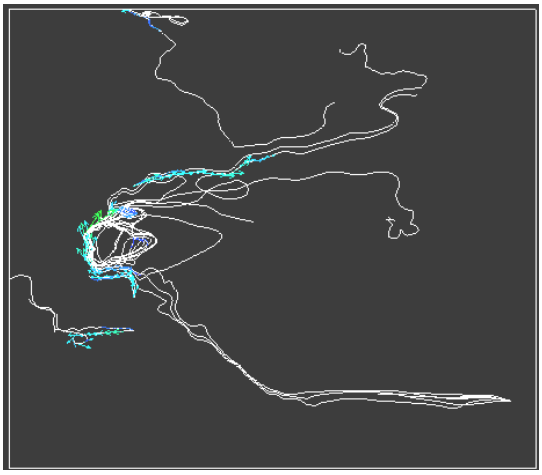

Figure III-37: Glyph Scale 20

Figure III-37 shows the arrows scaled up by a factor of 20. The arrows were still too small, so Glyph Scale was increased to 60.

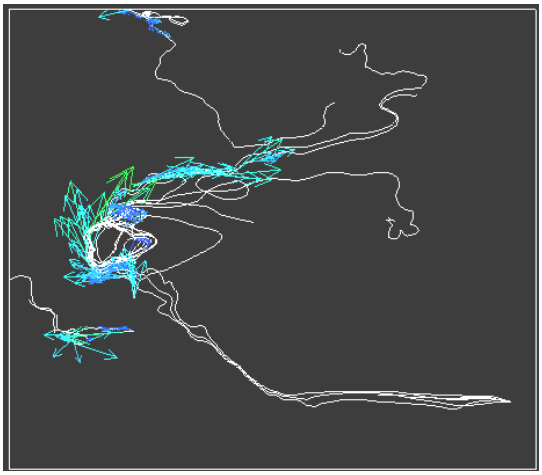

Figure III-38: Glyph Scale 60

The wireframe arrows in Figure III-38 are not clearly visible against the streamlines, so an **Arrow2** module was used instead of **Arrow1**, in the network shown in Figure III-34. **Arrow2** produced solid arrows, which would show up better than the wireframe arrows created by **Arrow1**, as shown in Figure III-39.

Figure III-38 shows that **advector** can produce streamlines and arrows. The left output of **advector** produces arrows and the right output produces streamlines. Either output can be disconnected, to show only arrows or only streamlines.

Figures III-37 and III-38 show that the arrows follow the same pattern as the streamlines. The **advector** module can create animations, to show the arrows moving along the streamlines. The arrows were scaled and animated by altering the parameters of **advector** (Appendix A Figure A-20).

- Glyph Scale alters the size of the arrows.
- Start Time sets the initial time of advection.

- End Time sets the time when advection ends.
- Step sets the intervals between time steps during advection.
- Release Interval sets how long arrows remain visible during animation.
- Time shows the time value currently being visualized.
- Run when pressed, starts or stops advection.
- Reset Time sets the time back to the Start Time.
- Cycle causes advection to restart from the Start Time, when the End Time is reached.

The Glyph Scale was set to 100, because this made the arrows large enough to be seen clearly. The Start Time was set to zero, and the End Time was set to 400, so that the advection would run for 400 time units. The Step was set to 50, so that when the advection was running, it would take time steps of 50 time units. The Release Interval was set to 0, so that old arrows would disappear as soon as new arrows appeared, so the motion of the arrows would be seen more clearly.

The Run button was pressed to start the advection. Below are images of the advection at different time steps, showing the arrows moving along the streamlines.

Figure III-39: time 0

Figure III-39 shows the position of the arrows at time 0, before advection started. Some large arrows can be seen on the streamlines near the east coast of America. The following images show how the arrows moved along the streamlines, at time steps of 50.

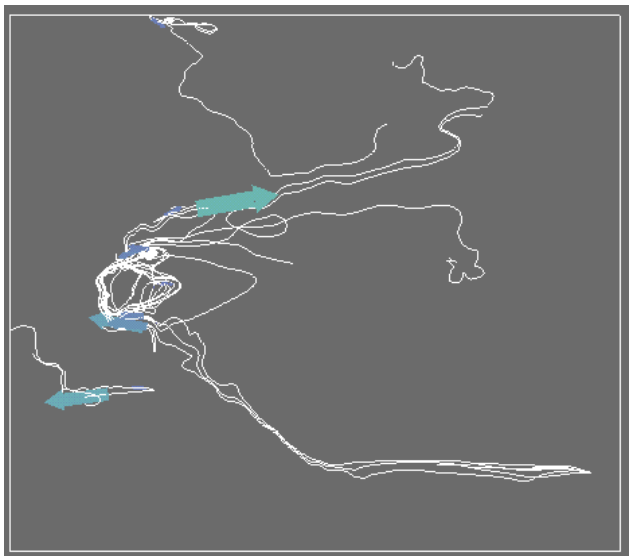

Figure III-40: time 50

In Figure III-40, at time 50, the arrows are smaller and bluer. The size and colour is determined by the magnitude of the velocity at that point, so this shows that the arrows have slowed down since time 0.

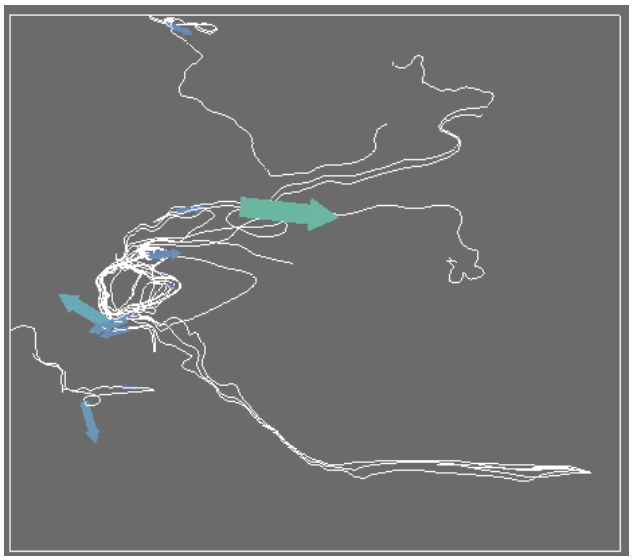

Figure III-41: time 100

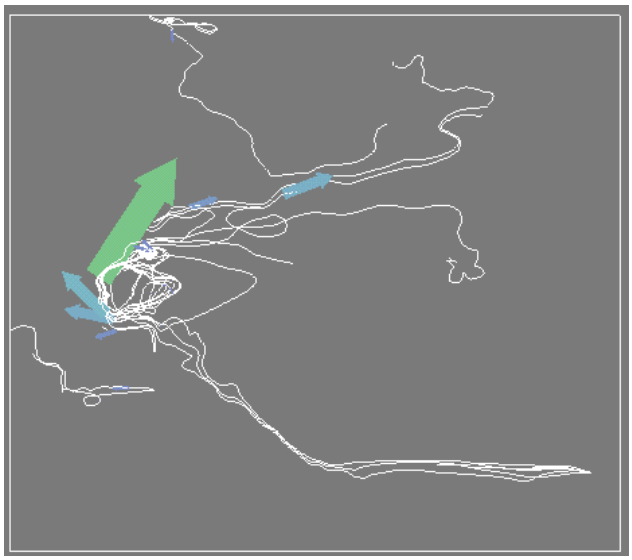

Figure III-42: time 150

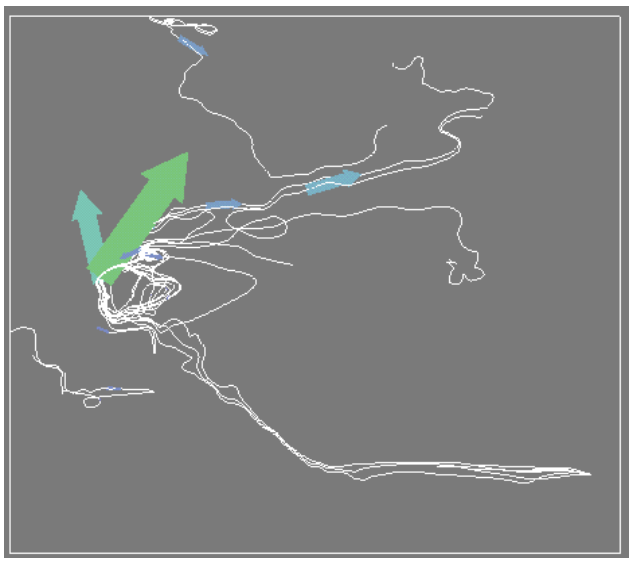

Figure III-43: time 200

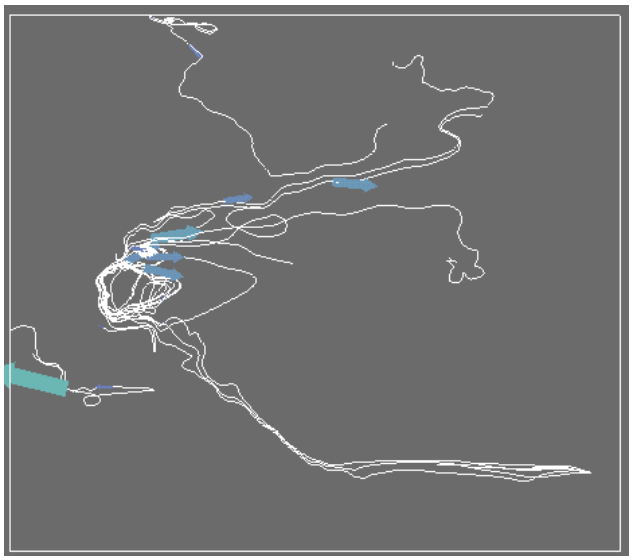

Figure III-44: time 250

Figure III-44 shows that the arrows have reached the mid-Atlantic. The arrows are now smaller, and darker shades of blue, because they have less magnitude. The arrows are moving more slowly across the Atlantic, in a northeasterly direction.

Figure III-45: time 300

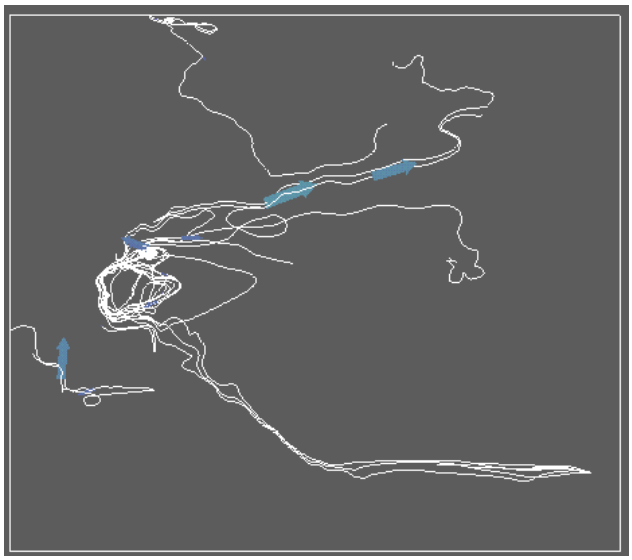

Figure III-46: time 350

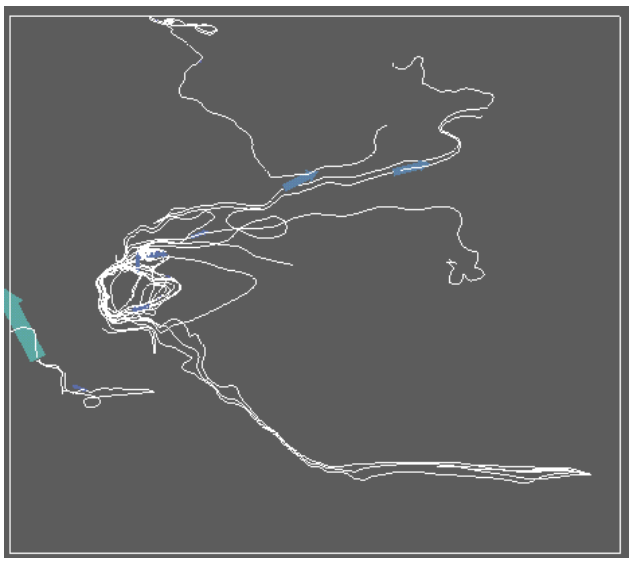

Figure III-47: time 400

Figure III-47 shows time 400, when advection was stopped. Small, dark blue arrows are visible in the north Atlantic.

The advection showed that the arrows were large at the start, near the coast of Florida, and moved quickly towards the mid-Atlantic, but then slowed down and became smaller when moving northeast across the North Atlantic. The size of the arrows depends on the magnitude of the velocity, so this suggests that the Gulf Stream is fastest near the coast of America.

3.4 Conclusion

This chapter described an example application of visualization of meteorological data, using AVS/Express as the visualization tool. The global ocean data showed images of the world scanned in 20 slices. The data had five components, giving the magnitude and directions of the velocity of water flow in the oceans, and the temperature.

It was necessary to import the data into AVS/Express, in order to visualize it. A field file was made, describing the structure and type of data (Section 3.1), and the **Read_Field** module was used to read the field file into AVS/Express (Section 3.2).

Each component of the data could be viewed using **extract_scalar** (Section 3.2). The datamap was altered to change the range of colours shown, which was useful for showing more information in the images (Section 3.3.1). Looking at the temperature component showed that the temperature was higher towards the equator. Looking at the magnitude component showed that the magnitude of current velocity was higher in the Gulf Stream than in the surrounding waters of the Atlantic Ocean.

The **orthoslice** module showed slices of data individually, which allowed different depths of the sea to be viewed (Section 3.3.2). The **set_null** module was used to set the data values of the land to null, so that the land would remain colourless, and only the seas would be visualized in the images.

The flow field of the Gulf Stream was examined using particle advection techniques (Section 3.3.3). Releasing and advecting weightless particles in a flow field can simulate a variety of effects. The particles can be represented as points. A particle's current position can be connected to the position it would be advected to. Repeating for several time-steps and accumulating the line segments gives a streamline. The **streamlines** module generates a set of lines showing the progress of weightless particles through a flow field. This was used to show the path of weightless particles through the Gulf Stream (Section 3.3.3.1). The **glyph** module was used to show the magnitude and direction of the current velocity at different points in the Atlantic Ocean (Section 3.3.3.2).

The **advector** module simulates the release of weightless particles into a flow field. This network was used to create an animation showing the progress of particles though the Gulf Stream (Section 3.3.3.3).

The advection network shown in Figure III-34 had the most useful features for visualizing the Gulf Stream, because it generates both streamlines and arrows, showing the magnitude and direction of the current velocity.

4 Visualizing Engineering Data

This chapter illustrates an example application using finite element data from an engineering simulation of a bridge and it's deformation under load. This describes how to get from the raw data to the rendered image.

The data is provided in the form of an Unstructured Cell Data or 'UCD' file, a common format for simulations of this type, and represents half of a simple beam bridge which was generated by the DANFE finite element analysis package. The other half of the bridge is a mirror image of that provided by the data.

The UCD data file contains the structural description of the bridge in 2336 nodes which form 1404 hexahedral cells. These cells are split between 6 separate cell sets with different material types defining to which cell set each belongs.

The data file also contains node data information (3 values per node) describing the x, y and z components of the displacement of the bridge under load due to gravity. The values of the displacements are in the range from $1x10^{-05}$ to $1x10^{-09}$ while the coordinates of the bridge elements themselves, are in the range of $1x10^{00}$. Consequently the displacements are of the order of one hundred thousand times smaller than the coordinates. The x-axis of the bridge runs along the length of the bridge, the y-axis lies parallel to the plane of the bridge surface and the z-axis lies parallel with the support towers.

4.1 Reading the Data

AVS/Express is provided with a specific reader for UCD data, called Read**_UCD**, in the Data IO library, (Appendix A Figure A-1), which is the natural choice of module for reading in this data. **Read_UCD** reads in a UCD file and converts it to an AVS/Express field. The field contains metadata with specific information describing the structure and type of data, so that AVS/Express can understand it. The field contains six specific cell sets which refer to separate polygon types making up the bridge structure. The field also contains three arrays of values for the x, y and z components of the displacement vector, one value in each array for each node in the structure.

Since Express provides the specific reader for the UCD data file, no field file is required for this example data. This is unlike the other data types covered by this document which require a field description file and the use of the **Read_Field** module.

The resulting field which is output from the **Read_UCD** module consists of a mesh of 2336 points and an array of 6 cell sets reflecting the material types in the model. These can be viewed independently if the user knows how and it can be seen that the 'materials' are bridge surface, vertical tower, cable, two types of concrete and bedrock.

4.2 Visualizing the Data

AVS/Express has libraries of predefined modules, which can be joined together to form a network. A network was made to process the data, using the Network Editor (Appendix A Figure A-1).[14] The **Read_UCD** module has two 'object' outputs, the red connectors, which can be directly connected to the viewer to allow an immediate visualization

[14] For instructions on how to find modules, refer to Appendix A Figures A-4 to A-5.

of the data read in as shown in Figure IV-1, below. Of the two ports available, connecting only one, that shown in the figure, makes sense in this context. The object rendered is a representation of the cell coordinate and connectivity data.

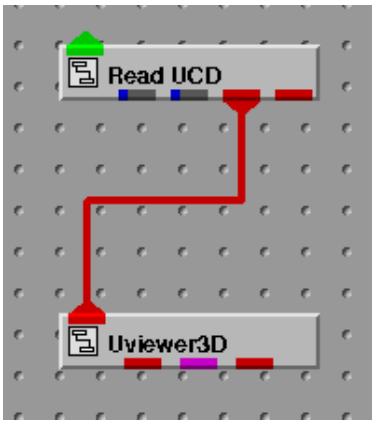

Figure IV-1: Minimal network using **Read_UCD**

Read_UCD: reads the data file and converts it to an AVS/Express field which is output on the blue/grey output connectors, it also produces a viewable object. This was used to read information from the field file, to create an image.

Uviewer3D: creates a 3D image in the DataViewer Scene window (Appendix A Figure A-2). This module defines all of the necessary components for the display portion of a graphics application, and is necessary for viewing the image in three dimensions.

The above network immediately produces a rendered image which shows the structure of the bridge as shown in Figure IV-2, below:

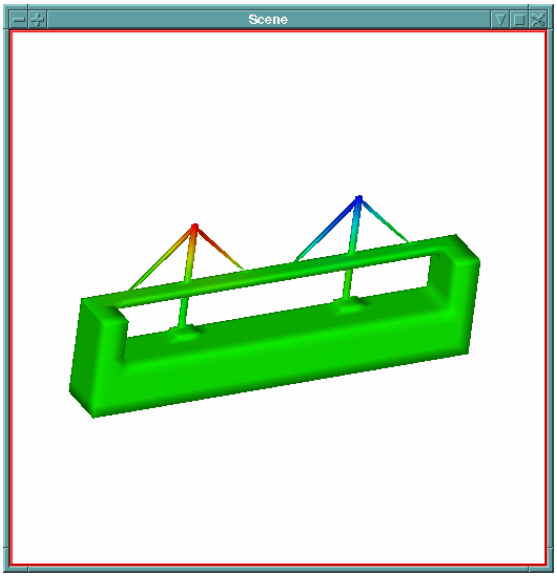

Figure IV-2: Output of the network shown in Figure IV-1.
To obtain this image it may be necessary to switch on bidirectional lighting using the control in the viewer main panel (see Figure A-3 in Appendix A).

