THE UNIVERSITY *of* MANCHESTER

Advanced AVS Course

Student Workbook

ADVISORY GROUP ON Computer Graphics

S Larkin, A J Grant, F Lin, N Hill

Manchester and North HPC T&EC

MAN T&EC
The Manchester and North HPC Training and Education Centre

Edition 6.0 December 1994

Document Editor

The original document was compiled and typeset by Steve Larkin, Computer Graphics Unit, Manchester Computing Centre using the document preparation system LaTeX.

Additional exercises and improvements to the original text were supplied by Naomi Hill. This work was undertaken whilst working in the Computer Graphics Unit as part of the AGOCG Summer Student Projects 1994.

The document is published and distributed by the Manchester and North High Performance Computing Training and Education Centre (MAN T&EC).

For further information of MAN T&EC see the following World Wide Web page:

http://www.hpctec.mcc.ac.uk/hpctec/

or send email to:

hpc-info@mcc.ac.uk

or contact Andrew Grant (a.j.grant@mcc.ac.uk) or by mail at:

MAN T&EC
Manchester Computing Centre
University of Manchester
Manchester M13 9PL
UK

Table of Contents

1 Introduction

1.1 The Advanced AVS Course

The Advanced AVS Course has been developed by the Manchester and North High Performance Computing Training and Education Centre (MAN T&EC), and is based on an earlier version produced by Manchester Computing Centre (MCC) for the UK Advisory Group on Computer Graphics (AGOCG).

This course is designed for the user with some basic experience of the Application Visualization System (AVS) including knowledge of the network editor to create networks of modules. The course covers extensively the topic of creating AVS modules to incorporate your own application code into the system. This can range from embedding simulation codes to implementing visualisation techniques and each class of module is covered in the course. The C programming language is used for the practical exercises. Prerequisites are: Basic Unix and C programming experience. The Introductory AVS Course, which is also developed by MAN T&EC, fulfils the prerequisite for necessary AVS experience.

This document contains all the practical exercises and solutions for the Advanced AVS Course.

1.2 Before starting the exercises

The advanced course assumes the source code for both the questions and answers to these exercises has been made available under the directory `/usr/avs/auxdata/advanced`. Copies of the source code can be obtained via anonymous FTP from the University of Manchester (ftp.mcc.ac.uk or IP address 130.88.203.12, `pub/mantec/avs/adv.tar.Z`).

You will need a local copy of these directories and your course tutor will explain how you do this if required.

You may now proceed with the examples detailed in this workbook.

2 do_threshold (ex1)

2.1 Description of the module

This module will import 3D arrays of data with a single scalar at each data element. The data will be processed and values which exceed the defined minimum and maximum range will be set to zero. This threshold range can be changed by altering either of the two dials which control the minimum and maximum values.

2.2 Specification of the module

The exercise will illustrate how to construct the `do_threshold` module described above using the AVS module generator. The concluding part of the exercise will involve loading the module into AVS and using it as part of a network. A specification of the module follows:

Module name:	do_threshold
Module class:	filter
Language:	C
Module type:	subroutine
Input port name 0:	input
Input port type 0:	"field 3D scalar uniform float"
Output port name 0:	output
Output port type 0:	"field 3D scalar uniform float"
Parameter name 0:	pmin
Parameter type 0:	real (dial) 0.0–1.0
Parameter default 0:	0.0
Parameter name 1:	pmax
Parameter type 1:	real (dial) 0.0–1.0
Parameter default 1:	1.0
File name:	`threshold.c`

2.3 The exercise

The following extract of code performs the threshold function and can be found in the file `ex1/threshold.body`:

```
int i, j, k;
float min = *pmin;
float max = *pmax;
```

```
/* allocate the output field to have same structure as input */

*output = (AVSfield_float *)AVSfield_alloc (input,0);

/*
 * We use a triply nested loop to traverse the field.  The macros MAXX,
 * MAXY, and MAXZ determine the maximum extent of the field in each of
 * the three dimensions.  We know this will be a 3-dimensional
 * field because of the declaration in the description routine, so
 * we don't need to check.  When we want to reference an element of the
 * field we use the I3D macro which picks an element of a 3D field.
 * Note that the first index (i) varies the fastest in memory, so we
 * make that the innermost loop.
 */

 for (k = 0; k < MAXZ(input); k++)
       for (j = 0; j < MAXY(input); j++)
            for (i = 0; i < MAXX(input); i++)
               if (I3D(input, i, j, k) > max) {
                     I3D(*output, i, j, k) = 0.0;
                    } else if (I3D(input, i, j, k) < min) {
                             I3D(*output, i, j, k) = 0.0;
                           } else {
                            I3D(*output, i, j, k) = I3D(input, i, j, k);
                           }
```

The following exercise shows the steps taken to integrate this code into an AVS module.

2.3.1 Starting the module generator

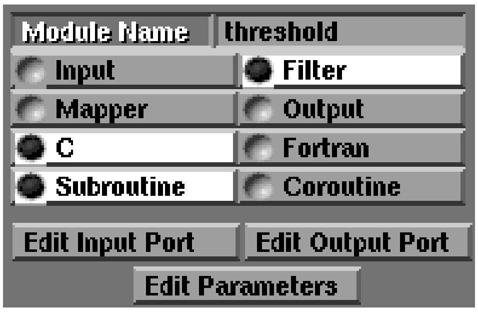
1. Start AVS by typing `avs`.

2. Click on the panel labelled "Network Editor" to open the AVS network editor.

3. Select the module `module generator` which can be found in the "Data Output" column by holding down the **left** mouse button and dragging the module onto the workspace.

   The module generator control panel should appear on the left hand side of the screen as shown below:

2.3.2 Defining the module characteristics

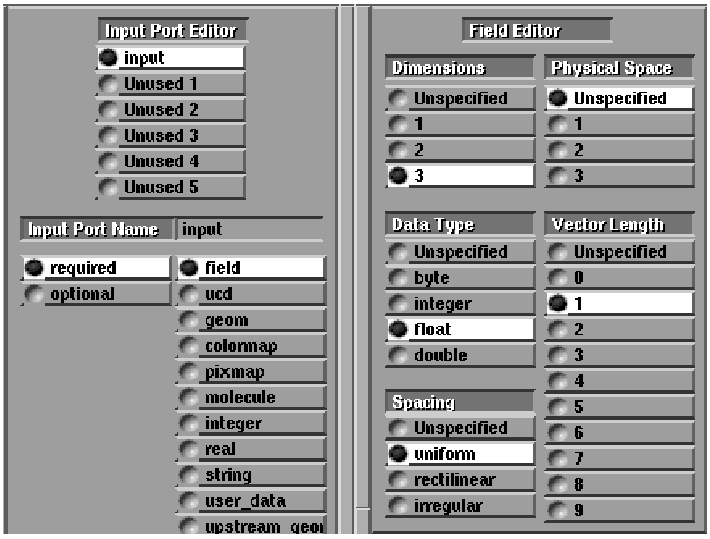


1. Enter the module's name `do_threshold` in the panel labelled "Module Name:".

2. Click on the button "Filter" to select the modules class.

3. Leave the module set as a "subroutine" and the language set as "C".

2.3.3 Defining the input port



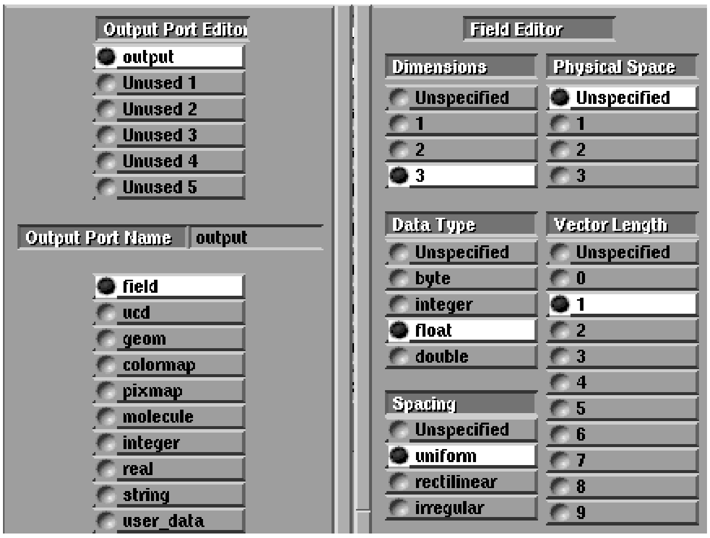
1. Click on the button labelled "Edit Input Port" and two windows should appear.

2. In the new left window change the text "Unused 0" to "input" and leave this set as "field".

3. In the new right window change the dimensions to "3", the data type to "float", the vector length to "1" and the spacing to "uniform". The physical dimensions should be left as "unspecified".

4. Again click on the panel labelled "Edit Input Port" and the two windows will disappear.

2.3.4 Defining the output port



1. Click on the button labelled "Edit Output Port" and two windows should appear.

2. In the new left window change the text "Unused 0" to "output" and leave this set as "field".

3. In the new right window change the dimensions to "3", the data type to "float", the vector length to "1" and the spacing to "uniform". The physical dimensions should be left as "unspecified".

4. Again click on the panel labelled "Edit Output Port" and the two windows will disappear.

2.3.5 Specifying parameters

1. Click on the button labelled "Edit Parameters" and a window should appear.

2. In the new window below the text browser change the text "Unused 0" to "pmin" and select the parameter type by clicking on the button labelled "real".

3. Located towards the bottom of this new window are the panels to specify the range and default value of the parameter. Set the range to 0.0 and 1.0 with a default setting of 0.0.

4. In the text browser in this window click on the text labelled "Unused 1".

5. In the panel below the browser change the text "Unused 1" to "pmax" and select the parameter type by clicking on the button labelled "real".

6. Located towards the bottom of this new window are the panels to specify the range and default value of the parameter. Set the range to 0.0 and 1.0 with a default setting of 1.0.

7. Click again on the button "Edit Parameters" and the new window should disappear.

2.3.6 Writing the source code and makefile

1. Using the file browser select the current directory to be `ex1`.
2. Click on the button labelled "New File" and specify the name `threshold.c`. Click on the "OK" button to confirm the choice.
3. Click on the buttons "Write Source" and "Write Makefile" to write the appropriate files.

2.3.7 Exiting from the module generator

The module generator can be used to edit, compile and load modules into AVS. For the purposes of this course we will also illustrate how this may be done independently of the module generator.

1. Drag the module "module generator" onto the hammer at the side of the network editor workspace (destroying the module).
2. Exit from the network editor and then AVS.

2.3.8 Compiling and Loading modules into AVS

In the directory `ex1` you should two new files `threshold.c` and `Makefile.threshold`. Rename the makefile `Makefile.threshold` to be called `makefile` e.g., type

```
mv Makefile.threshold makefile
```

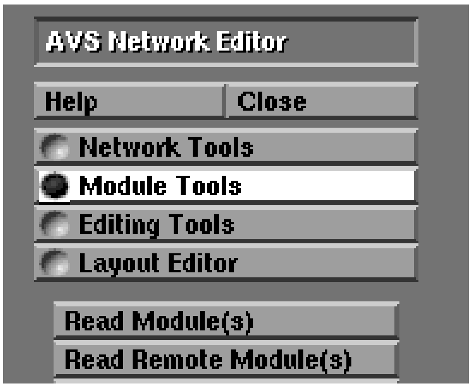
Now edit the file called `threshold.c` and locate the following lines in the routine `do_threshold_compute`:

```
/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */
/* <---- END OF USER-SUPPLIED CODE SECTION #3                            */
```

Insert between these comment lines the code in the file `ex1/threshold.body` and **save** the file. A full listing of the complete module `threshold.c` can be found in appendix A.

The next steps illustrate compiling and loading an AVS module and applies to most (if not all) modules you will use.

1. Compile the module by typing `make threshold`
2. Start AVS and open the "Network Editor".

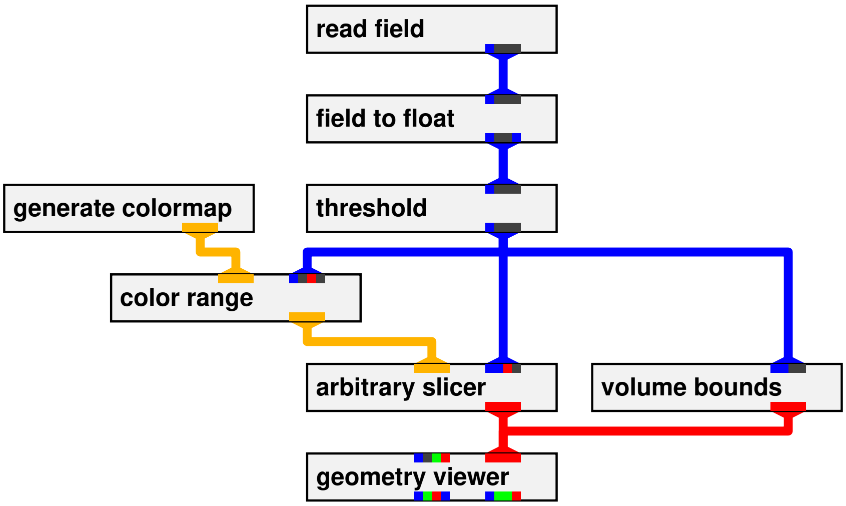


3. On the network editor control panel select the main menu option "Module Tools" and the sub-menu selection "Read Module(s)".
4. A file browser should appear. Click on the executable file, `threshold`, generated by compiling the module.
5. Scrolling through the module palette under the "Filters" column you will now find the module "threshold" which can be used like any other module in AVS.