4.3 Altering the images produced by this simple network

The network is used to produce an image, which can be manipulated to the desired format. Simple transformations such as scale, translate and rotate can be performed using the icons on the toolbar of the Dataviewer Pad window (Appendix A, Figure A-3). The parameters of each module can be altered. From the Editors menu, select Modules, then the module name, and its parameters will be displayed. The parameters can then be changed using the various widgets, such as sliding bars, or by typing new values into the various type-in boxes.

4.3.1 Altering the colouring component.

The 3D structure shown in Figure IV-2 is automatically coloured according to the magnitude of one of the components of the displacement values and the colour mapping appears to be significant but the user should be aware that this is an unknown factor since we do not know what component is being used and so the colour should not be assumed to be meaningful in advance of further consideration of the visualization properties. In order to obtain a view which is more meaningful we can make use of a module called magnitude to select individual components of the displacement vector at each node and colour the structure using this specific data value.

An example network for this is shown in Figure IV-3, below:

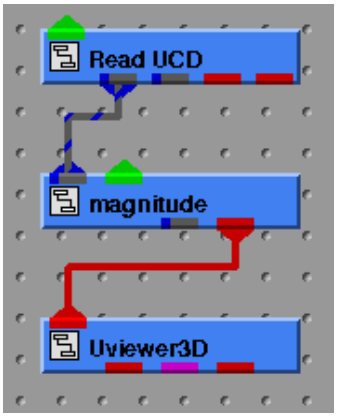

Figure IV-3: Addition of the magnitude module.

magnitude: takes an input AVS/Express field and outputs both a reduced field, having a single data item, and an object suitable for display in the UViewer3D.

Sample output images for this network are shown in Figures IV-4a, b and c. In each case the original scene, rendered by the simple network in Figure IV-1 is shown in the leftmost scene window and the new image, with data items selected through the magnitude module, in the rightmost. From these images we can see that the renderer is automatically selecting the first component, the displacement along the x axis, as that by which to colour the structure.

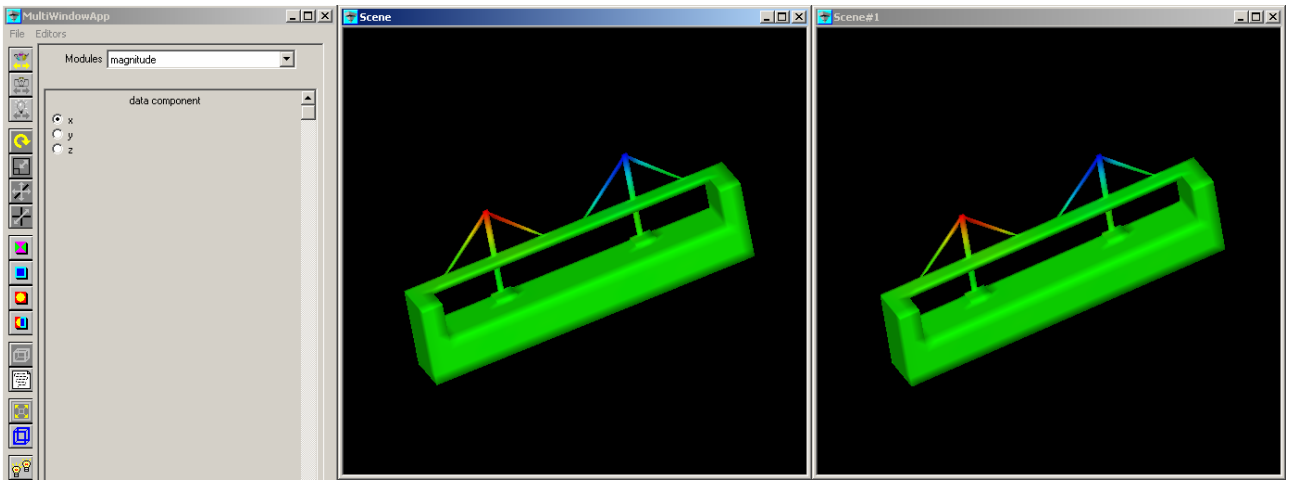

Figure IV-4a: Colourmap according to x displacement selected through the magnitude module.

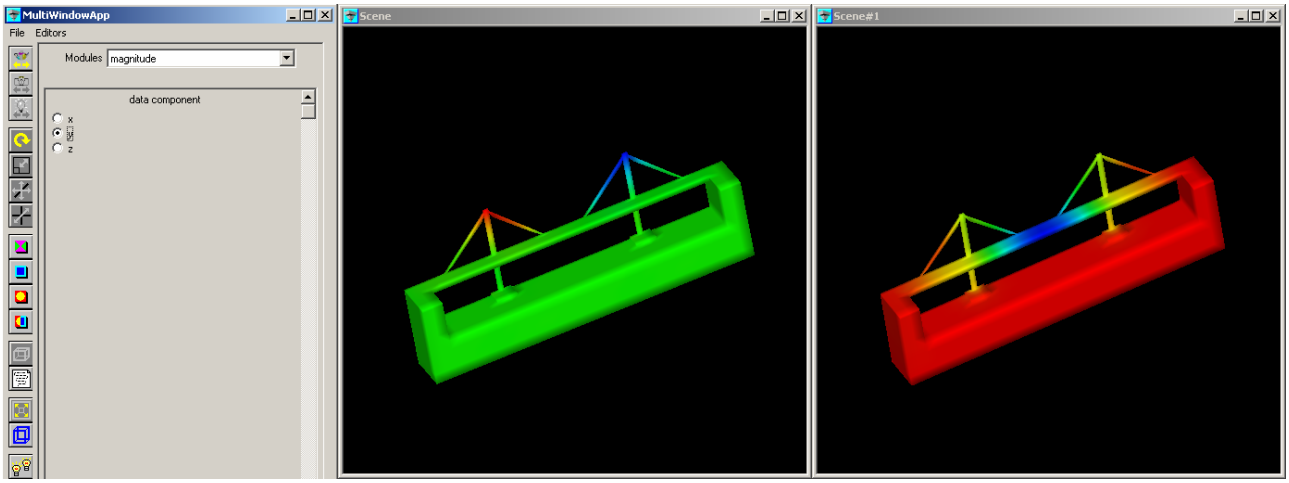

Figure IV-4b: Colourmap according to y displacement selected through the magnitude module.

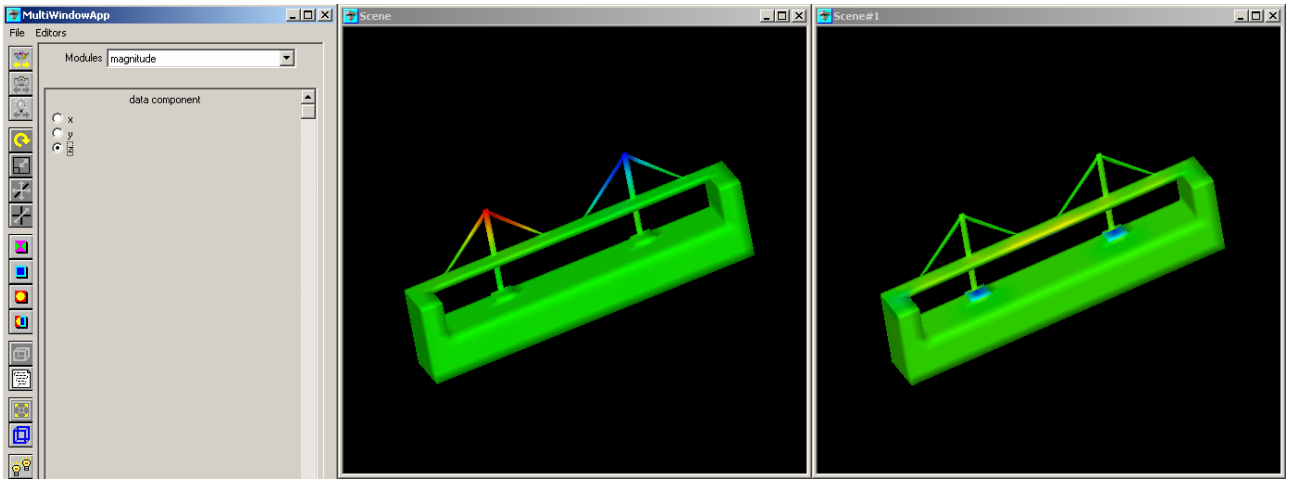

Figure IV-4c: Colourmap according to z displacement selected through the magnitude module.

4.3.2 Changing the Datamap

The images shown so far have been coloured according to default values, defined by the datamap. The displacement in the X axis, as shown in Figure IV-2, is such that the displacement is always towards the centre of the bridge and so is positive in one part of the bridge and negative in another. The default colourmap is such that the blue is the extreme of the negative end of the range and red the extreme positive end of the range. Changing the datamap could show different colours in regions of interest, where there is variation in data values. The user may also be only interested in the magnitude of the displacements at each point in the surface. This can be achieved as follows.

- Select Editors from the Dataviewer Pad window, and select Datamap (Appendix A Figure A-10). The Datamap Editor will now appear in the Dataviewer Pad window (Appendix A Figure A-11).
- Click on the Select Object… button at the bottom of the Dataviewer Pad window (Appendix A Figure A-3).
- From the Object Selector window, which now appears (Appendix A Figure A-21), select **magnitude** from the list of items, and click on OK. This will now show the datamap for the magnitude object in the datamap editor.
- From the Options menu, select Edit Color (Figure III-5). From the Color Range Mapping menu, choose Linear (Figure III-6). This gives a continuous range of colours, so a large variation of colours will be shown. Constant would produce only two colours – red and blue.

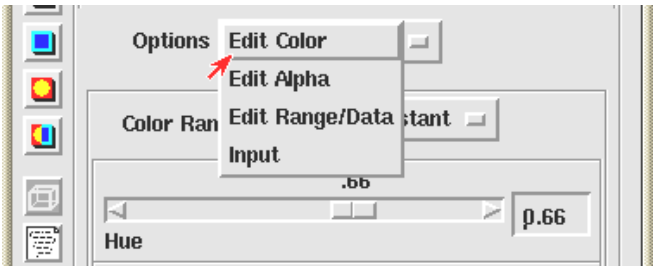

Figure IV-5: Editing Colour of Datamap

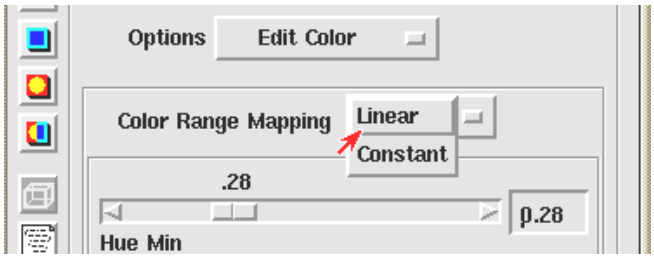

Figure IV-6: Editing Colour Range Mapping of Datamap

- We will now change the range of the datamap for the object ‘magnitude’ such that it goes from red at the extreme negative to green at zero and then to red again at the extreme positive. To do this actually requires two ranges and so the first step is to add a new second range by clicking on the Add Range button. The new range splits the range of the data in two and is initialized to null values and so appears in black as shown in Figure IV-8.

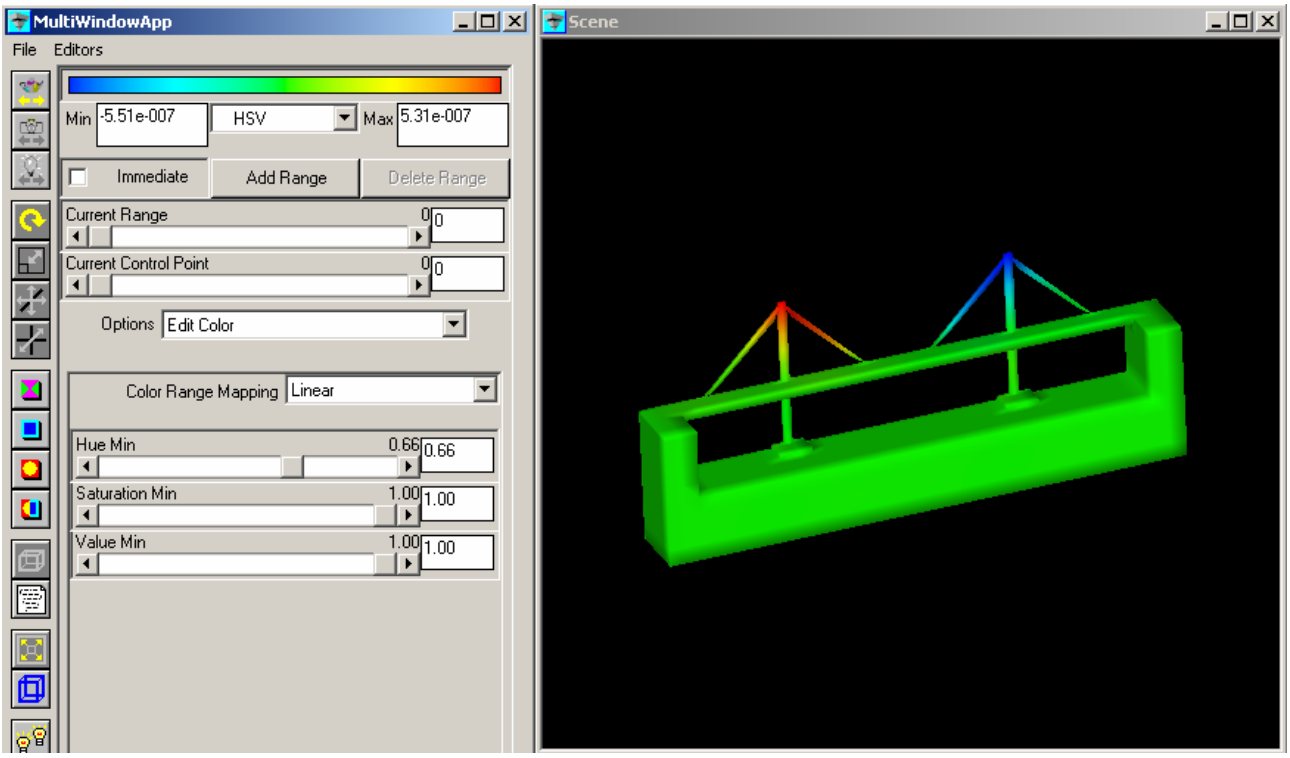

Figure IV-7: Editing Range of Datamap

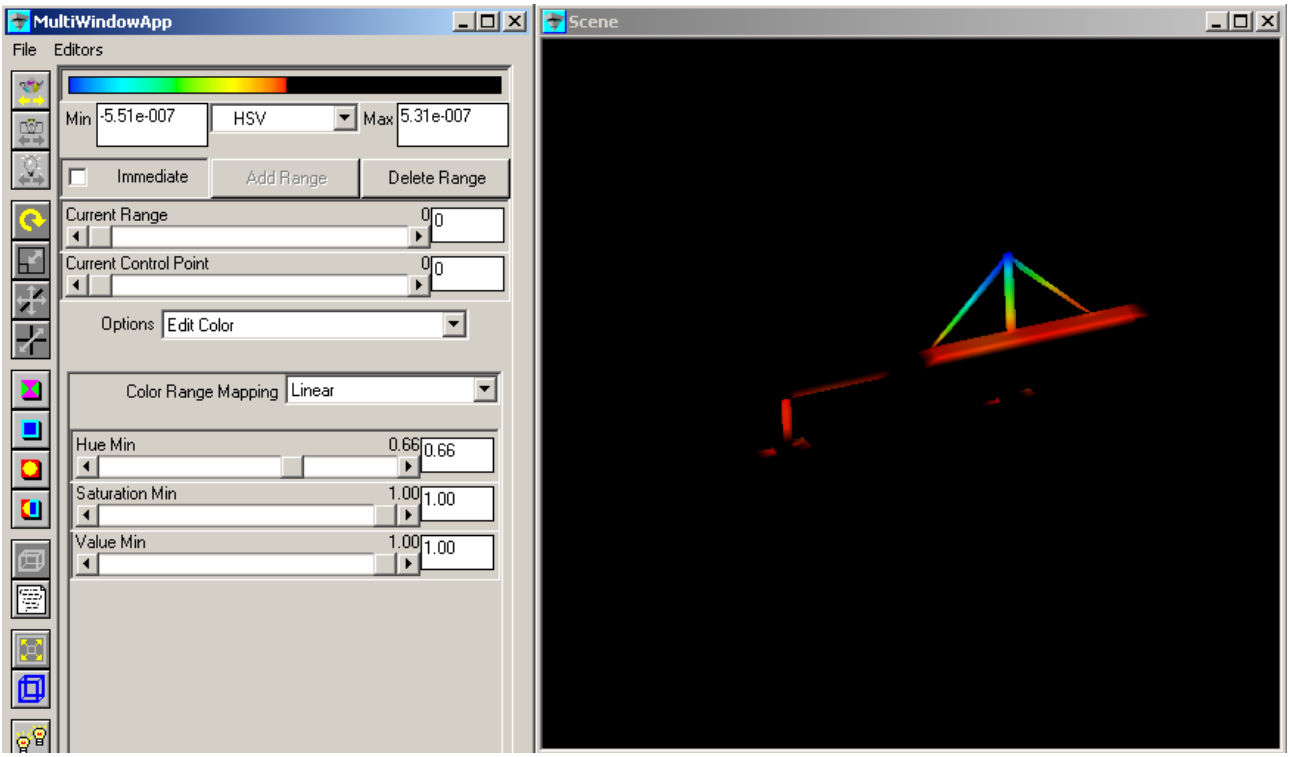

Figure IV-8: Datamap and scene showing effect of new second range.

Both the old range (range 0) and the new one (range 1) can now have the colour values adjusted as follows:

- Set both the Current Range slider to 0 and the Current Control Point slider to 0. The changes made will now affect the left-most point in the data range map.
- Move the Hue slider to 0 to change the colour at this point to red. The datamap will update to give a uniform red-red range.
- Now set the Current Control Point slider to 1, selecting the rightmost point of the left hand range.
- Move the Hue slider to 0.33 (or use the type-in box next to the slider) to set the colour of this point to green. The range will now be a linear colour change from red to green but the right-hand range will still be shown in black. The scene should now look like that shown in Figure IV-9.