2.3.9 Using the threshold module

The following network should be constructed to test that the `threshold` is functioning correctly. Before reading the data select the "normalise" button on the `field_to_float` module.

Try reading the data files `/usr/avs/data/field/hydrogen.fld` and `/usr/avs/data/field/lobster.fld`.



3 Read Temps (ex2)

3.1 Description of the module

A module must be written to read an ASCII data file which contains data on a regular 2D grid. The data component at each element is a single floating point value and the first two numbers in the file indicate the X and Y dimensions of the grid respectively.

An extract from the data file is shown below:

```
31 31
  213.0
  213.5
  210.1
  .....
```

3.2 Specification of the module

Using the module generator create the following module and appropriate makefile:

Module name:	read_temps
Module class:	data input
Language:	C
Module type:	subroutine
Output port name 0:	temp_field
Output port type 0:	"field 2D 2-space 1-vector uniform float"
Parameter name 0:	datafile
Parameter type 0:	string
Parameter file type 0:	.dat (remember delete the ":" already there)
Parameter file name 0:	leave blank
File name:	`read_temps.c`

3.3 The exercise

1. Locate the line in the source code file which appears similar to the one below and change it to read the same:

```
param = AVSadd_parameter("datafile","string",0,0,".dat");
```

2. Include the file `ex2/read_temps.body` in the user defined code area of the `read_temps_compute` routine and fill in the appropriate code where indicated.

The source code for the `read_temps.body` file is shown below and a full listing of the answer can be found in appendix B.

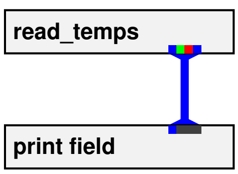
3.4 Source Code (`read_temps.body`)

```
  FILE *file_ptr;
  int  dims[2];
  int  i,j;
  float datavalue;

  /* check to see if a datafile has been supplied */

################################ INSERT CODE HERE #######################################

  /* check the datafile is readable */

################################ INSERT CODE HERE #######################################

  /* read in the dimensions of the field */

################################ INSERT CODE HERE #######################################

  /* check to see if the field has been previously allocated */

################################ INSERT CODE HERE #######################################

  /* allocate enough space for the output field */

################################ INSERT CODE HERE #######################################

  /* read the data from the file and add to the field */

################################ INSERT CODE HERE #######################################
```

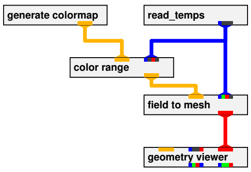
3.5 Testing the module

The following networks will be useful for testing your module executes correctly:

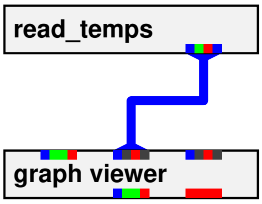
Print Field



Field to Mesh



Graph Viewer



4 Read Conc (ex3)

4.1 Description of the module

A module must be written to read an ASCII data file which contains scattered data in 3D space. The data component at each element is a single floating point value and has an associated x,y and z coordinate.

The first number in the file indicates the number of data points.

An extract from the data file is shown below:

```
7831
21.0 19.0 1.0  .01
21.0 17.0 1.0  .02
21.0 15.0 2.0  .02
.... .... ...  ...
```

4.2 Specification of the module

Using the module generator create the following module and appropriate makefile:

Module name:	read_conc
Module class:	data input
Language:	C
Module type:	subroutine
Output port name 0:	conc_field
Output port type 0:	"field 1D 3-space 1-vector irregular float"
Parameter name 0:	datafile
Parameter type 0:	string
Parameter default 0:	".dat"
File name:	`read_conc.c`

4.3 The exercise

1. Locate the line in the source code file which appears similar to the one below and change it to read the same:

```
param = AVSadd_parameter("datafile","string",0,0,".dat");
```

2. Include the file `ex3/read_conc.body` in the user defined code area of the `read_conc_compute` routine and fill in the appropriate code where indicated.

The source code for the `read_conc.body` file is shown below and a full listing of the answer can be found in appendix C.

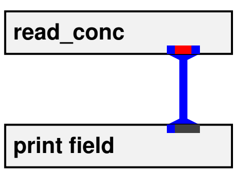
4.4 Source Code (read_conc.body)

```
  FILE *file_ptr;
  int  dims[1];
  int  i,j;
  float xpos,ypos,zpos,datavalue;
  float *coords;

  /* check to see if a datafile has been supplied */

################################ INSERT CODE HERE #######################################

  /* check the datafile is readable */

################################ INSERT CODE HERE #######################################

  /* read in the dimension of the field */

################################ INSERT CODE HERE #######################################

  /* check to see if the field has been previously allocated */

################################ INSERT CODE HERE #######################################

  /* allocate enough space for the output field */

################################ INSERT CODE HERE #######################################

  /* read the data from the file and add to the field */

################################ INSERT CODE HERE #######################################
```

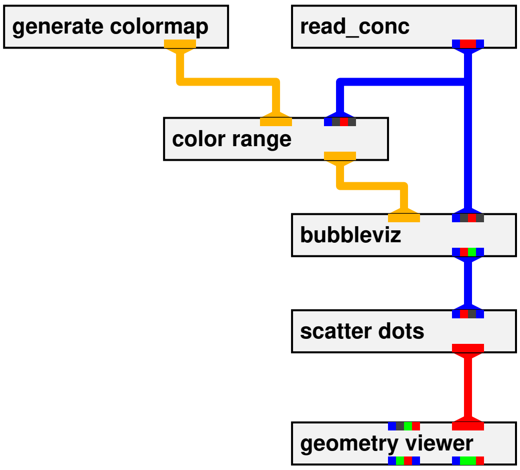
4.5 Testing the module

The following networks will be useful for testing your module executes correctly:

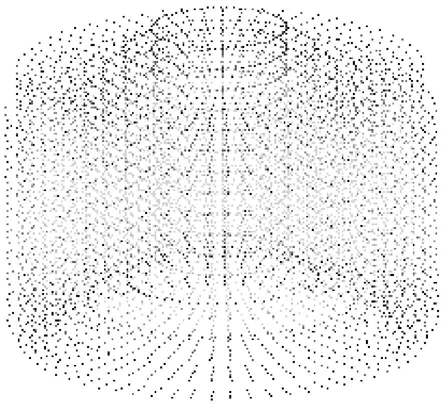
Print Field



Scatter Dots



If the module is functioning correctly then the `geometry_viewer` should show a display similar to the one below:



5 Scatter Plot (ex4)

5.1 Description of the module

A module must be written which accepts a list of scattered data in 3D space with each data element being a single scalar float value. This is the same format that the module `read_conc` reads in the previous exercise. You will find a file `conc.fld` which describes this data file to the `read_field` module.