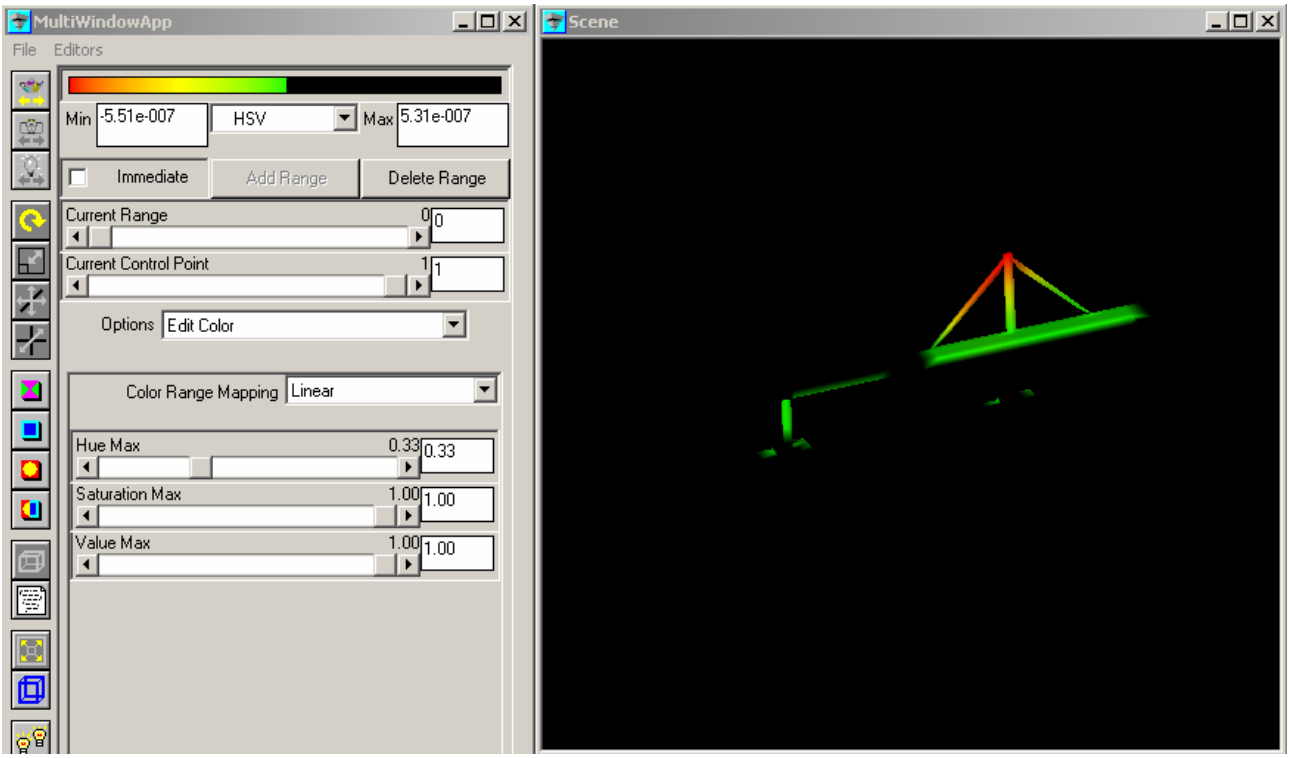

Figure IV-9: Datamap and scene showing effect of modified left-hand range.

- Set the Current Range slider to 1 and the Current Control Point slider to 0. The changes made will now affect the left-most point in the right-hand data range. The data values shown for the hue, saturation and value for this point are all <NULL> and so must be set by hand.
- Move the Hue slider to 0.33 (or use the type-in box next to the slider) to set the colour of this point to green. Set the Saturation and Value sliders both to 1.0, their maximum value.
- Set the Current Control Point slider to 1 to select the right-most point in the range map. Again the Hue, Saturation and Value will be set to <NULL> and must be set by hand.
- Finally, set the Hue slider to 0, and the Saturation and Value sliders both to 1.0, their maximum value to produce the final view as shown in Figure IV-10, below.

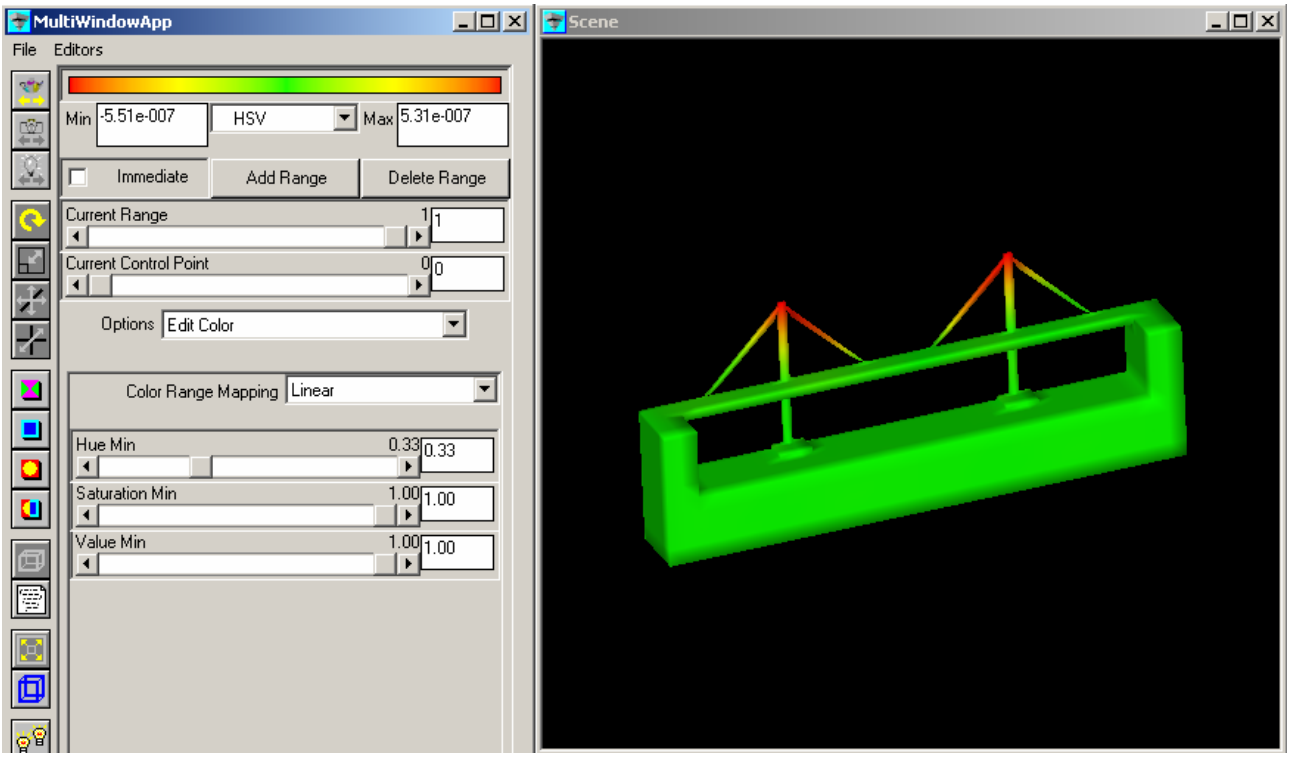

Figure IV-10: Datamap showing the modified ranges giving the effect of magnitude only to the scene.

4.3.3 Working with Vectors

As mentioned above, the data provided at each node is actually a vector quantity, the x, y and z components of the displacement of the bridge when placed under load due to gravity. We have, so far, been examining this quantity as a set of 3 independent, scalar values but this is not really an acceptable way to consider this data which should be examined as a vector quantity. In order to do this a procedure must be carried out to convert the data as read, three scalar arrays, into an interleaved array which Express can handle as a vector quantity. This process is not trivial but will now be worked through in easy stages to explain what is being done by each component of the network developed.

The process essentially consists of 3 phases:

1. Separating the three arrays into separate arrays outside of the field file.
2. interleaving the three into a single array whose format is, in essence $x_1,y_1,z_1,x_2,y_2,z_2,\ldots,x_n,y_n,z_n$
3. Re-merging the interleaved array with the structural information from the original field to create a new field with a single (3-)vector value for each data point.

These three steps are achieved through the network shown in Figure IV-11, below:

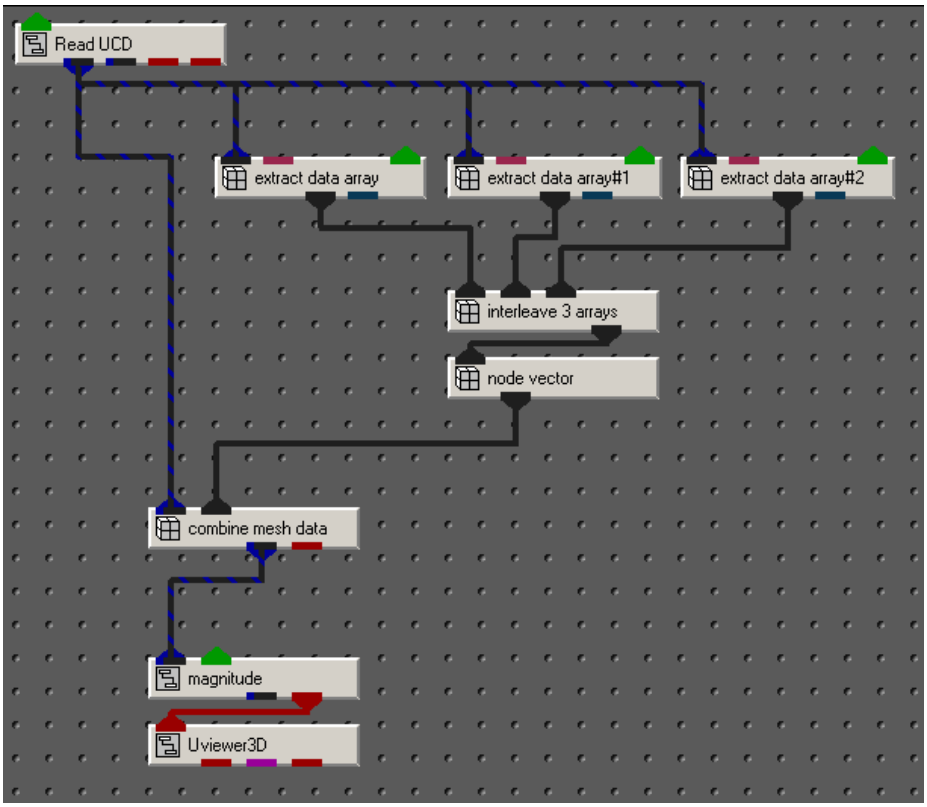

Figure IV-11: Network converting the scalar displacement arrays into a single vector.

extract_data_array: This module, found in the Field mappers/Array extractors library in the Main library set, extracts a single array (specified through the module's user interface) and outputs that array on its data output port.

interleave_3_arrays: This module, found in the Field mappers/Combiners library takes three arrays, of equal dimension, from three input ports and outputs a single array containing interleaved values from the three inputs. This output tarray can be interpreted as a 3-vector array by Express.

node_vector: This module, found in the Field mappers/Data_Mappers library takes the vector array and outputs the same array combined with additional components to allow Express to interpret the array as node data values.

combine_mesh_data: This module, found in the Field mappers/combiners library takes the node vector array and the original field file and outputs a new field, a Mesh+Node_Data, which contains the mesh of points, cells sets.

The output from this network is shown in Figure IV-12, below:

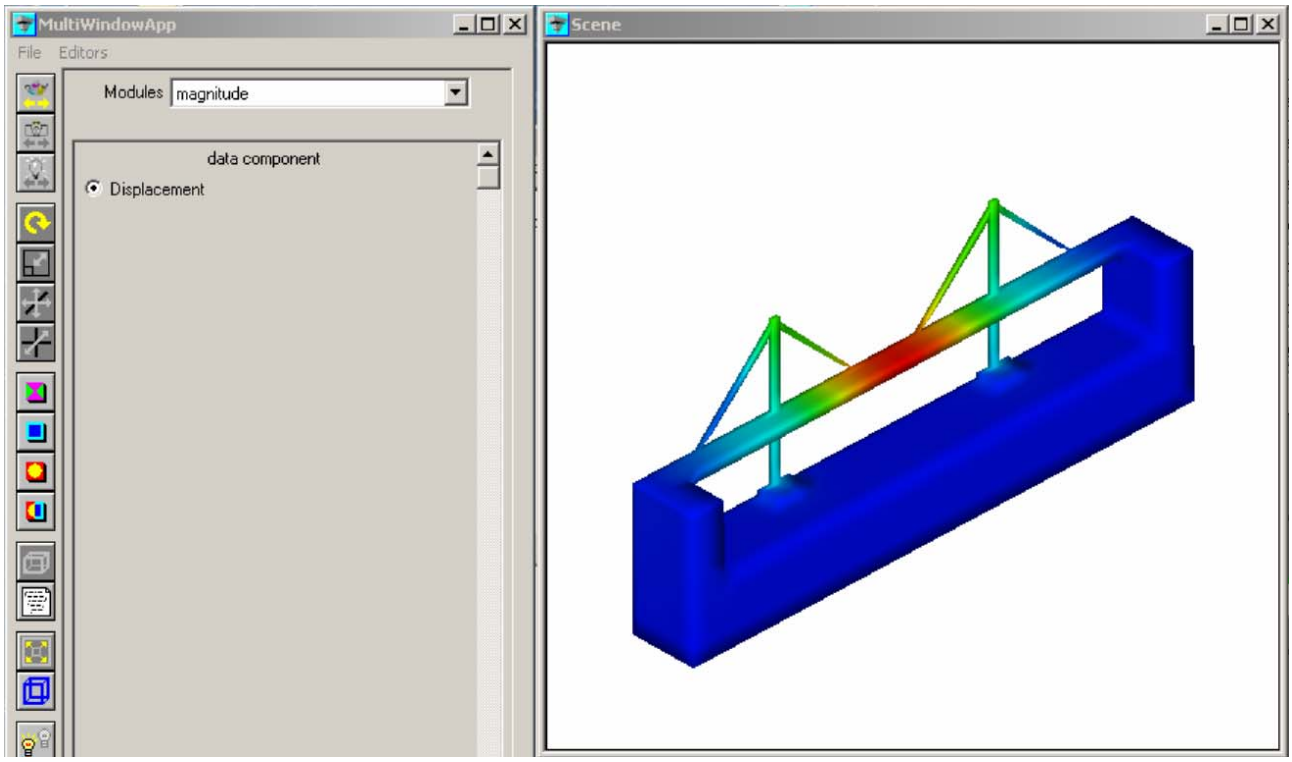

Figure IV-12: Network converting the scalar displacement arrays into a single vector.

As before, the magnitude module is used to select the required data value and, since the node data is now a vector, to compute the magnitude of that vector. This magnitude is then used to index into the datamap to determine the colour of each point.

4.3.4 Using Glyphs.

Since the node data is now a vector, both the magnitude and direction can be visualized for each node in the structure. The simplest way to achieve this is to use a glyph, in the shape of an arrow, to represent both the direction and magnitude of the displacement at each node. This can be achieved by extending the network shown in Figure IV-11, and adding a glyph module and connecting a glyph geometry, arrow1, to that module.

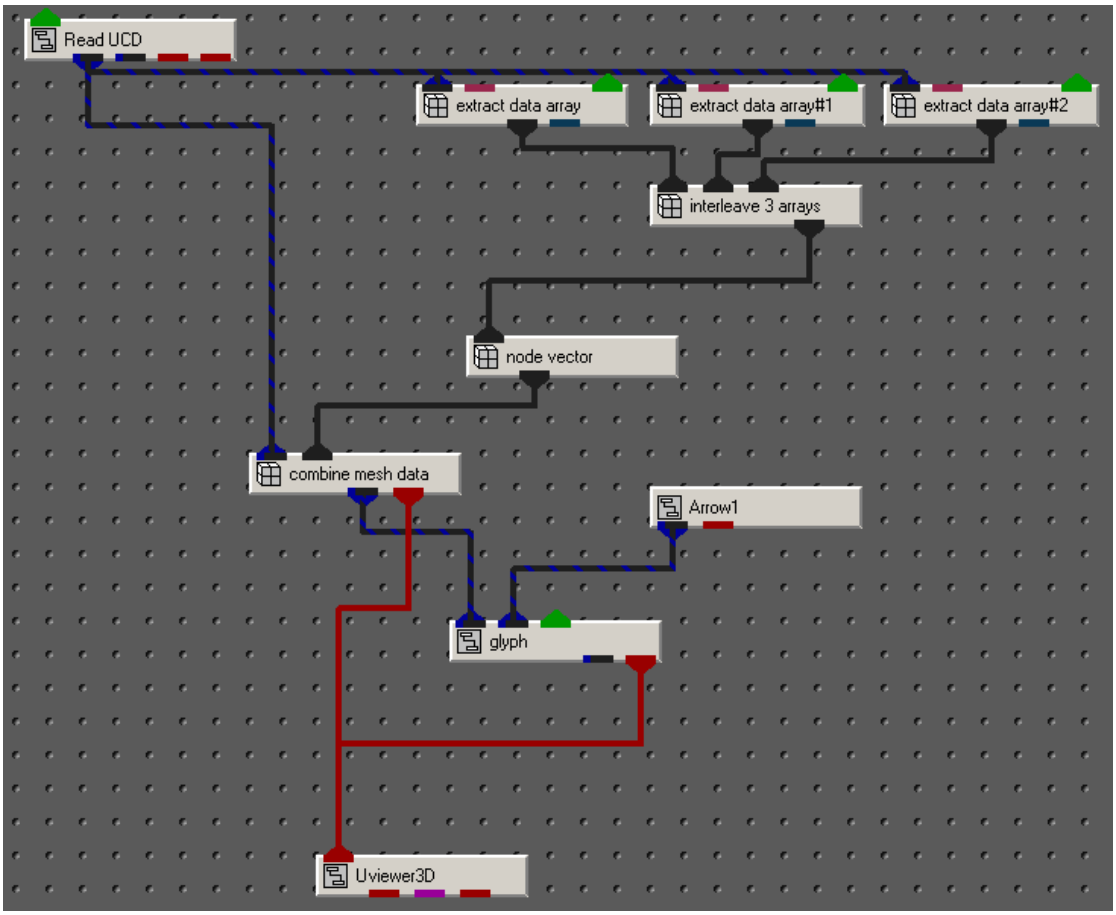

Figure IV-13: Network adding glyphs to the vector node data.

glyph: This module, found in the Mappers library, takes a field and a glyph geometry as input and outputs a renderable object with a copy of the glyph mapped onto each node of the structure and scaled and directed according to the data value at the node.

Arrow1: This module, found in the Geometries library, provides a basic geometry for a simple arrow for use with the glyph module.

As noted in the original data description, the displacement is typically at least one hundred thousand times smaller than the coordinates themselves and so the glyphs will be too small to be seen at the scale of the image of the bridge structure. To compensate for this select the glyph modules control panel and use the scale function to make the displacement vectors visible. The default range of the scale slider is far to small to be useful with this data set and so, in order to do this, it will be necessary to bring up the type-in panel for the scale slider using the button to the right of the slider. Set the maximum value to a very large number, perhaps 100,000,000 and then set the value to a number of the order of 1,000,000. Vary the scale value until the arrows are clear as shown in Figure IV-13a, below:

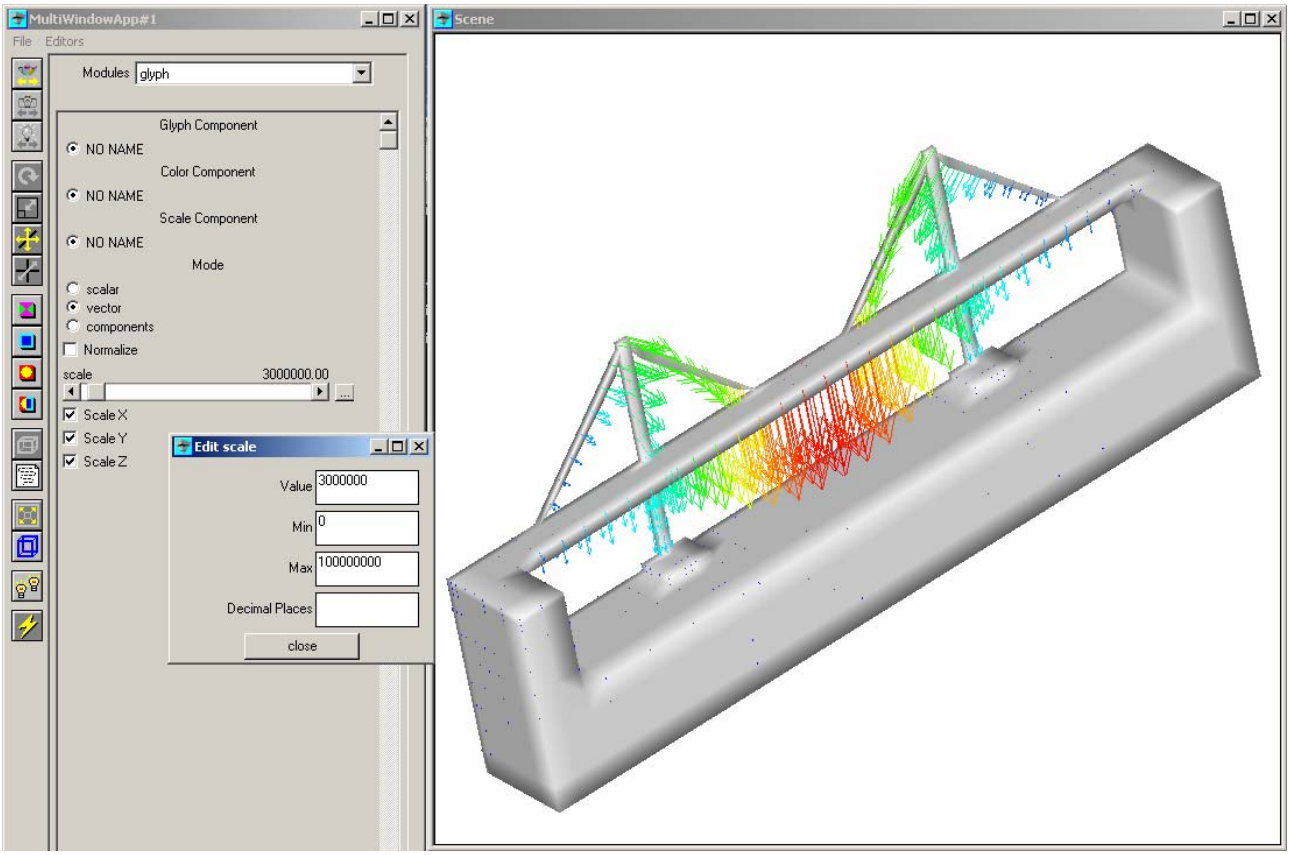

Figure IV-13a: arrow1 glyphs mapped to the bridge structural nodes.

A glyph is placed at each node point in the structure and many of these are inside of the volume of the bridge materials and so are invisible in the image above. Removing the bridge structure by disconnecting the object which represents it (the red output of the magnitude module) would allow these to be seen but it would lose the context of the location in the bridge structure. In order to overcome this we can render the material of the bridge structure translucent using a object properties controls. This is achieved through the following procedure:

1. Click on the Select Object… button at the bottom of the Dataviewer Pad window (Appendix A Figure A-3) to bring up the object selection window.

2. Select the combine_mesh_data object.

3. In the application window click on the Editors tab and select the Object item.

4. In the Object menu which is now displayed, select the Properties item to bring up the display of parameters affecting the appearance of the object being displayed.

5. In the Type menu, select the item Surface.

6. Set the value of Opacity to something like 0.25 and the bridge structure will become translucent, allowing the glyphs within it to be seen.

The results of this procedure are as shown in Figure IV-13b:

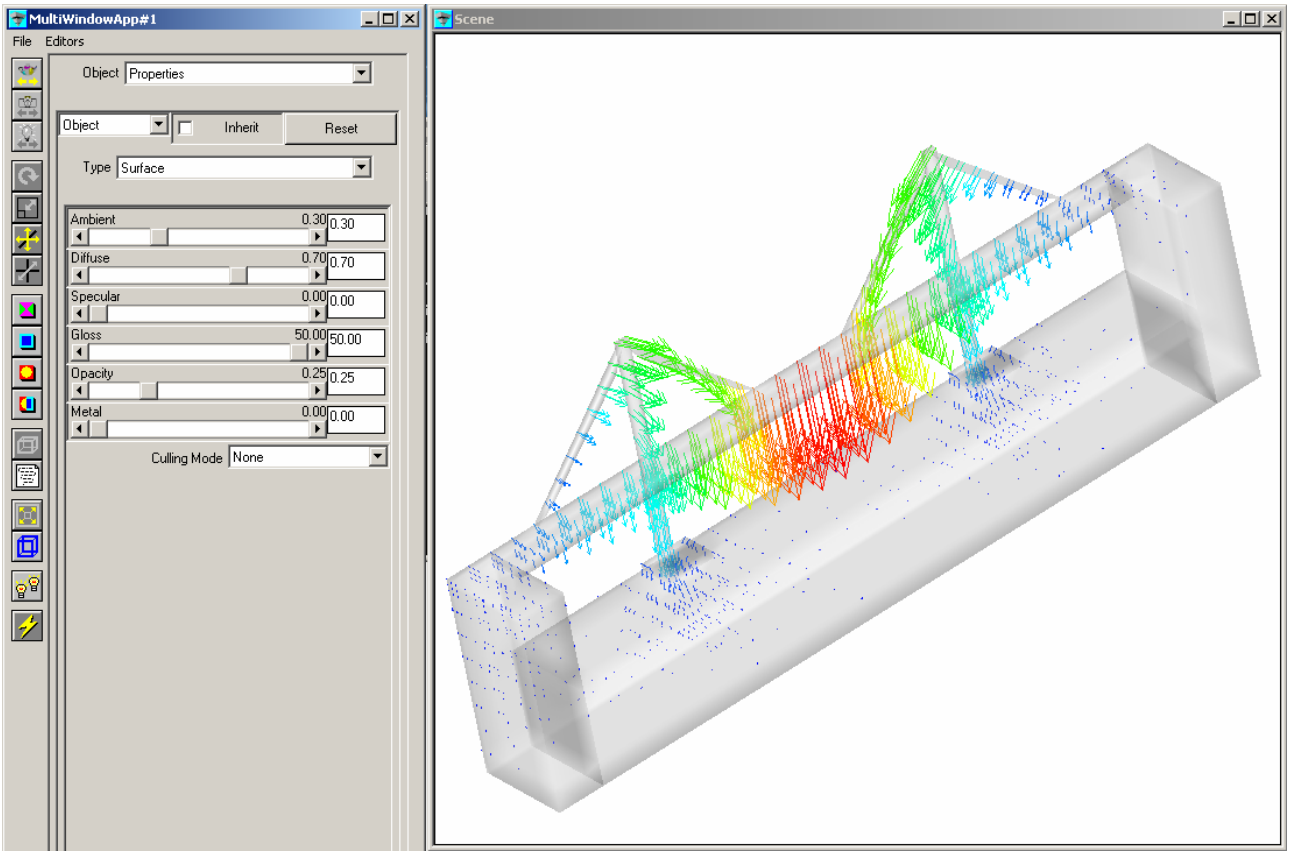

Figure IV-13b: arrow1 glyphs within the translucent bridge structure.

The user can now see all of the nodes in the context of the structure and can see that the values within the base of the bridge, its foundations, are extremely small. The glyphs in the foundation can be made visible by selecting the Normalize option in the glyph user 9interface (see Figure IV-13a) and setting the scale value to 1. Normalization of the glyphs ensures that, while their direction is retained, the size of each glyph is the same so the user can see the glyphs throughout the structure regardless of the magnitude of the displacement.

The large number of glyphs, one for every node in the structure can be confusing where a very large and complex structure is involved. This problem can be alleviated in simple rectilinear structures using standard modules such as downsize but in unstructured data, such as that read from UCD, this is more complex. Fortunately a module is provided by the International AVS Centre (http://www.iavsc.org/) to provide exactly this facility. That module is called DVdownsize_scat and takes an irregular grid as input and outputs a new grid with a reduced number of data points. This effectively reduces the number of glyphs which will be drawn within the structure. A slider is provided to set the degree of downsizing of the points which is applied. The DVdownsize_scat module should be obtained from the IAC web site, compiled into express following the instructions provided with that module and then incorporated into the network as shown in Figure IV-14 and the output from this new network is shown in Figure IV-14a:

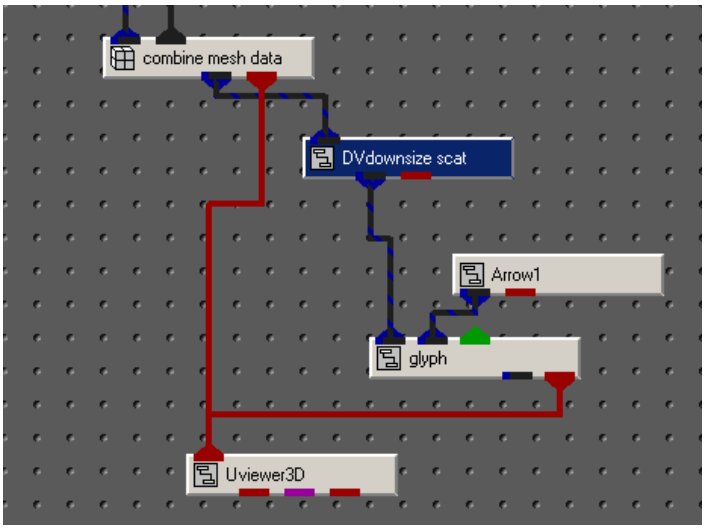

Figure IV-14: Modified network including the DVdownsize_scat module.

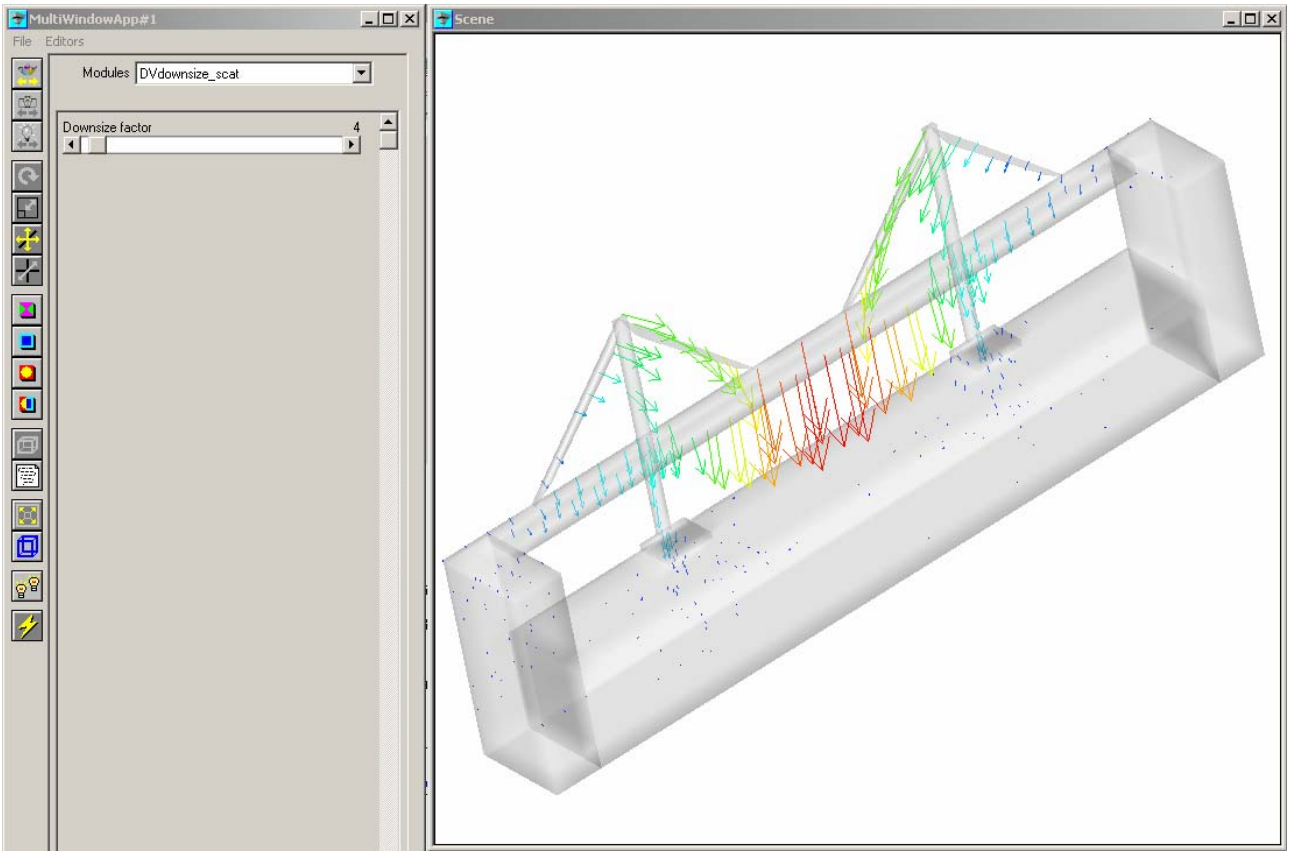

Figure IV-14a: reduced number of arrow1 glyphs within the bridge structure.

4.3.5 Coordinate mathematics.

Since the node data with which we have been provided are displacements, it is also possible to consider the use of these displacements to distort the co-ordinate structure with which we have been provided to represent the displacement of the structure by an actual visible movement of the bridge. In order to achieve this we can make use of the

coordinate_math module which, in a manner similar to the data_math module takes two input fields and adds their coordinates together according to user specified formulae.

The network used to achieve this goal is shown in Figure IV-15, below:

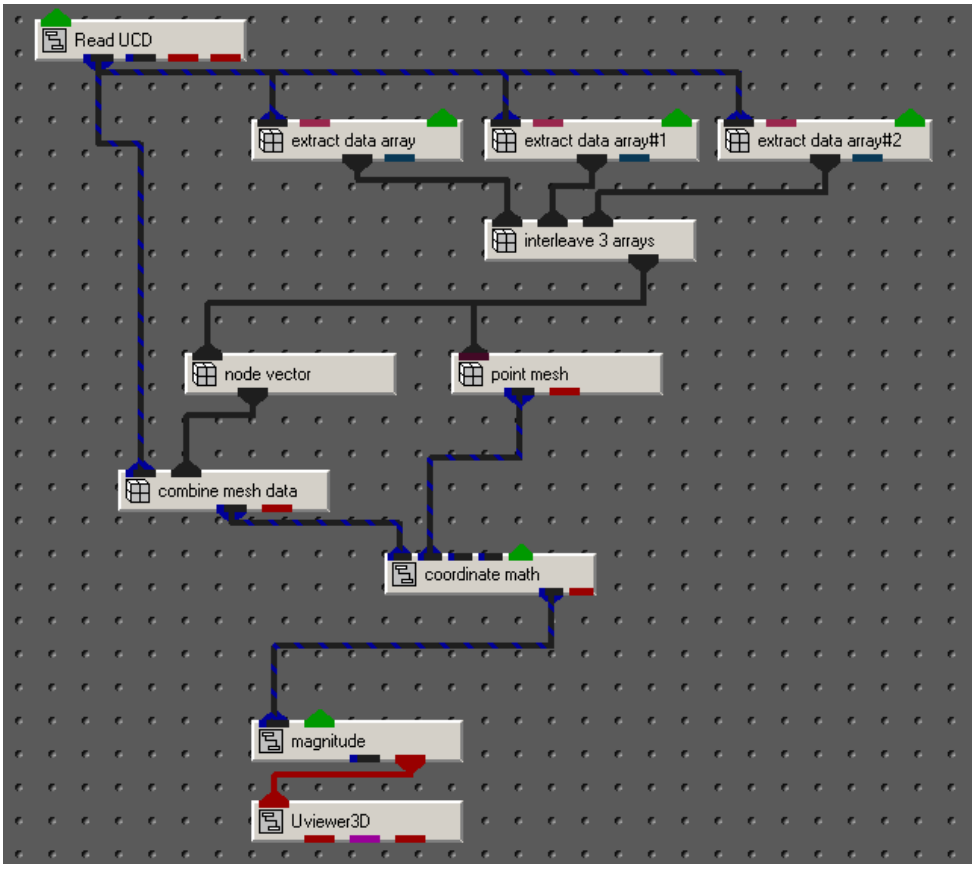

Figure IV-15: coordinate math module adding the coordinates of the two input meshes.

point_mesh: This module, which is found in the Mappers/Mesh Mappers library, takes the vector of displacements as input and converts them into a mesh of co-ordinates for output.

coordinate_math: this module, which is located in the Filters library, is used to carry out mathematical operations on the coordinates of the fields passed into it. The output is a combination of the node data from the input fields and the new calculated coordinate set.

The coordinates generated by the coordinate_math module are the result of formulae set in the user interface for the module. In this case they are simply the sum of each of the three ordinates for inputs 1 and 2 as shown in Figure IV-16. Note that it is also necessary to set the value of nspace for the output field to 3.

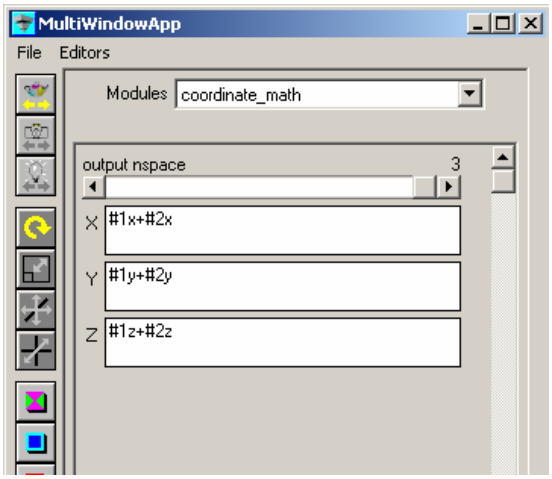

Figure IV-16: parameters for the coordinate math module adding the coordinates of the two input meshes.

The output from this network, with these parameters is, unfortunately, not very informative since, as has been described earlier in this section, the displacements are several orders of magnitude smaller than the coordinates of the structure and so the displacements must be scaled in order to affect the structure coordinates sufficiently for the observer to discern the change. This can be done by simply altering the formulae for the coordinate_math module to those shown in Figure IV-17, with the effects shown in that same figure.

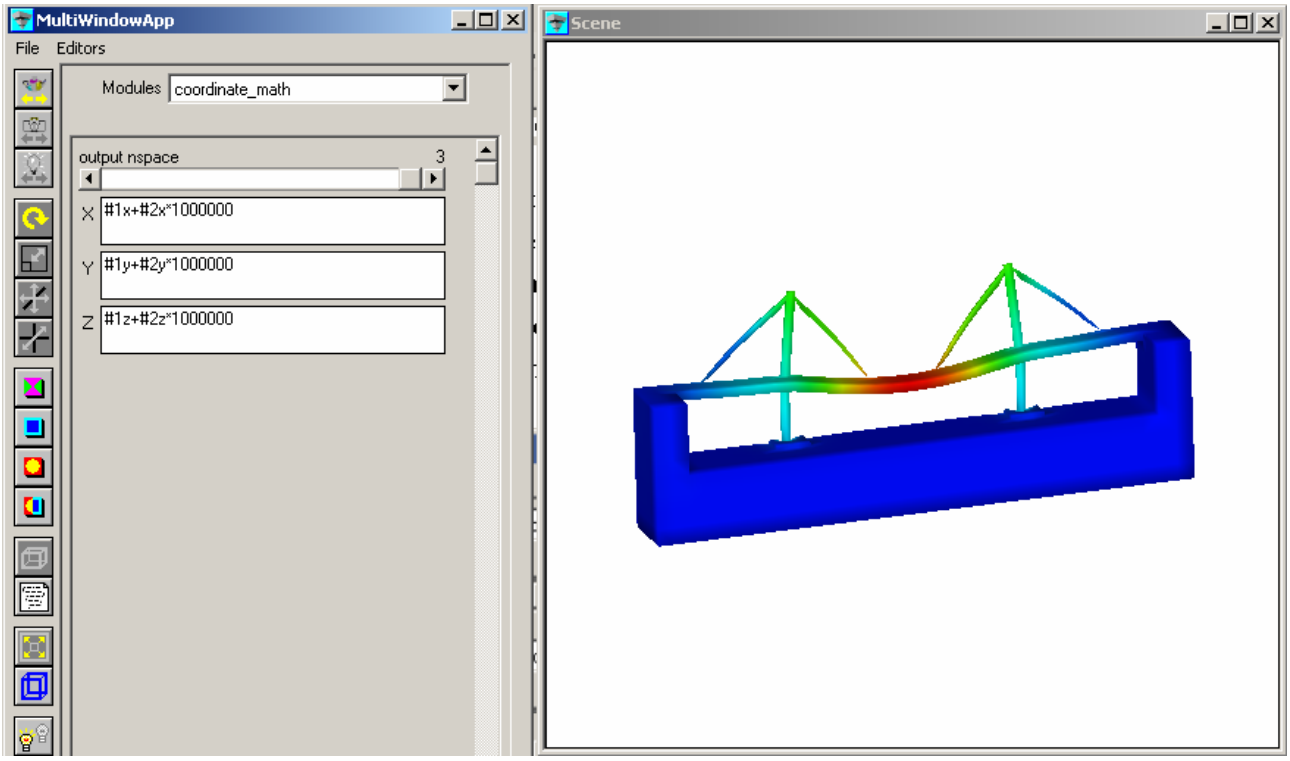

Figure IV-17: scaled values for the displacements allow the exaggerated deformation to be seen.