The module should process the data values and output an AVS GEOM edit list which creates a sphere for each data value. The sphere's colour should be controlled by the data value and the radius by a float parameter.

5.2 Specification of the module

Using the module generator create the following module and appropriate makefile:

Module name:	scatter_plot
Module class:	mapper
Language:	C
Module type:	subroutine
Input port name 0:	scatter_data
Input port type 0:	"field 1D 3-space 1-vector irregular float"
Output port name 0:	scatter_obj
Output port type 0:	GEOM (spheres)
Parameter name 0:	pradii
Parameter type 0:	real 0.0 – 0.2 (dial)
Parameter default 0:	0.0
File name:	`scatter_plot.c`

5.3 The exercise

1. Include the file `ex4/scatter_plot.body` in the user defined code area of the `scatter_plot_compute` routine and fill in the appropriate code where indicated.

The source code for the `scatter_plot.body` file is shown below and a full listing of the answer can be found in appendix D.

5.4 Source Code (scatter_plot.body)

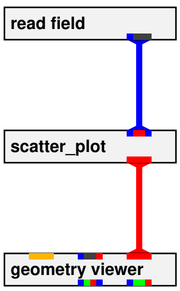
```
  GEOMobj *spheres;
  int dims[1],i;
  float *centres,*coords,*ptr_centres,*ptr_colours,*colours,
        *radii,*ptr_radii;

  /* get the number of points */

  AVSfield_get_dimensions (scatter_data,dims);

  /* allocate enough space for each sphere */

  centres = (float *)malloc(sizeof(float)*dims[0]*3);
  colours = (float *)malloc(sizeof(float)*dims[0]*3);
  radii   = (float *)malloc(sizeof(float)*dims[0]);

  /* get the coords for each sphere */

  coords = scatter_data->points;
  ptr_centres = centres;
  ptr_colours = colours;
  ptr_radii   = radii;

  for (i=0; i < dims[0] ; i++)
    {
      *radii++ = *pradii; /* radius from dial parameter */

      *centres++ = coords[i];
      *centres++ = coords[i + dims[0]];
      *centres++ = coords[i + (2*dims[0])];

      if ( *I1DV(scatter_data,i) < 1.0 )
{
  *colours++ = *I1DV(scatter_data,i); /* red   */
  *colours++ = *I1DV(scatter_data,i); /* green */
  *colours++ = 0.0; /* blue  */
}
      else
{
  *colours++ = 1.0;
  *colours++ = 1.0;
  *colours++ = 0.0;
}
    }

  centres = ptr_centres;
  colours = ptr_colours;
  radii   = ptr_radii;

  /* create the edit list */

################################ INSERT CODE HERE #######################################

  /* create the list of spheres */

  if ( *pradii < 0.0 )
```

```
    {
      /* if radii is 0.0 then create "real" dots */

################################ INSERT CODE HERE #######################################

    }
  else
    {

################################ INSERT CODE HERE #######################################

    }

  GEOMadd_float_colors (spheres,colours,dims[0],GEOM_COPY_DATA);

  GEOMauto_transform (spheres);

  /* now add spheres to the edit list */

################################ INSERT CODE HERE #######################################

  /* free up the spheres */

################################ INSERT CODE HERE #######################################

  /* free the allocate arrays */

  free (centres);
  free (colours);
  free (radii);
```

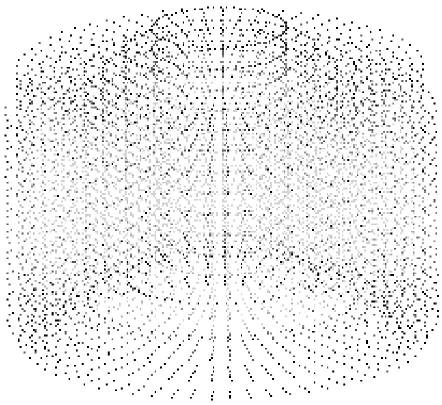
5.5 Testing the module

The following network will be useful for testing your module executes correctly. You should select the file `ex4/conc.fld` using the `read_field` file browser to test the module.

Reading the datafile



If the module executes correctly then the `geometry_viewer` should show something similar to the figure below:



5.6 Additional exercise

The module will only alter the red and green colour component associated with each sphere if the data value is less than 1.0 otherwise the data value is ignored.

This exercise involves altering the section of code which assigns the colour to each sphere to instead:

1. Inquire the minimum and maximum extent of the data using the AVS routine `AVSfield_get_minmax`.
2. Calculate the mid-value of the data range.
3. Assign blue to the sphere if the data value is less than the mid-value else assign red to the sphere.

6 Read Bricks (ex5)

6.1 Description of `read_bricks`

The data used is based on the results from calculations produced by FLOW3D[1]. The ASCII data format is an arbitrary one for the purposes of this course.

The brick data format is described below:

Nodes

The nodes which make up the structure have a position in 3D space and the following data associated with each node:

- Velocity: consists of 3 values (X_{vel}, Y_{vel}, Z_{vel})
- Pressure: scalar real
- Density: scalar real
- Viscosity: scalar real
- Kinetic energy: scalar real

Cells

The cells in the structure all hexahedral in shape and therefore consist of eight nodes.

6.2 Importing the data

The data sets are contained in ordinary ASCII files and the format is shown below:

```
numnodes numcells

nodeid X Y Z coordinates                                  (node positions)

cellid <list of nodes (8 in total)>                       (cell construction)

nodeid <velocity> <pressure> <density> <viscosity> <KE> (node data)
```

To import the data into AVS a module was written to read the ASCII file format and map this onto the UCD data type in AVS. The module processes the file and makes specific calls to the AVS UCD library which creates the equivalent structure using the AVS UCD data format. The module has one output port of the type UCD and this allows the module to introduce the data into an AVS network by sending the internal UCD structure created.