It is left as an exercise for the reader to make use of a user interface widget to control this scaling factor and allow interactive adjustment of the degree by which the displacement is exaggerated.

4.3.6 Isosurfacing.

Another method which may be suitable for data of this type may be the use of an isosurface module to locate regions of the structure within which the displacement exceeds a certain tolerance. This can be achieved using the network shown in Figure IV-18, below which takes the vector displacement and computes an isosurface for a certain value of the magnitude.

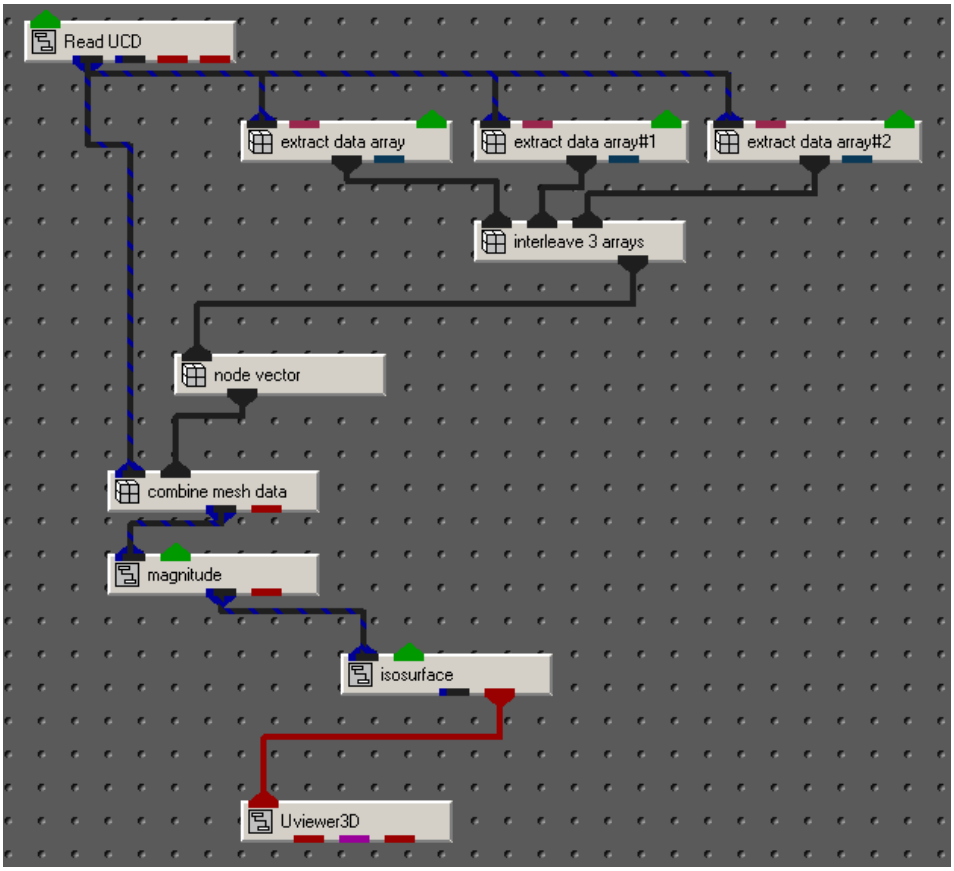

Figure IV-18: Network to isosurface the magnitude of the vector displacement.

The parameters of the user interface and the rendered display are as shown in Figure IV-19, below:

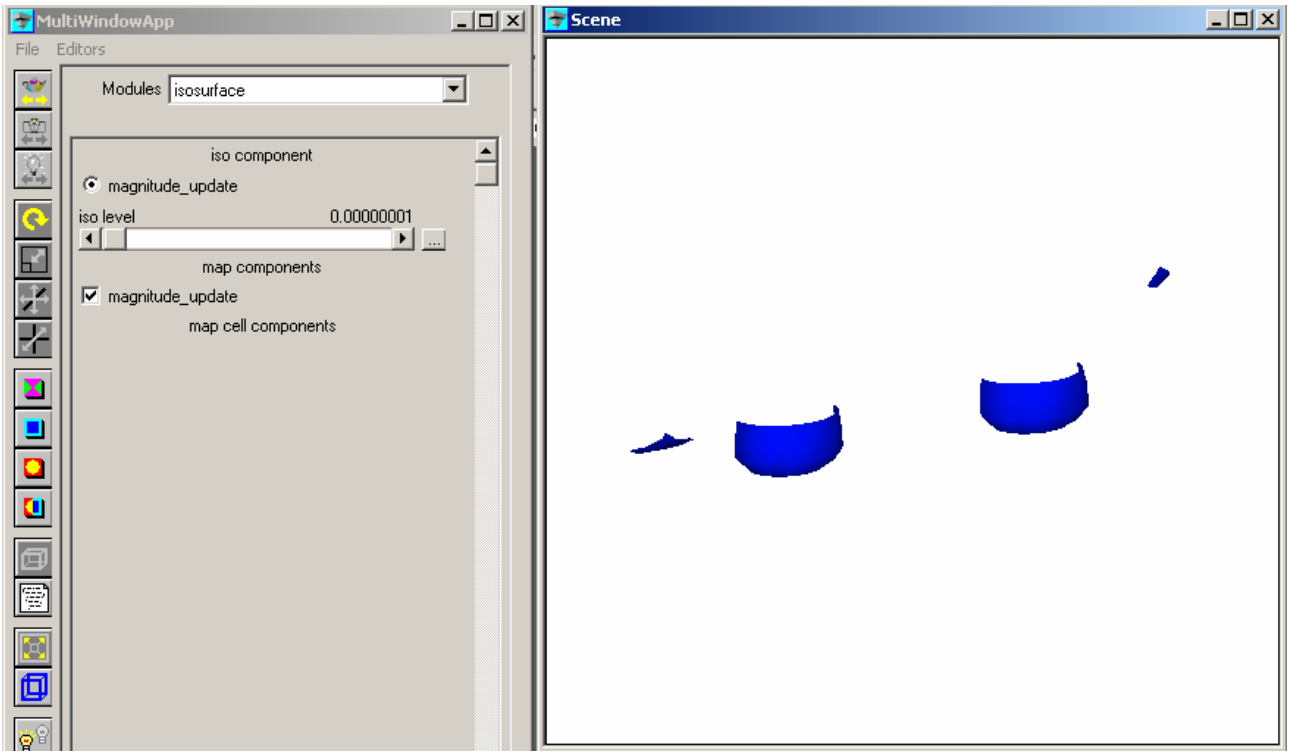

Figure IV-18: Isosurface of displacement in bridge structure and surround.

Obviously this image is of little use without the context given by showing the bridge structure. Unfortunately, to put back the bridge structure, which can be easily achieved by simply connecting the red output of the magnitude module to the red input on the UViewer3D, but the bridge drawn will be opaque and will obscure the detail of the isosurface. In order to get around this problem it is possible to render the bridge structure translucent, as in the case of the glyphing method described above, by reducing the opacity setting of the object. This is achieved through the following procedure:

1. Click on the Select Object… button at the bottom of the Dataviewer Pad window (Appendix A Figure A-3) to bring up the object selection window.

2. Select the magnitude object.

3. In the application window click on the Editors tab and select the Object item.

4. In the Object menu which is now displayed, select the Properties item to bring up the display of parameters affecting the appearance of the object being displayed.

5. In the Type menu, select the item Surface.

6. Set the value of Opacity to something like 0.25 and the bridge structure will become translucent, allowing the isosurface within it to be seen.

The results of this procedure are as shown in Figure IV-19:

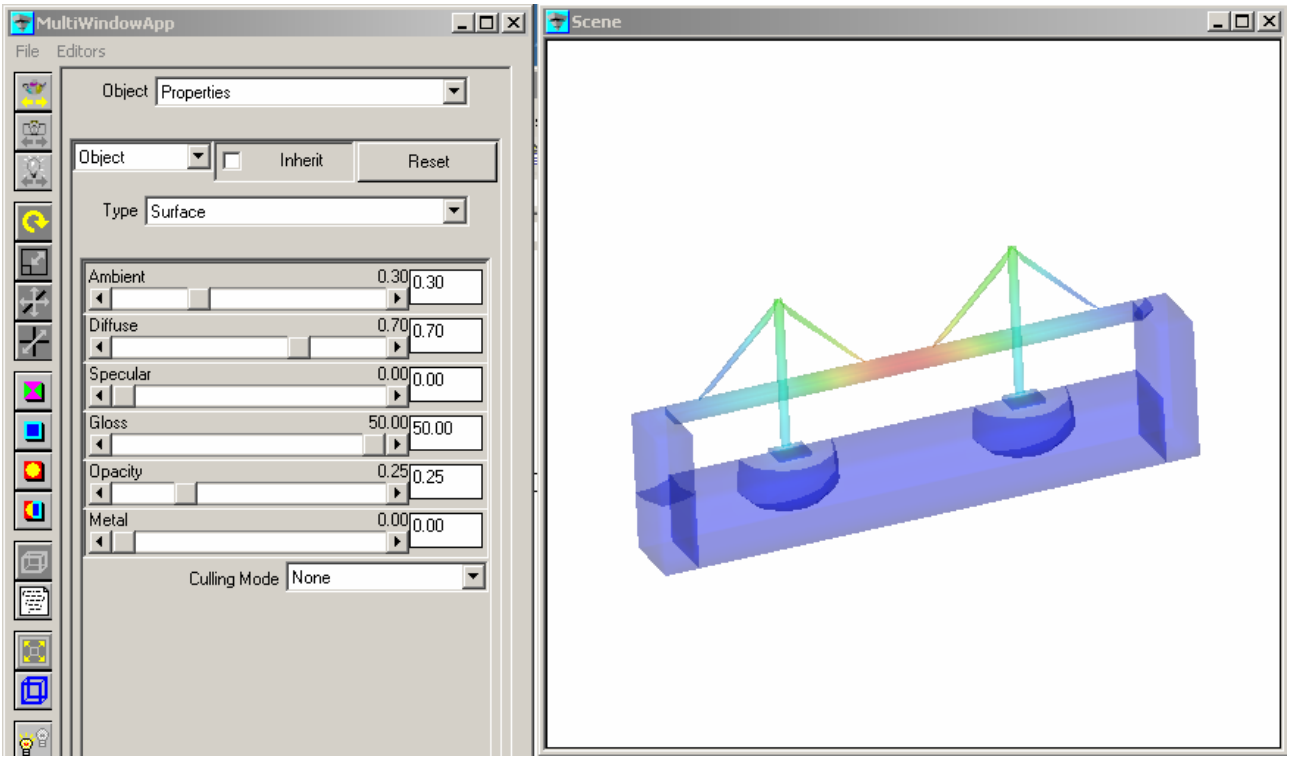

Figure IV-19: Isosurface of displacement visible through the translucent bridge structure.

4.4 Conclusions

AVS/Express can manage UCD data fairly well once it has been read into the system and the UCD reader is capable of reading well formatted UCD data produced by a range of Finite Element software packages. The data can be difficult to manipulate within the Express system due to the nature of the arrays of data read and the need to manipulate the arrays of ordinates and data and interleave them to produce vector data is difficult but standard modules provided with Express can be used to make this process easier.

Standard methods for visualization of structural data are available using simple structure renderings with colour for scalar data values and glyphs to show vector values are available but the user should also consider methods less strongly associated with structural data such as volume and surface rendering methods which can be effective with some scalar variables, nor should the user overlook the use of transparency and other object ‘material’ properties in the generation of the rendered images where these can provide access to extra information and greater clarity of insight.

Appendix A: AVS/Express windows

Figure A-1 shows the AVS/Express Network Editor, used for creating and displaying networks.

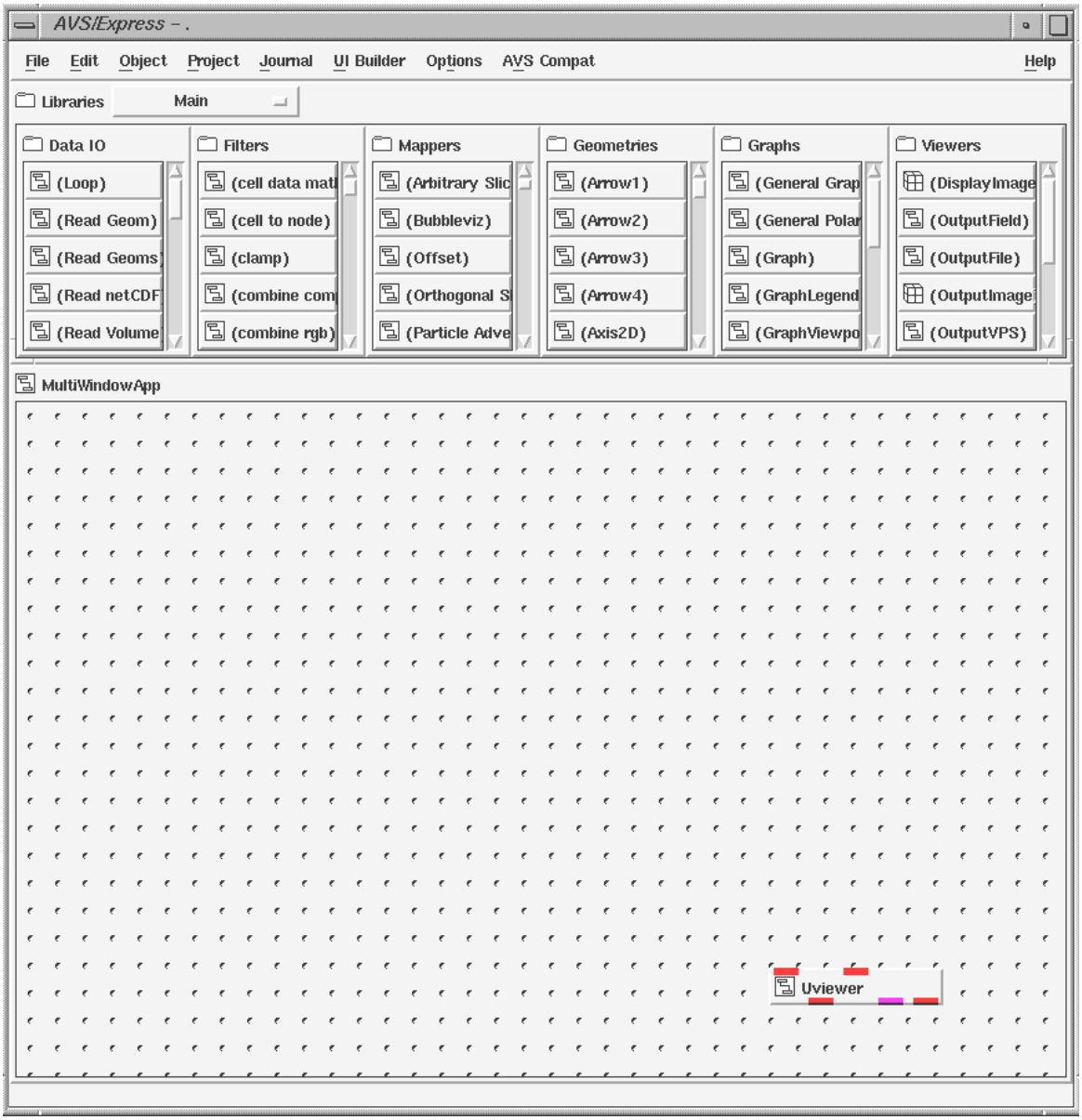

Figure A-1: Network Editor

Figure A-2 shows the AVS/Express Dataviewer scene window, used for displaying rendered images.

Figure A-2: Dataviewer Scene

Figure A-3 shows the AVS/Express Dataviewer Pad window, used for altering parameters of modules and viewer features.

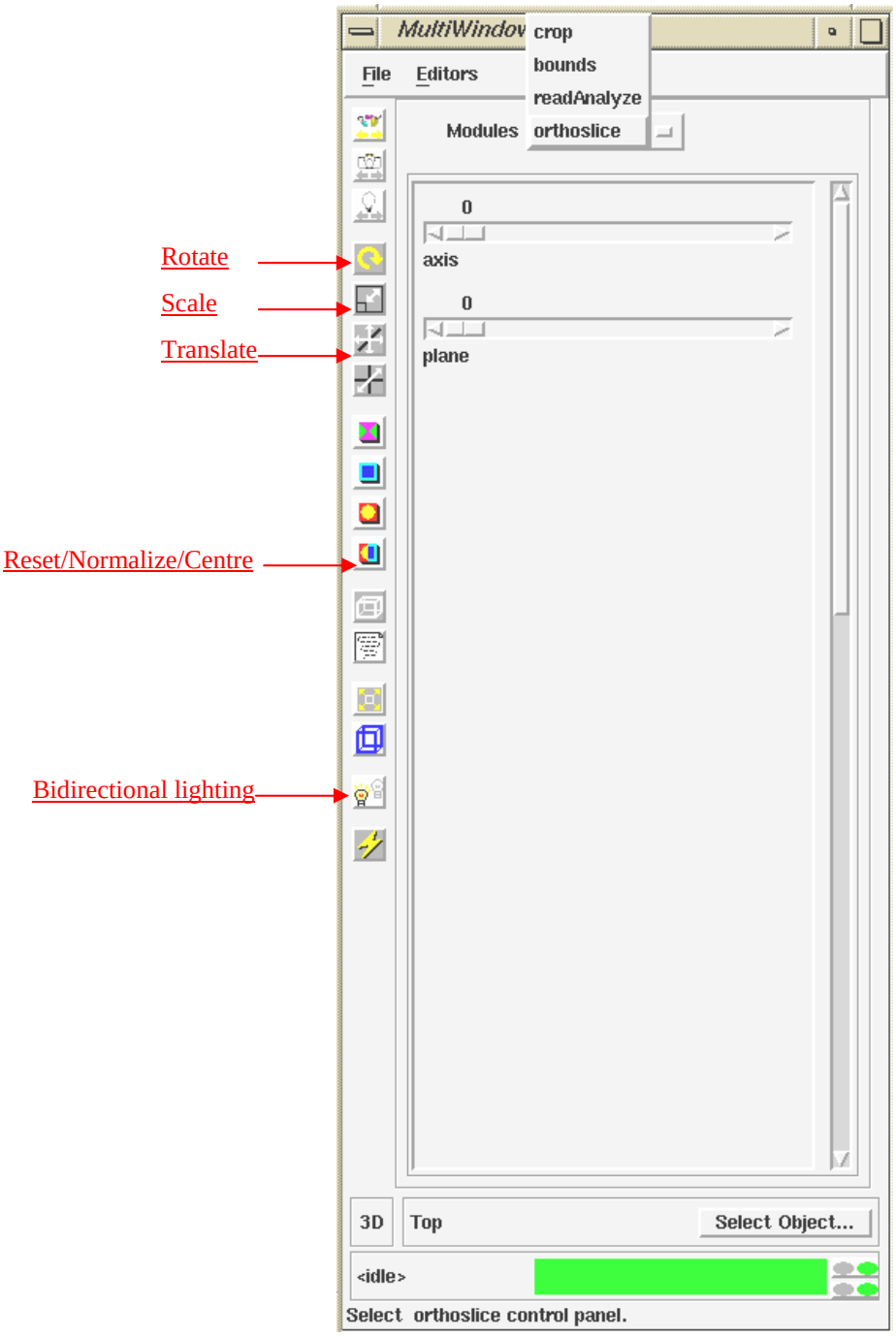

Figure A-3: Dataviewer Pad

When building networks, the Object Finder can be used to search for a particular module. From the Network Editor click on Object, and from the menu that appears, select Find in All Libraries… (Figure A-4). The Object Finder window will now appear (Figure A-5).

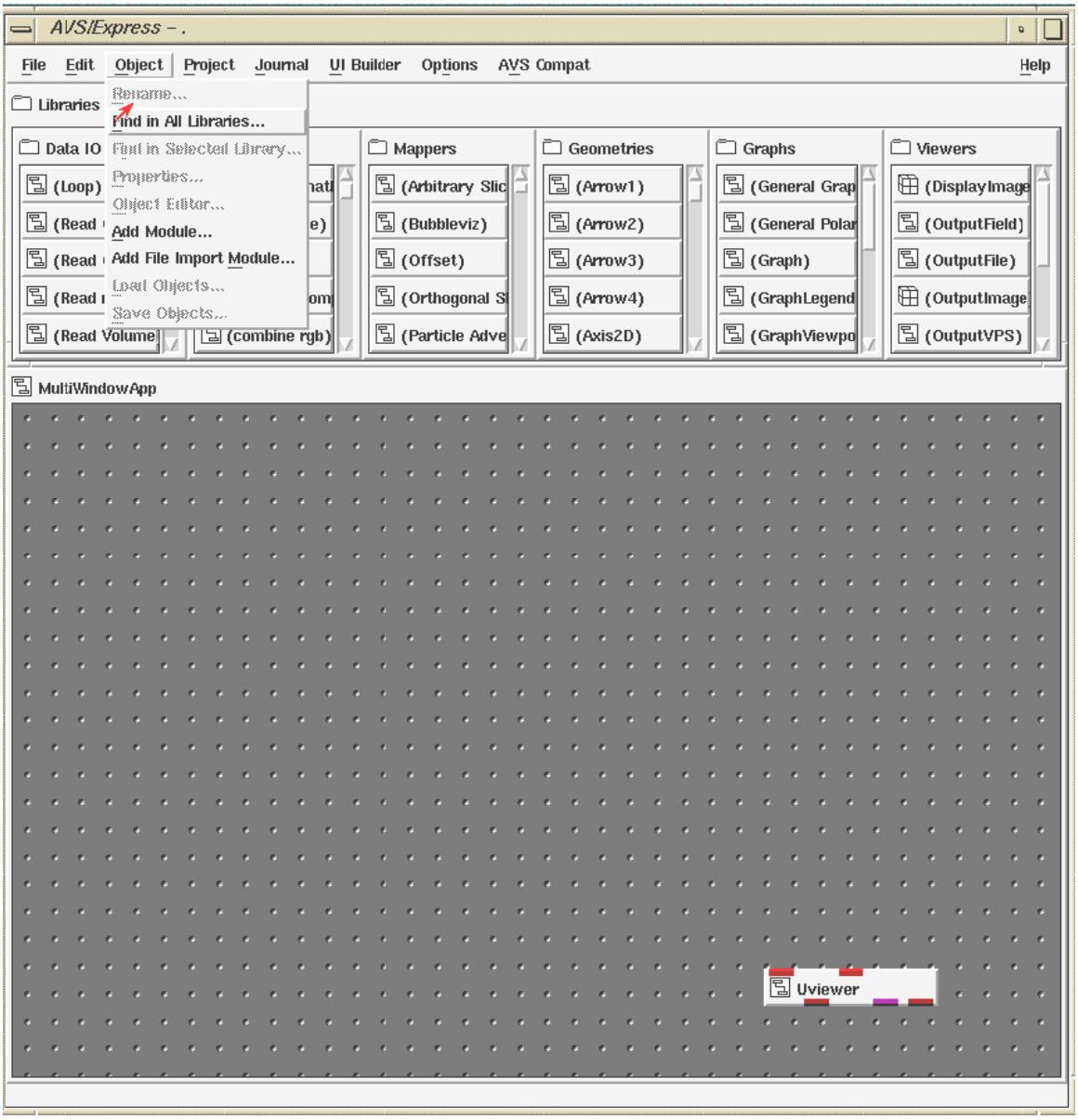

Figure A-4: Finding a module

The Object Finder window is shown in Figure A-5. Under Search Pattern, type the name of the module, e.g. *bounds*, and click on Find. This will give, under Objects, the library location of that module. For example, the **bounds** module is found in the Mappers library. Clicking on Show will highlight that module in the Network Editor libraries.

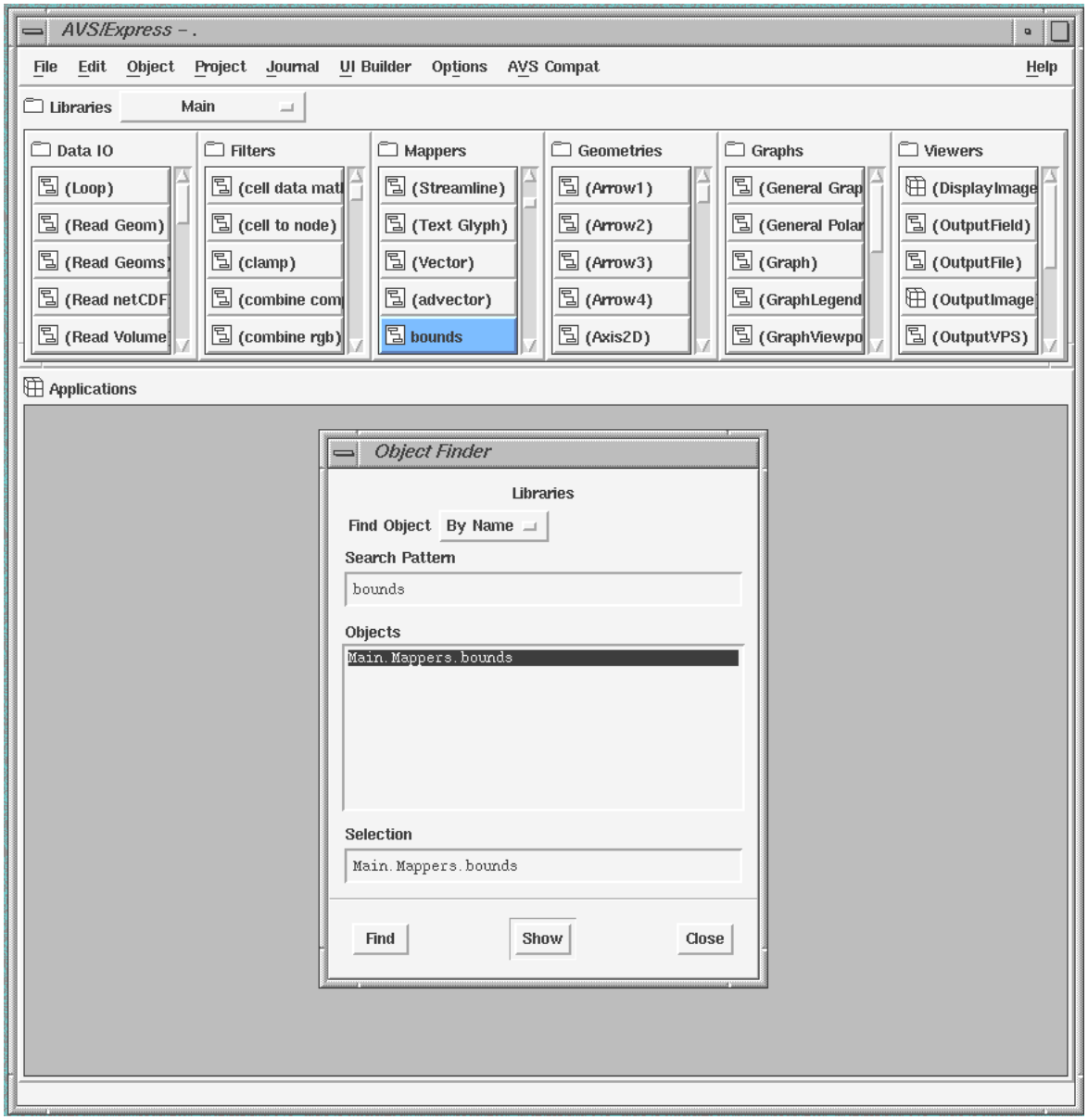

Figure A-5: Object Finder

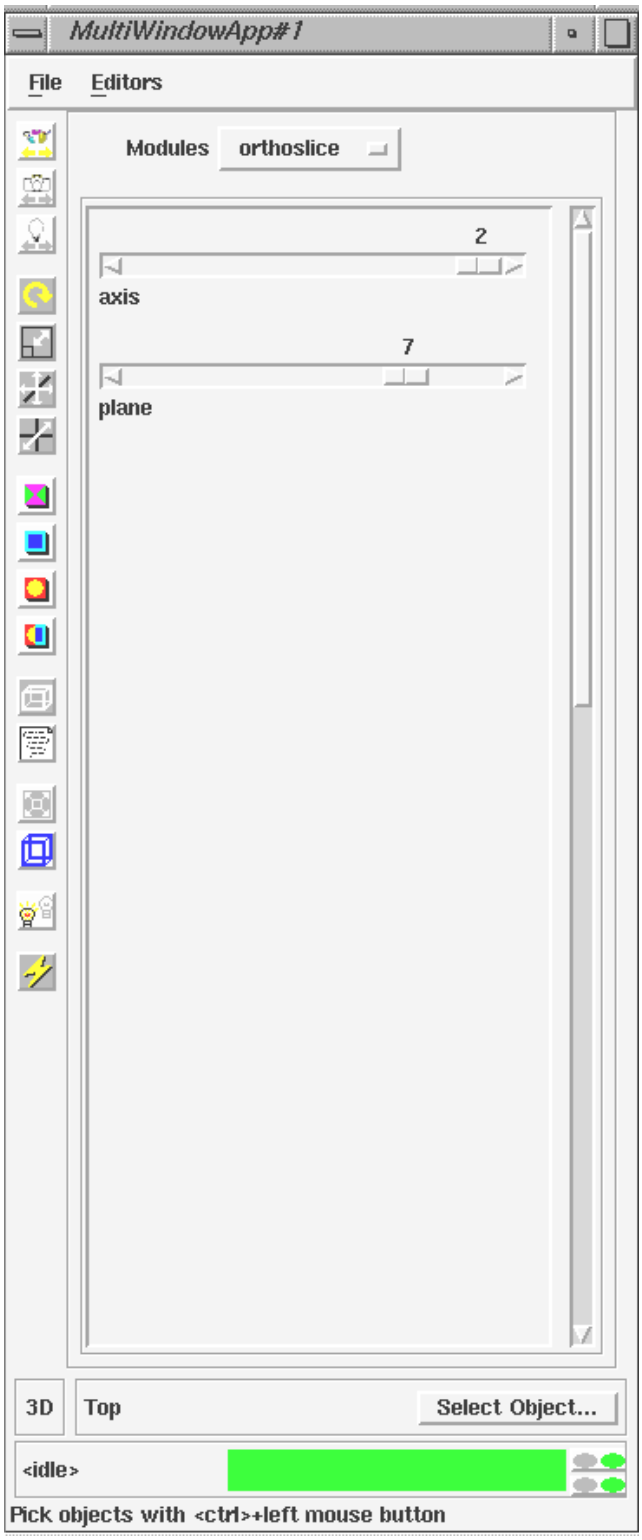

Figure A-6: Parameters of **orthoslice**

Figure A-7 shows the parameters of the **crop** module, which can be altered by moving the sliding bars.