[1] Developed by Harwell Laboratories, Oxford, UK

6.3 Source Code (read_bricks)

```
/*

   Module Name : Read Bricks

   Author : Steve Larkin
            Computer Graphics Unit
            Manchester Computing Centre
            University of Manchester
            Oxford Road
            Manchester M13 9PL

            Tel : 061 275 6095
            Fax : 061 275 6040

   Date : March 1993

   Description : reads a file containing a number of 8 noded cells
                 with data components at each node.


*/

/* DEBUG           used to switch on the printf statements           */
/* TEMP_STRING_LEN max length of path name this module can parse     */
/* NODE_DATA_COMPS number of data components                         */
/* NODE_DATA_VALS  number of data values                             */

#define DEBUG           0
#define TEMP_STRING_LEN 100
#define NODE_DATA_COMPS 5
#define NODE_DATA_VALS  7

#include <stdio.h>
#include <malloc.h>
#include <avs/avs.h>
#include <avs/field.h>
#include <avs/ucd_defs.h>

int read_bricks_compute(out_UCDstruct, data_file)
/*

  main routine to create UCD structure from
  neutral file format.

*/

UCD_structure **out_UCDstruct;    /* contains final model */
char *data_file;                  /* name of file to read */

{
 FILE *dfp;                    /* file pointer to data_file */

 int  i,j,k,l,                    /* used for loops in general */
      numcells,                   /* number of cells in UCD struct */
      numnodes,                   /* number of nodes in UCD struct */
```

```
      nummats,                    /* number of material types per node */
      node_veclen,                /* data components per node */
      cell_veclen,                /* data components per cell */
      data_veclen,                /* data components per model */
      node_clistsz,               /* expected size of connecty list */
      cell_clistsz,               /* expected size of connecty list */
      util_flag,                  /* utility flag */
      ucd_flags,                  /* indicates what is labelled and type */
      cellid,                     /* used to read cell ids */
      tstep,                      /* used to read time step number */
      num_components,             /* num of data components at each node   */
      max_node_ref,               /* used to find max num a node is ref'd */
      mod_percent,                /* used to display status */
      comp_list[NODE_DATA_COMPS], /* used for dims of each data component */
      *node_mat_types,           /* used to store node material types */
      nodeid,                    /* used to read each node id (not used) */
      active_list[NODE_DATA_COMPS],    /* used to set active data component */
      node_list[8],              /* used to read and set cell information */
      cell_type;                 /* specify cell type */

 float tvalue,                    /* used to read in the time step value */
       min_node_data[NODE_DATA_COMPS],
       max_node_data[NODE_DATA_COMPS], /* store the min/max node data values */
       min_extent[3],             /* store the UCD structures min extent */
       max_extent[3],             /* store the UCD structures max extent */
       *node_x_coords,
       *node_y_coords,
       *node_z_coords,            /* used to read in node x y z coordinates */
       *node_data;                /* used to store components of node data */

 char *model_name,                /* label the whole UCD structure */
      *data_units,                /* used to store data component units */
      *data_labels,               /* used to store data component labels */
      temp_string[TEMP_STRING_LEN]; /* used to create model name */

  /* check filename supplied */

  if (!data_file)
   return(1);

  AVSmodule_status("Initialising",10);

#if DEBUG
  printf("Opening the data_file %s\n",data_file);
#endif

  /* open the data_file */

  if ( (dfp = fopen(data_file, "r")) == NULL )
  {
   AVSerror (" Error in read_bricks: cannot open file.\n Name : %s\n",
              data_file);
   return(0);
  }

#if DEBUG
  printf("Reading the header information\n");
```

```
#endif

  /* read in number of cells, nodes and material types */

  if ( fscanf (dfp,"%d%d%d\n", &numcells,&numnodes,&nummats) != 3 )
  {
   AVSerror (" Error in read_bricks: cannot read the header\n");
   return(0);
  }

#if DEBUG
  printf("No of cells : %d\n",numcells);
  printf("No of nodes : %d\n",numnodes);
  printf("No of mats  : %d\n",nummats);
#endif

  /* set up values which are needed but not specified by the
     data file being read */

  /* data values per node */

  node_veclen = NODE_DATA_VALS;

  /* 0 data values per cell */

  cell_veclen = 0;

  /* 0 data values per model */

  data_veclen = 0;

  /* connectivity list sizes (approx 8*no of cells)
     this assumes no mid-edge data and hexahedra
     shaped cells (8 nodes per cell)                */

  cell_clistsz = (numcells*8);

  /* connectivity list sizes (same as cell list size) */

  node_clistsz = cell_clistsz;

  /* default utility flag is set */

  util_flag = 0;

  /* set up the UCD flags

       names will be integers      - UCD_INT
       material ids will be stored - UCD_MATERIAL_IDS
       node names will be stored   - UCD_NODE_NAMES
       cell names will be stored   - UCD_CELL_NAMES

  */

  ucd_flags = UCD_INT | UCD_MATERIAL_IDS | UCD_NODE_NAMES | UCD_CELL_NAMES;

  /* construct model name from data_file name */
```

```
  if ( strlen(data_file) > TEMP_STRING_LEN )
  {
   strcpy (temp_string, "/tmp/noname.nff");
#if DEBUG
   printf ("The maximum length of temp_string was exceeded !\n");
   printf ("This is due to a very long pathname.\n");
   printf ("A default name will be used.\n");
#endif
  }
  else
  {
   strcpy (temp_string, data_file);
  }

  /* find last occurance of / and return address of this posn + 1 */

  model_name = (char *)(strrchr(temp_string,'/') + 1);

  /* start at end of string and place end of string marker where
     the file extension starts (e.g., fred.dat == fred\0 )        */

  for (i = strlen(model_name) ; model_name[i] != '.' ; i-- );
  model_name[i] = '\0';

  /* note the cast to UCD_structure from char* */