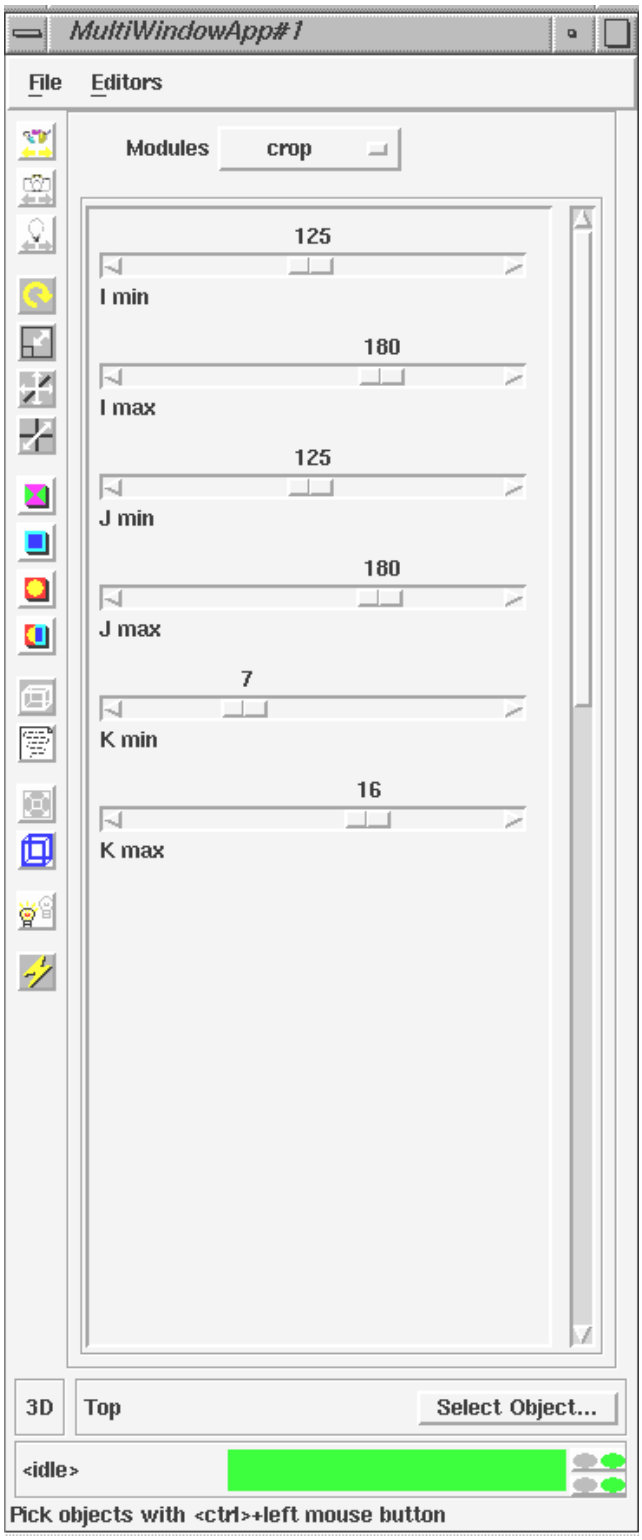

Figure A-7: Parameters of **crop**

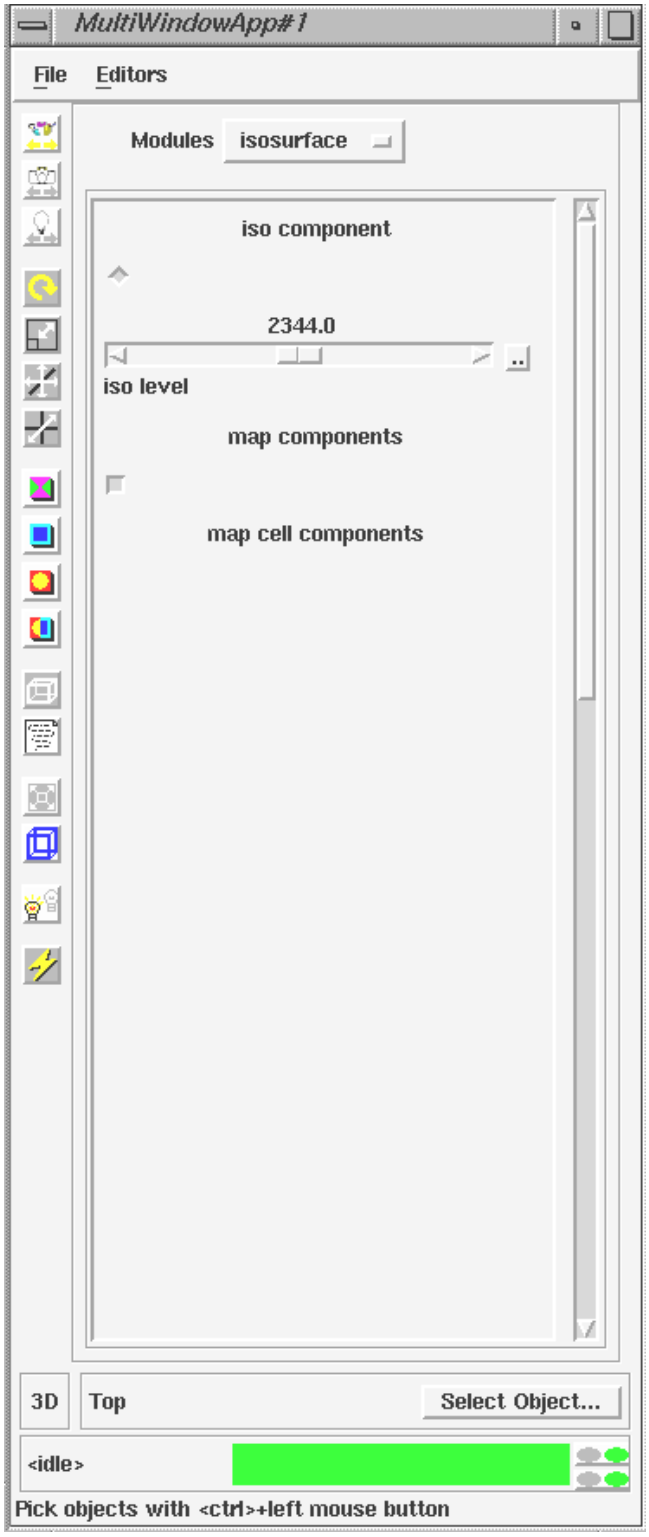

Figure A-8: Parameters of **isosurface**

Figure A-9: Object Selector

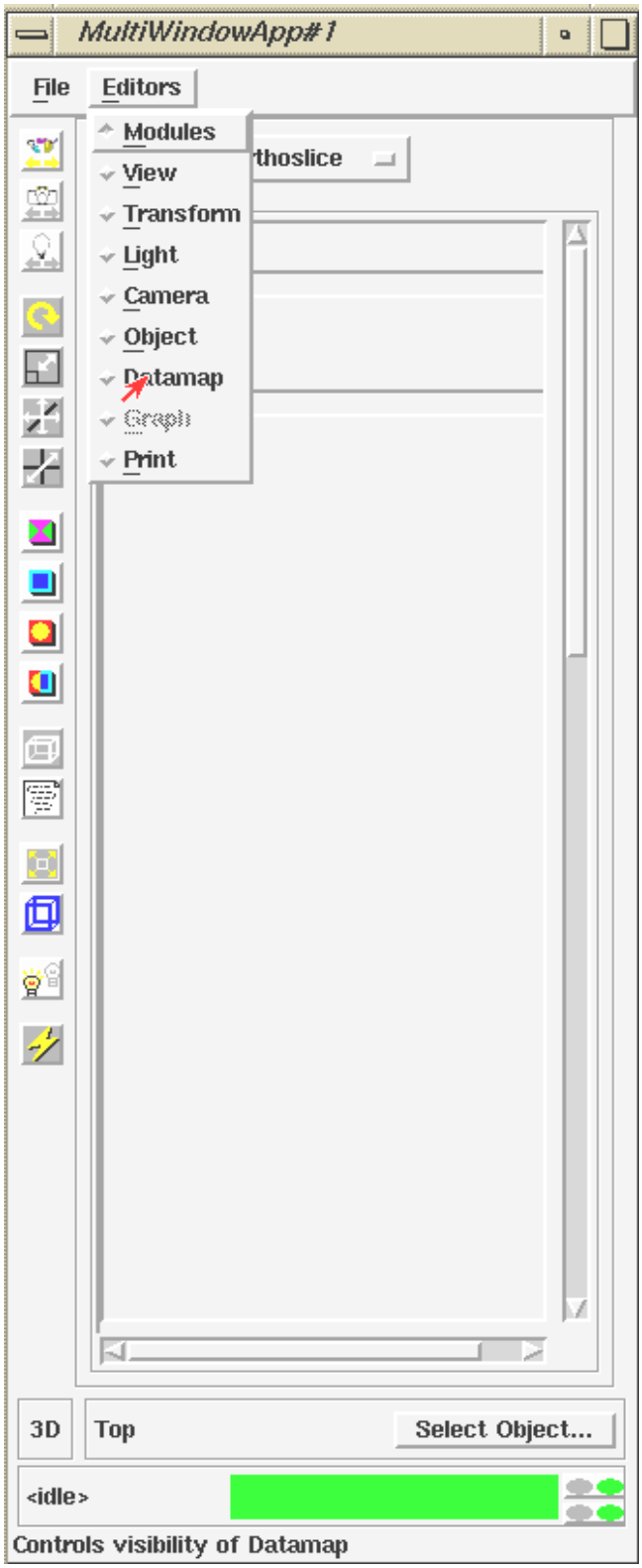

Figure A-10: Editors menu

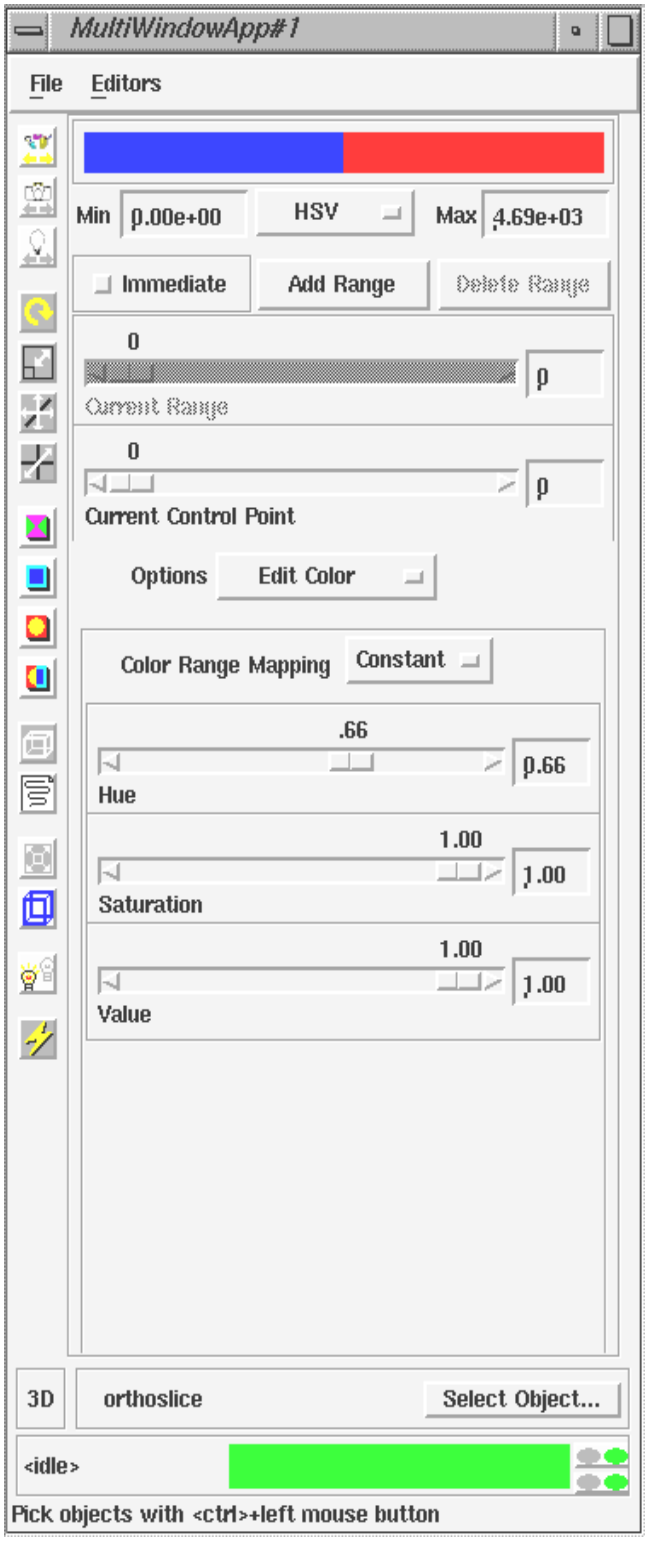

Figure A-11: Datamap Editor

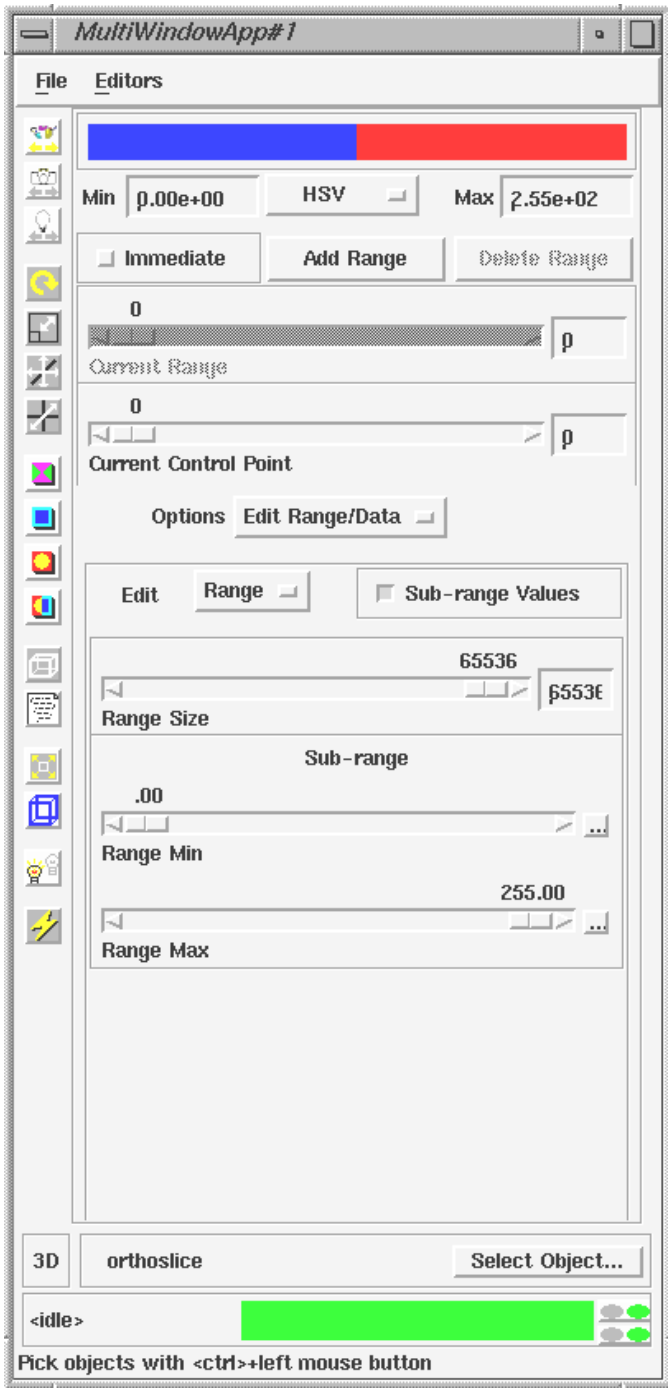

Figure A-12: Changing the Datamap

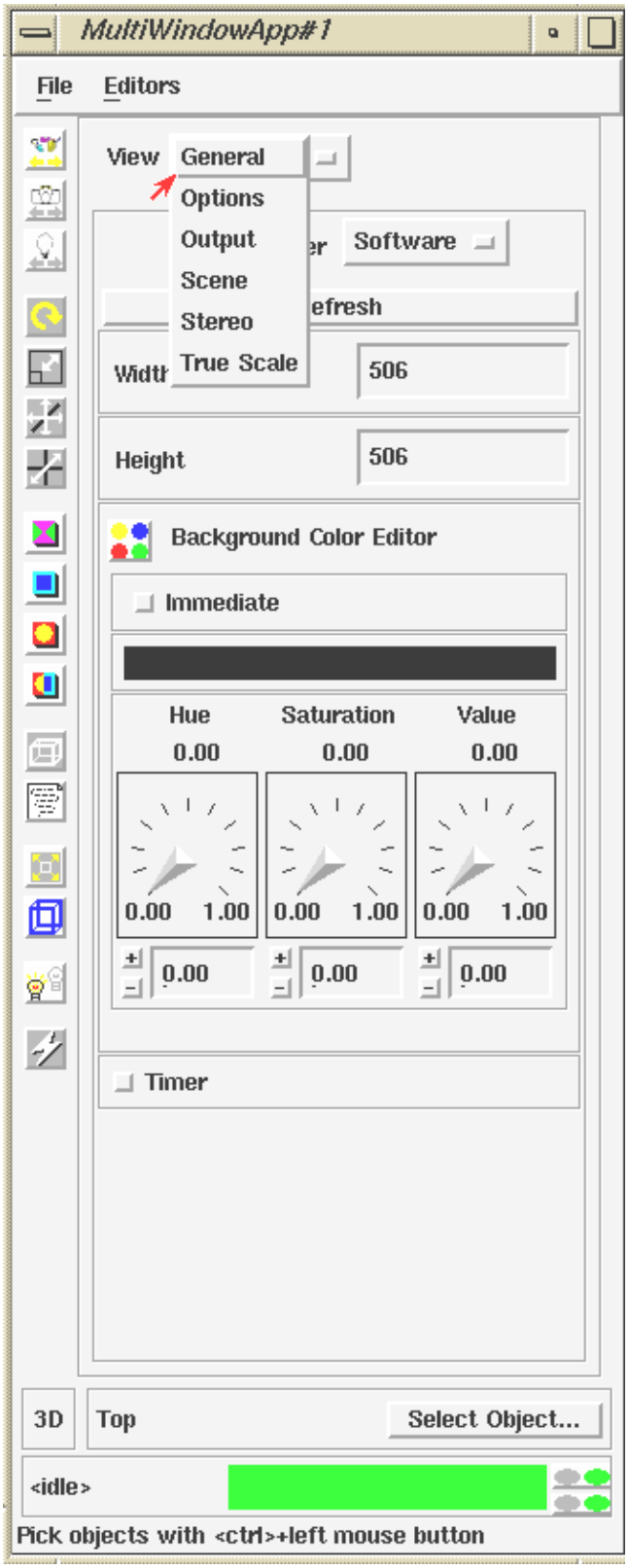

Figure A-13: Changing the View

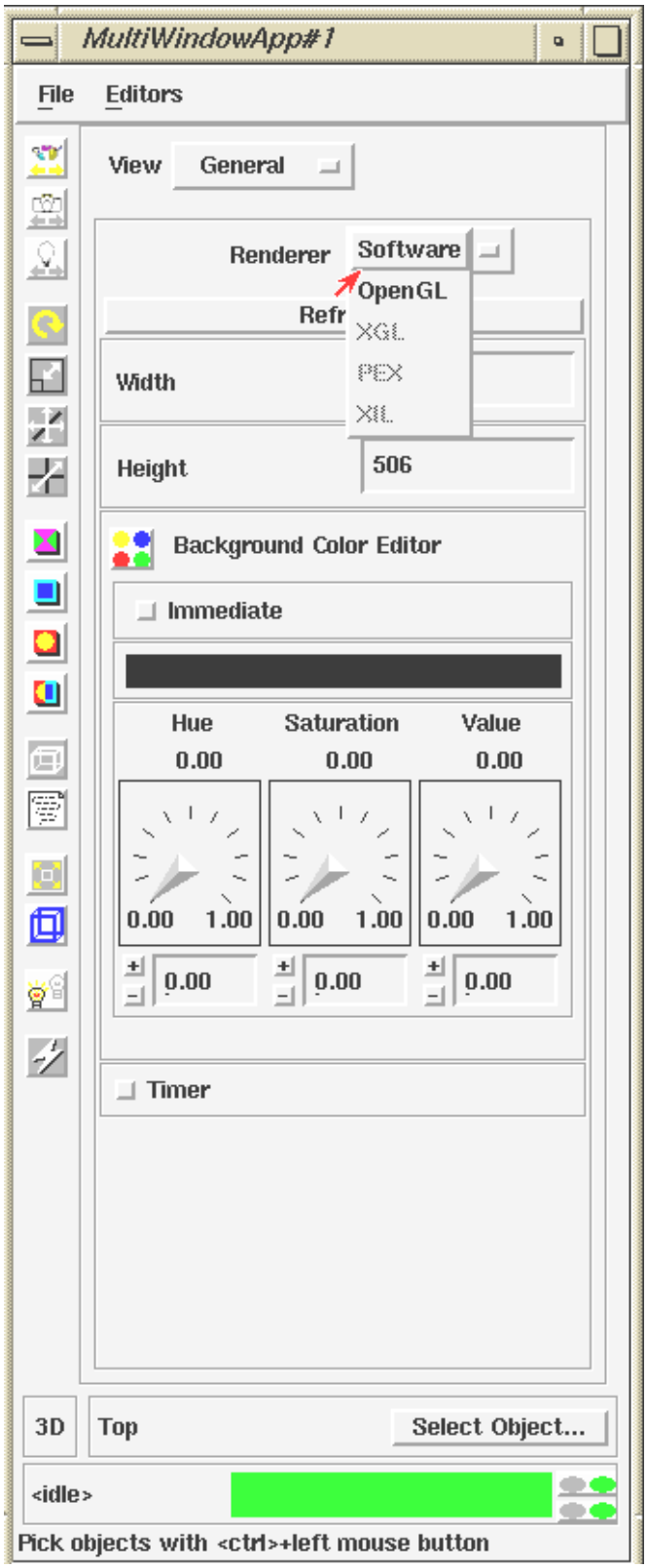

Figure A-14: Changing the View

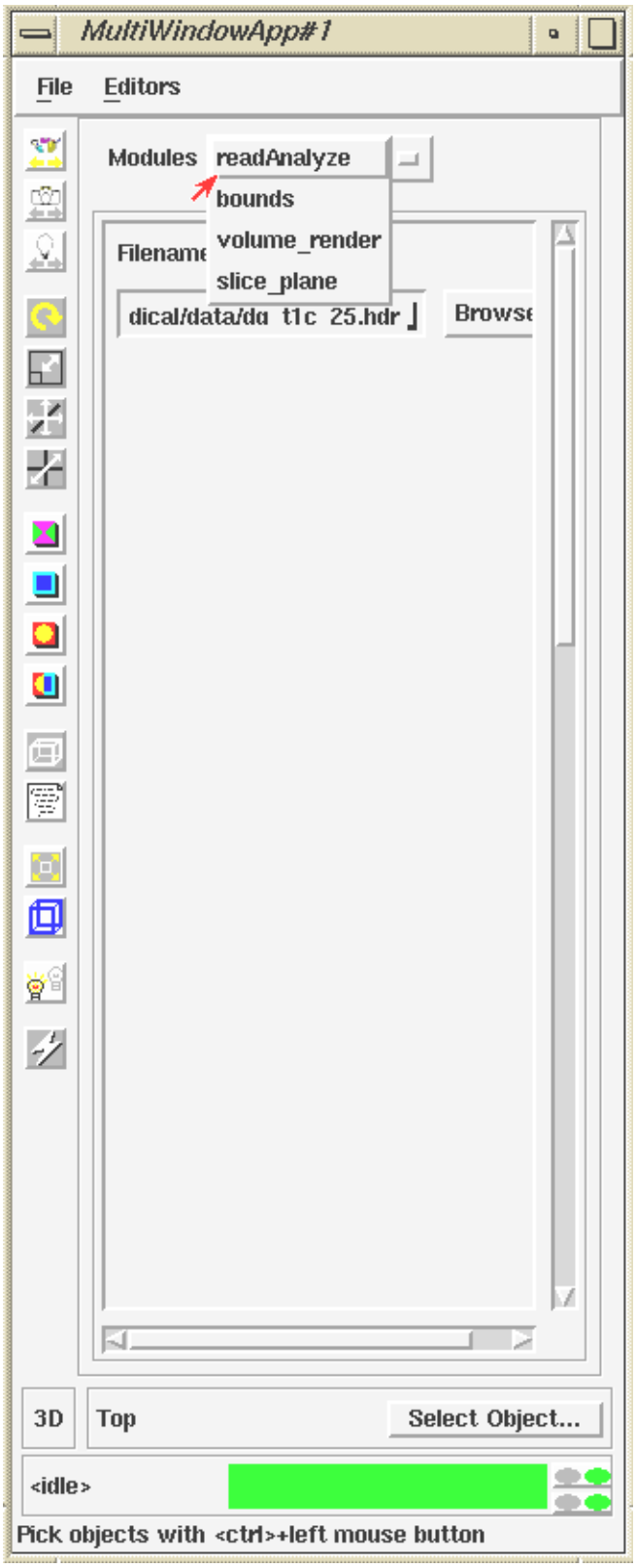

Figure A-15: **readAnalyze** module

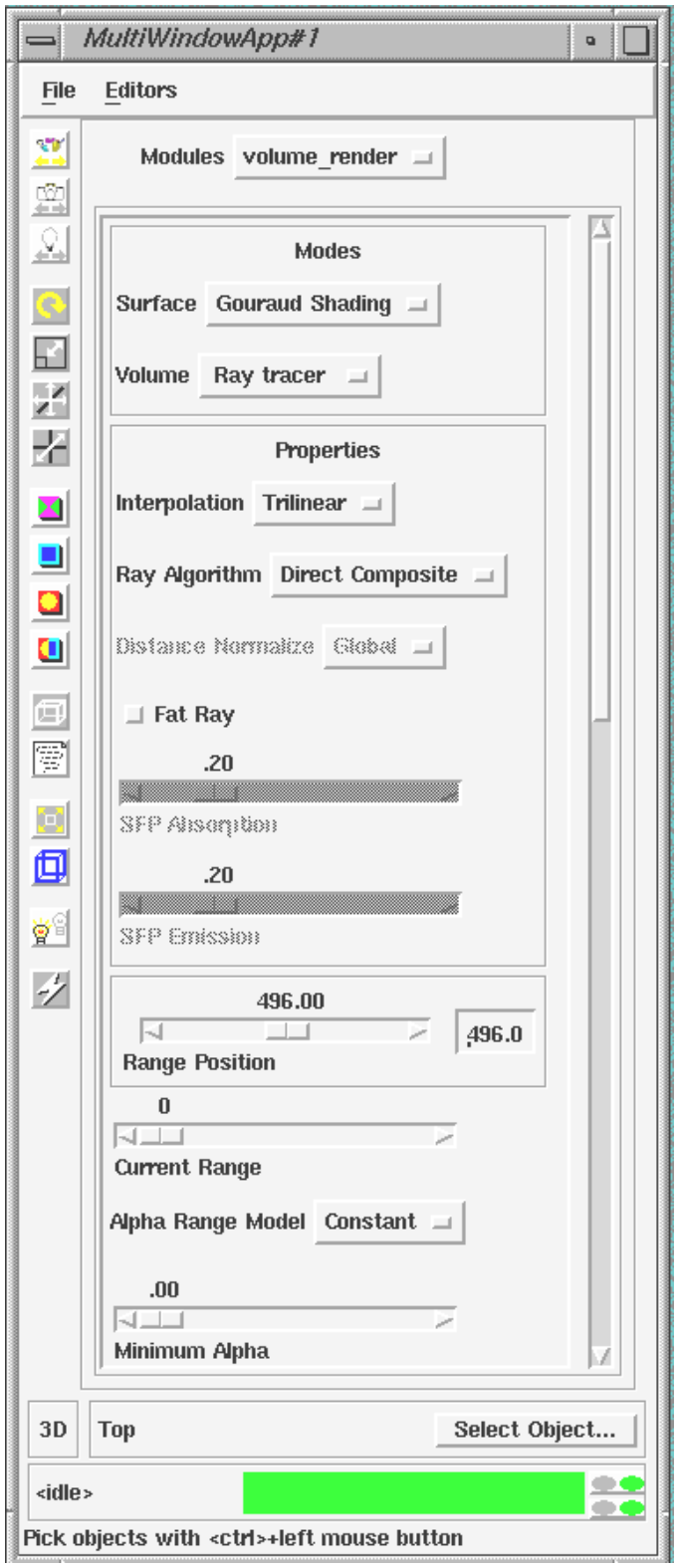

Figure A-16: Parameters of **volume_render**

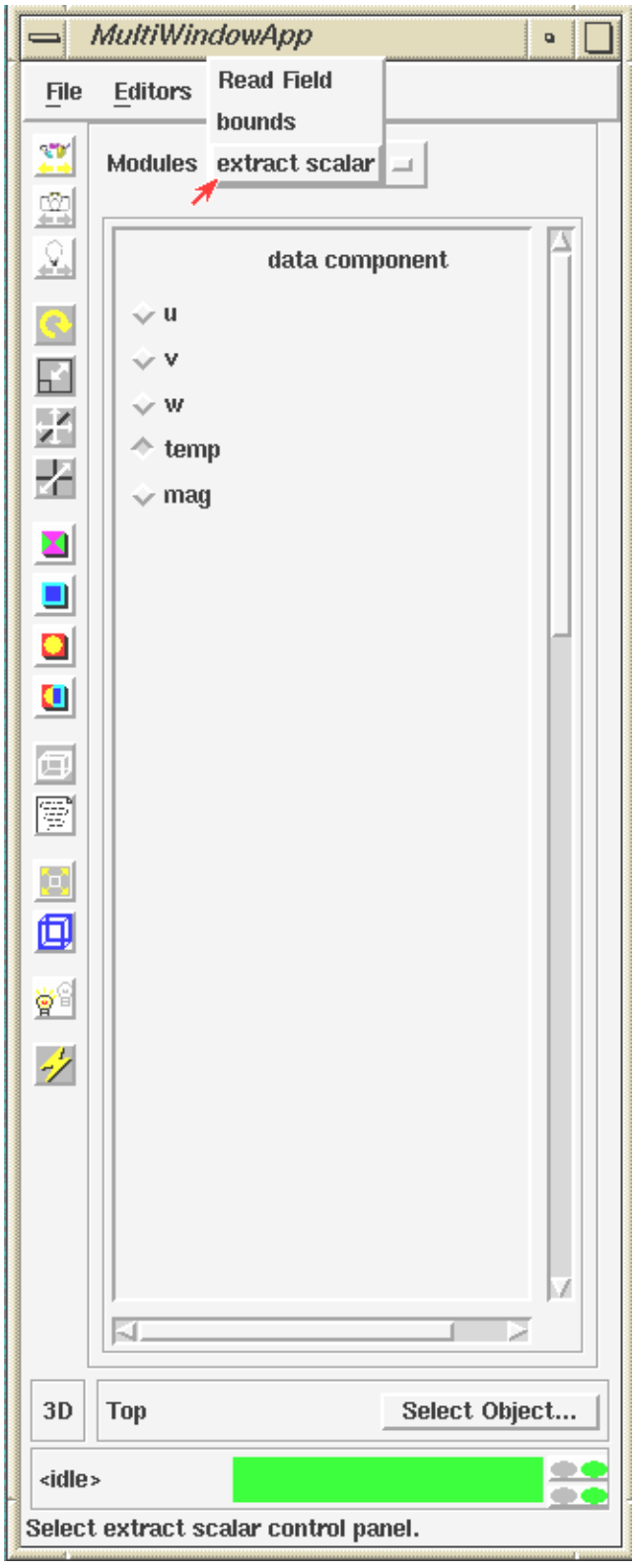

Figure A-17: Parameters of **extract_scalar**

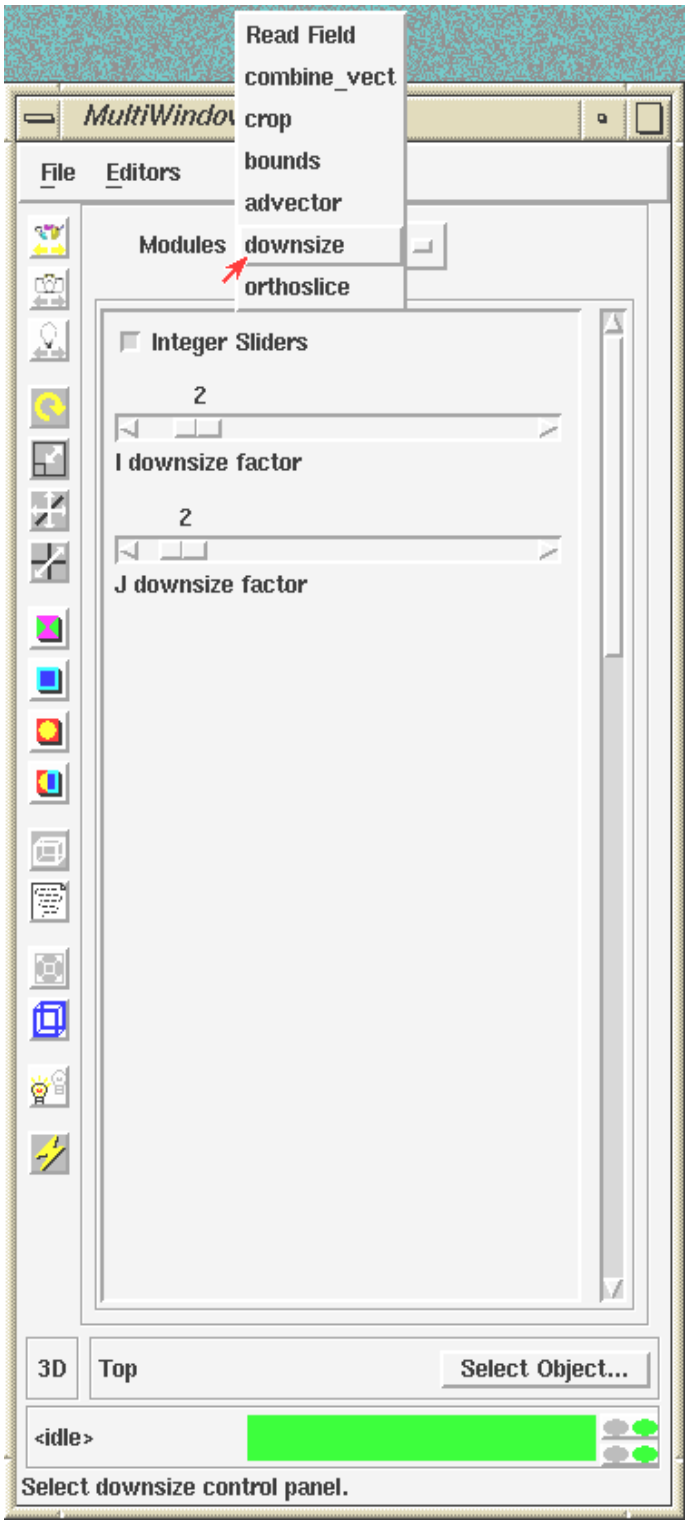

Figure A-18: Parameters of **downsize**

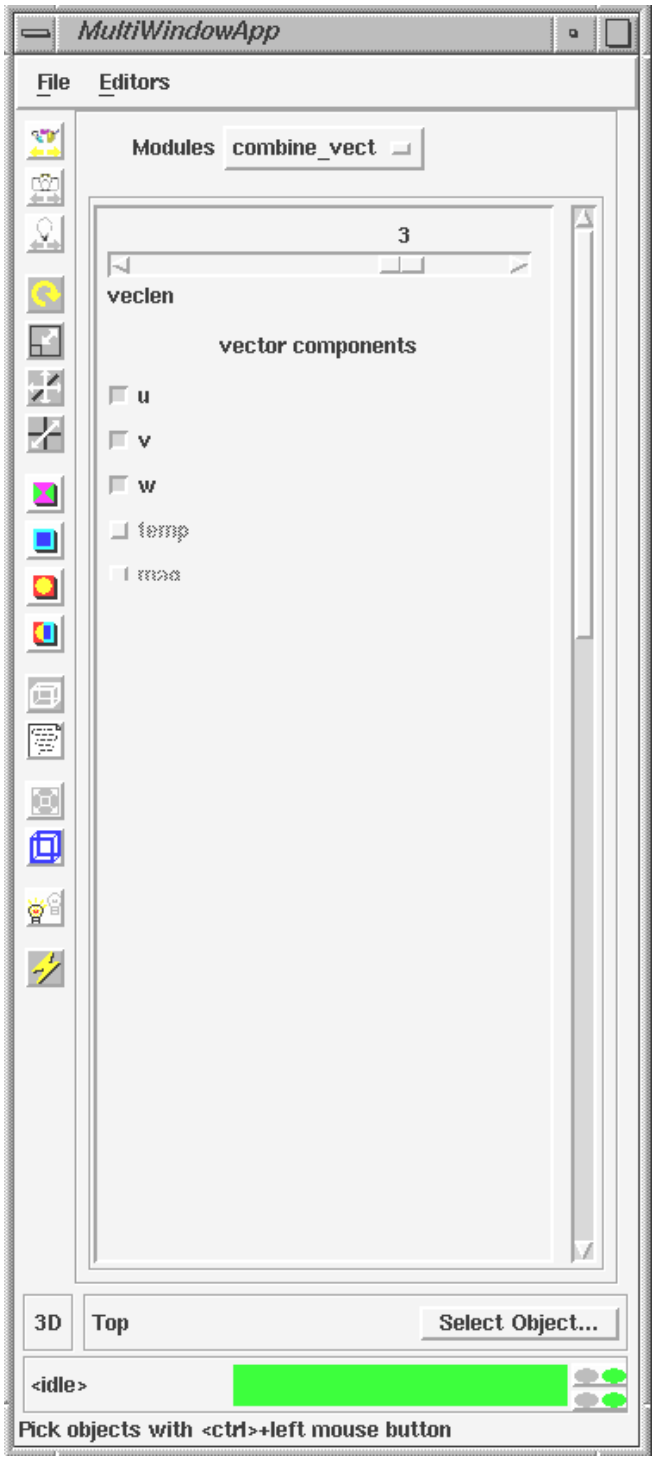

Figure A-19: Parameters of **combine_vect**

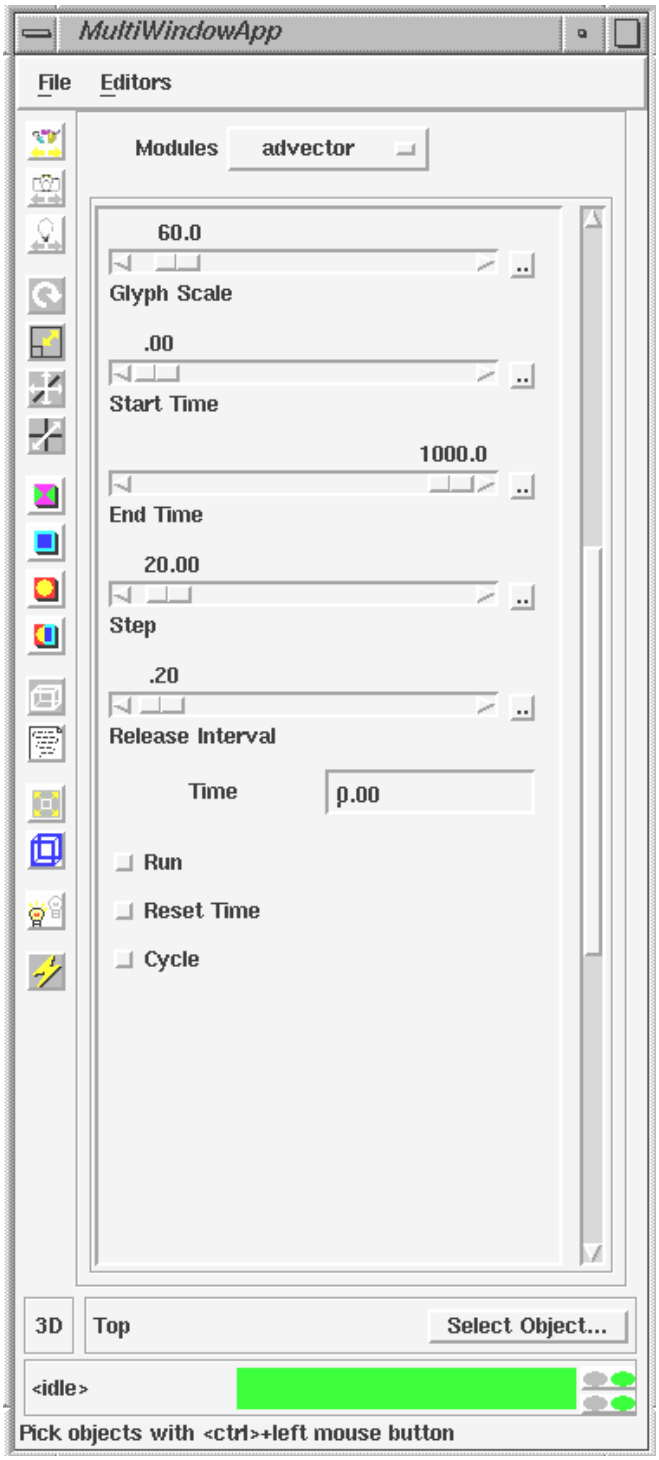

Figure A-20: Parameters of **advector**

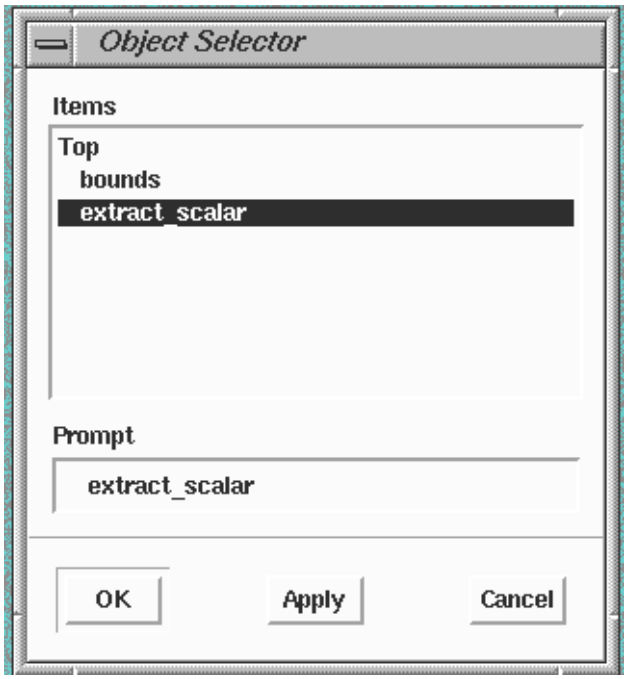

Figure A-21: Object Selector

Appendix B: Field Files

B.1: Field file for t1 data

Below is the field file for the data called t1, found in the *dg_t1_25.img* file:

```
# AVS field file
# medical data
ndim = 3       # 3 dimensions in computational space
dim1 = 256     # size of dimension 1
dim2 = 256
dim3 = 25
nspace = 3     # 3 dimensions in co-ordinate space
veclen = 1     # 1 data component per element
data = short   # type of data components
field = uniform # mapping method
label = t1
variable 1 file=dg_t1_25.img filetype=binary skip=0 stride=1
```

B.2: Field file for t2 data

Below is the field file for the data called t2, found in the *dg_t2_25.img* file:

```
# AVS field file
# medical data
ndim = 3       # 3 dimensions in computational space
dim1 = 256     # size of dimension 1
dim2 = 256
dim3 = 25
nspace = 3     # 3 dimensions in co-ordinate space
veclen = 1     # 1 data component per element
data = short   # type of data components
field = uniform # mapping method
label = t2
variable 1 file=dg_t2_25.img filetype=binary skip=0 stride=1
```

B.3: Field file for pd data

Below is the field file for the data called pd, found in the *dg_pd_25.img* file:

```
# AVS field file
# medical data
ndim = 3       # 3 dimensions in computational space
dim1 = 256     # size of dimension 1
dim2 = 256
dim3 = 25
nspace = 3     # 3 dimensions in co-ordinate space
veclen = 1     # 1 data component per element
data = short   # type of data components
field = uniform # mapping method
label = pd
variable 1 file=dg_pd_25.img filetype=binary skip=0 stride=1
```

B.4: Field file for ir data

Below is the field file for the data called ir, found in the *dg_ir_25.img* file:

```
# AVS field file
# medical data
ndim = 3       # 3 dimensions in computational space
dim1 = 256     # size of dimension 1
dim2 = 256
dim3 = 25
nspace = 3     # 3 dimensions in co-ordinate space
veclen = 1     # 1 data component per element
data = short   # type of data components
field = uniform # mapping method
label = ir
variable 1 file=dg_ir_25.img filetype=binary skip=0 stride=1
```

References

[1] The AVS website: www.avs.com

[2] AVS/Express Visualization Edition: An Introductory Course by F Lin, A Grant, R Slinger at Manchester and North HPC T&EC, J Irwin, H Morphet at International AVS Centre (Manchester)

[3] The UK AVS and UNIRAS User Group website: www.uauug.org

[4] The International AVS Centre website: www.iavsc.org

[5] Data Visualization Techniques: edited by Chandrajit Bajaj. John Wiley & Sons, 1999. ISBN 0 471 96356 9

[6] The Biomedical Imaging Resource at Mayo Clinic website: www.mayo.edu/bir

[7] "Gulf Stream": Microsoft® Encarta® Online Encyclopedia 2001 http://encarta.msn.com © 1997-2001