  *out_UCDstruct = (UCD_structure *)UCDstructure_alloc (model_name,
                                                        data_veclen,
                                                        ucd_flags,
                                                        numcells,
                                                        cell_clistsz,
                                                        cell_veclen,
                                                        numnodes,
                                                        node_clistsz,
                                                        node_veclen,
                                                        util_flag);

  if (*out_UCDstruct == 0)
  {
   AVSerror (" Error in read_bricks : UCD Structure not allocated\n");
   return(0);
  }



#if DEBUG
  printf("Read in the nodes coordinates\n");
#endif

  /* first allocate enough space to read in coordinates */

  node_x_coords = (float *)malloc(sizeof(float) * numnodes);
  node_y_coords = (float *)malloc(sizeof(float) * numnodes);
  node_z_coords = (float *)malloc(sizeof(float) * numnodes);

  /* allocate space for material types at nodes */
```

```
  node_mat_types = (int *)malloc(sizeof(int) * numnodes * nummats);

  for (i = 0; i < numnodes ; i++)
  {
   mod_percent = (int)(((float)i/(float)numnodes)*100.0);
   AVSmodule_status("Reading Coordinates",mod_percent);

   if (fscanf(dfp,"%d%f%f%f",&nodeid,&node_x_coords[i],
                                  &node_y_coords[i],
                                  &node_z_coords[i]) != 4)
   {
    AVSerror (" Error in read_bricks : cannot read node x y z (%d)\n",i);
    return(1);
   }

#if DEBUG
    printf ("######################################################\n");
    printf ("# Node id %d \n",nodeid);
    printf ("######################################################\n");
    printf ("Node       : x=%f\n",node_x_coords[i]);
    printf ("Node       : y=%f\n",node_y_coords[i]);
    printf ("Node       : z=%f\n",node_z_coords[i]);
#endif

    /* calculate the min and max extent of the nodes */

    if (i)
    {
     /* calculate xmin */

     if (min_extent[0] > node_x_coords[i])
     {
      min_extent[0] = node_x_coords[i] ;
     }

     /* calculate xmax */

     if (max_extent[0] < node_x_coords[i])
     {
      max_extent[0] = node_x_coords[i] ;
     }

     /* calculate ymin */

     if (min_extent[1] > node_y_coords[i])
     {
      min_extent[1] = node_y_coords[i] ;
     }

     /* calculate ymax */

     if (max_extent[1] < node_y_coords[i])
     {
      max_extent[1] = node_y_coords[i] ;
     }
```

```
     /* calculate zmin */

     if (min_extent[2] > node_z_coords[i])
     {
      min_extent[2] = node_z_coords[i] ;
     }

     /* calculate zmax */

     if (max_extent[2] < node_z_coords[i])
     {
      max_extent[2] = node_z_coords[i] ;
     }
    }
    else
    {
     /* initialise the min/max extents */

     min_extent[0] = node_x_coords[i]; max_extent[0] = node_x_coords[i];
     min_extent[1] = node_y_coords[i]; max_extent[1] = node_y_coords[i];
     min_extent[2] = node_z_coords[i]; max_extent[2] = node_z_coords[i];

    }/* end elseif */

#if DEBUG
     printf("Reading material types\n");
     printf("No of material types per node is %d\n",nummats);
#endif

   for (j=0;j < nummats; j++)
   {
#if DEBUG
    printf("Material %d\n",j);
#endif
    if (fscanf(dfp,"%d",&node_mat_types[(i*nummats)+j]) != 1)
    {
     AVSerror (" Error in read_bricks : cannot read material types (%d)\n",i);
     return(0);
    }
#if DEBUG
    printf("Material (%d,%d) is %d\n",i,j,node_mat_types[(i*nummats)+j]);
#endif
   }

   /* read in the end of line */
#if DEBUG
   printf("Reading the EOLN\n");
#endif
   if (fscanf(dfp,"\n") == EOF)
   {
    AVSerror (" Error in read_bricks : unexpected EOF\n");
    return(0);
   }
  }

#if DEBUG
   printf("Setting the node coordinates\n");
```

```
#endif

  if ( UCDstructure_set_node_positions (*out_UCDstruct,
                                        node_x_coords,
                                        node_y_coords,
                                        node_z_coords) == 0)
  {
   AVSerror (" Error in read_bricks : setting node coordinates\n");
   return(0);
  }

  /* set the min/max extent of the structure */

#if DEBUG
  printf("min/max of x coords is %f, %f\n",min_extent[0],max_extent[0]);
  printf("min/max of y coords is %f, %f\n",min_extent[1],max_extent[1]);
  printf("min/max of z coords is %f, %f\n",min_extent[2],max_extent[2]);
#endif

  if (UCDstructure_set_extent (*out_UCDstruct, min_extent, max_extent) == 0)
  {
   AVSerror (" Error in read_bricks : cannot set UCD structure extent.\n");
   return (0);
  }

  /* read the cell connectivity lists */

#if DEBUG
  printf("Reading the cell connectivity lists\n");
#endif

  cell_type = UCD_HEXAHEDRON;

  for (i=0; i < numcells ; i++)
  {
   mod_percent = (int)(((float)i/(float)numcells)*100.0);
   AVSmodule_status("Reading Cells",mod_percent);

   if (fscanf (dfp,"%d%d%d%d%d%d%d%d%d\n",&cellid,
                                          &node_list[0],
                                          &node_list[1],
                                          &node_list[5],
                                          &node_list[4],
                                          &node_list[3],
                                          &node_list[2],
                                          &node_list[6],
                                          &node_list[7]) != 9)
    {
     AVSerror (" Error in read_bricks : reading node connectivity\n");
     return (0);
    }

   /* readjust the node ids according to node name 1 == node id 0 */

   for (j=0; j < 8; j++)
   {
    node_list[j]--;
```

```
   }

#if DEBUG
    printf("Setting cell %d information\n",i);
#endif
   /* set cell information - UCD struct
                             cell to set
                             cell name
                             element type
                             material id (average out node material types)
                             cell type
                             mid edge data ? (0-no,1-yes)
                             list of nodes
   */

   if ( UCDcell_set_information ( *out_UCDstruct,
                                  i,
                                  cellid,
                                  "hex",
                                  1,
                                  cell_type,
                                  0,
                                  node_list ) == 0)
   {
    AVSerror (" Error in read_bricks : cannot set cell information\n");
    return(0);
   }
  }

   /* set the node names but do not calc node connectivity */
   /* reason : AVS inc say this is removed in version 4    */

   for (i=0; i<numnodes;i++)
   {
    if ( UCDnode_set_information (*out_UCDstruct,
                                  i,
                                  i+1,
                                  0,
                                  NULL) == 0 )
    {
     AVSerror (" Error in read_bricks : setting the node connectivity list\n");
     return (0);
    }
   }/* end i */

  /* set up the node data components */

#if DEBUG
  printf("Setting node components\n");
#endif

  num_components = NODE_DATA_COMPS;

  comp_list[0] = 3;   /* u,v,w velocity */
  comp_list[1] = 1;   /* pressure       */
  comp_list[2] = 1;   /* density        */
  comp_list[3] = 1;   /* viscosity      */
```

```
  comp_list[4] = 1;   /* trubulent K.E. */

  if ( UCDstructure_set_node_components (*out_UCDstruct,
                                         comp_list,
                                         num_components ) == 0)
  {
   AVSerror (" Error in read_bricks : setting the node components\n");
   return (0);
  }

  /* set the velocity to be the active component */

#if DEBUG
  printf("Setting the active list\n");
#endif

  active_list[0] = 1;
  active_list[1] = 0;
  active_list[2] = 0;
  active_list[3] = 0;
  active_list[4] = 0;

  if ( UCDstructure_set_node_active (*out_UCDstruct, active_list) == 0)
  {
   AVSerror (" Error in read_bricks : setting the active list\n");
   return(0);
  }
  /* set the data component labels */

#if DEBUG
  printf("Setting the data labels\n");
#endif

  data_labels = "velocity.pressure.density.viscosity.kinetic energy";

  if ( UCDstructure_set_node_labels (*out_UCDstruct,
                                    data_labels, ".") == 0)
  {
   AVSerror (" Error in read_bricks : setting the data labels\n");
   return (0);
  }

  /* set the data component units */

#if DEBUG
  printf("Setting the data units\n");
#endif

  data_units = "m/s.Newtons.m/s/s.m/s.Joules";

  if ( UCDstructure_set_node_units (*out_UCDstruct,
                                    data_units, ".") == 0)
  {
   AVSerror (" Error in read_bricks : setting the data units\n");
   return (0);
  }
```

```
  /* read in the 1st time step (change code to allow any
                               time step to be read)    */
#if DEBUG
  printf("Reading the time steps\n");
#endif

  if ( fscanf (dfp,"%d%f\n",&tstep,&tvalue) != 2)
  {
   AVSerror (" Error in read_bricks : reading timesteps\n");
   return(0);
  }

  /* allocate the node data */

  node_data = (float *)malloc(sizeof(float) * NODE_DATA_VALS * numnodes);

  /* read data values */

  for (i=0 ; i < numnodes ; i++)
  {
#if DEBUG
   printf("Reading data value for node %d\n",i);
#endif

   mod_percent = (int)(((float)i/(float)numnodes)*100.0);
   AVSmodule_status ("Reading Data",mod_percent);

   if (fscanf (dfp,"%d%f%f%f%f%f%f%f\n",&nodeid,
                                 &node_data[(i*3)],
                                 &node_data[(i*3)+1],
                                 &node_data[(i*3)+2],
                                 &node_data[i+numnodes*3],
                                 &node_data[i+numnodes*4],
                                 &node_data[i+numnodes*5],
                                 &node_data[i+numnodes*6]) != 8)
   {
    AVSerror (" Error in read_bricks : reading node data\n");
    return(0);
   }

  } /* end of i */

  /* set the node data values */

  if ( UCDstructure_set_node_data (*out_UCDstruct,node_data) == 0)
  {
   AVSerror (" Error in read_bricks : setting the node data\n");
   return (0);
  }

  /* initialise the min/max arrays                              */
  /* note that we don't bother calculating the min/max for vector */
  /* components as it is not used  i.e., component no 0         */

   min_node_data[0] = 0.0; max_node_data[0] = 1.0;

  for (j=1; j < NODE_DATA_COMPS ; j++)
```

```
  {
   min_node_data[j] = node_data[numnodes * (j+2)];
   max_node_data[j] = node_data[numnodes * (j+2)];
  }

  /* find the minimum and maximum data values */

#if DEBUG
  printf("Find the node data min/max values\n");
#endif

  for (i=0; i < numnodes ; i++)
  {
   for (j=1; j < NODE_DATA_COMPS ; j++)
   {
    /* find minimum value */

    if ( min_node_data[j] > node_data[(numnodes*(j+2))+i] )
    {
     min_node_data[j] = node_data[(numnodes*(j+2))+i];
    }
    /* find maximum value */

    if ( max_node_data[j] < node_data[(numnodes*(j+2))+i] )
    {
     max_node_data[j] = node_data[(numnodes*(j+2))+i];
    }
   }/* end j */
  }/* end i */

  /* set the min/max data values */

#if DEBUG
  printf("Set the node data min/max values\n");
  for (j=0; j < NODE_DATA_COMPS ; j++)
  {
   printf("Minimum value for %d is %f\n",j,min_node_data[j]);
   printf("Maximum value for %d is %f\n",j,max_node_data[j]);
  }
#endif

  if ( UCDstructure_set_node_minmax (*out_UCDstruct,
                                     min_node_data,
                                     max_node_data ) == 0 )
  {
   AVSerror (" Error in read_bricks : setting min/max data values\n");
   return (0);
  }

#if DEBUG
  printf("Closing the datafile\n");
#endif

  close(dfp);

  /* free the storage */
```

```
#if DEBUG
  printf("Freeing the store\n");
#endif

  free(node_x_coords);
  free(node_y_coords);
  free(node_z_coords);

  free(node_mat_types);

  free(node_data);

  return (1);
}

read_bricks()
{
 int out_port,
     param;

 AVSset_module_name ("Read Bricks",MODULE_DATA);

 out_port = AVScreate_output_port ("Output Structure", "ucd");

 /* set up the file browser widget for the data file */

 param = AVSadd_parameter ("Read File", "string", 0, 0, ".brick");
 AVSconnect_widget (param, "browser");

 AVSset_compute_proc (read_bricks_compute);
}

AVSinit_modules()
{
 AVSmodule_from_desc (read_bricks);
}
```

7 Discs (ex6)

7.1 Description of the module

This module will execute as a coroutine module and will generate a sequence of spheres with the following characteristics:

- X coordinate will increase linearly
- Y coordinate will randomly vary
- Z coordinate will remain fixed

7.2 Specification of the module

This is the specification of the module but do **not** use the module generator to produce the skeleton file; instead use `ex6/discs.c`.

Module name:	discs
Module class:	data input
Language:	C
Module type:	coroutine
Output port name 0:	output_discs
Output port type 0:	GEOMedit_list
Parameter name 0:	sleep
Parameter type 0:	boolean (toggle)
Parameter default 0:	1
File name:	`discs.c`

7.3 The exercise

1. Use the file `ex6/discs.c` as a template and add the arguments for all the routines starting `AVScorout`.

The source code for the `discs.c` file is shown below and a full listing of the answer can be found in appendix E.

7.4 Source Code (discs.c)

```
#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/geom.h>

#define MAX_DISCS 100

/* ****************************************/
/* Coroutine Main Routine                 */
/* ****************************************/
int main(argc, argv)
int argc;
char **argv;
{

        int num_discs;
int discs_desc();
int sleep = 1;
GEOMobj *obj = NULL;
GEOMedit_list output_discs = NULL;
float verts[3*MAX_DISCS] ;
float radii[MAX_DISCS] ;
float xpos,ypos;

double drand48();

verts[0] = 0.0;
verts[1] = 0.0;
verts[2] = 0.0;
radii[0] = 0.1;

num_discs = 1;
xpos = 0.0;
ypos = 0.0;

srand48(1.0);

################################ INSERT CODE HERE #######################################

AVScorout_init;

/* start of the main loop forever */

while (1) {

  /* check if the sleep parameter is false */
  /* this will cause corout_wait to exit   */

################################ INSERT CODE HERE #######################################

  if ((!sleep) && (num_discs < MAX_DISCS)) AVScorout_mark_changed;

################################ INSERT CODE HERE #######################################

  AVScorout_wait;
```

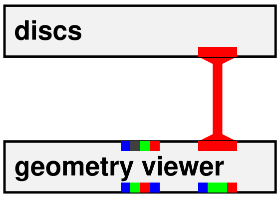
```
  /* get the new value of the sleep parameter */

################################ INSERT CODE HERE #######################################

  AVScorout_input;

  obj = GEOMcreate_sphere (NULL,
   verts,
   radii,
   NULL,
   NULL,
   num_discs,
   GEOM_COPY_DATA);

  /* create the edit list */

  output_discs = (GEOMedit_list) GEOMinit_edit_list(output_discs);

  /* add the disc object to the edit list */

  GEOMedit_geometry (output_discs,"disc",obj);

  /* send the edit list to the output port */

################################ INSERT CODE HERE #######################################

  AVScorout_output;

  /* update the disc position */

  num_discs++;
  xpos = xpos + 0.2;
  ypos = ypos + (float)( 0.2 * drand48() );
  ypos = ypos - (float)( 0.2 * drand48() );
  verts[(num_discs-1)*3] = xpos;
  verts[(num_discs-1)*3+1] = ypos;
  verts[(num_discs-1)*3+2] = 0.0;
  radii[num_discs-1] = 0.1;

}

}

/* ****************************************/
/*  Module Description                    */
/* ****************************************/
int discs_desc()
{

int in_port, out_port, param;

AVSset_module_name("discs", MODULE_DATA);

/* Output Port Specifications             */
out_port = AVScreate_output_port("output_discs", "geom");
```

```
/* Parameter Specifications                 */
param = AVSadd_parameter("sleep", "boolean", 1,0,1);
AVSconnect_widget(param, "toggle");

return(1);
}
```

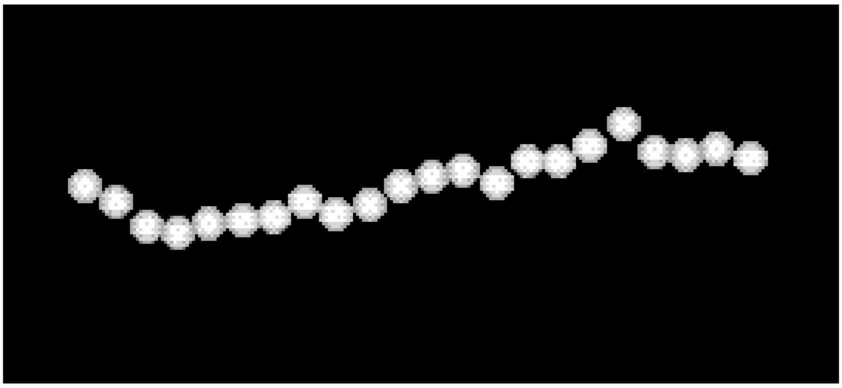
7.5 Testing the module

The following network will be useful for testing your module executes correctly:

Geometry Viewer



Click on the "sleep" button to start the coroutine executing and if the module executes correctly then the `geometry_viewer` should show something similar to the figure below:



7.6 Additional exercise

Alter the module to output a series of lines instead of spheres.

8 Add Extents (ex7)

8.1 Description of add_extents

The add_extents module copies the input field to the output field whilst recalculating the extent information for the data and coordinates supplied in the input field. All extra field information is copied to the output field e.g., labels, units, data, coordinates, type

It will accept any type of field, with N dimensions and M data components per data element and is useful when modules further up a network produce fields which do not have the extent information assigned.

8.2 Source Code (add_extents.c)

```
/*
    Module Name : add_extents

    Author : Steve Larkin
             Computer Graphics Unit
             Manchester Computing Centre
             University of Manchester
             Oxford Road
             Manchester M13 9PL
     United Kingdom

             Tel : +44 (0)61 275 6095
             Fax : +44 (0)61 275 6040

    Date : 19th March 1993

    Description : adds the extent information to the input field


*/

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* *****************************************/
/*  Module Description                   */
/* *****************************************/
int add_extents_desc()
```

```
{

int in_port, out_port, param;
extern int add_extents_compute();

AVSset_module_name("add_extents", MODULE_FILTER);

/* Input Port Specifications               */
in_port = AVScreate_input_port("field_in", "field", REQUIRED);

/* Output Port Specifications              */
out_port = AVScreate_output_port("field_out", "field");

AVSset_compute_proc(add_extents_compute);

return(1);
}

/* ****************************************/
/* Module Compute Routine                 */
/* ****************************************/
int add_extents_compute( field_in, field_out)
AVSfield *field_in;
AVSfield **field_out;
{
  int in_nspace,
      in_ndim,
      *in_dims,
      i,j,
      num_elems,
      offset,
      in_type,
      in_veclen,
      in_map_type,
      *ptr_int;

  float *min_ext,
        *max_ext,
        coord_value,
        *coords,
        *ptr_float;

  char  in_lab_str[AVS_FIELD_LABEL_LEN],
        in_lab_delim[AVS_FIELD_LABEL_LEN],
        in_unit_str[AVS_FIELD_UNIT_LEN],
        in_unit_delim[AVS_FIELD_UNIT_LEN],
        *ptr_char;

  double *ptr_double;

  /* free the output field if allocated */

  if (*field_out != NULL) AVSfield_free (*field_out);

  /* copy as much of the field as possible using AVSfield_alloc */
  /*                                                            */
  /* This routine will copy across:                             */
```

```
  /*                                                                   */
  /*    nspace,veclen,type,size,mapping type,points,dims               */
  /*                                                                   */
  /* This routine does not copy:                                       */
  /*                                                                   */
  /*    labels,units,min,max,min_ext,max_ext                           */

  /* inquire type of field */

  in_type = AVSfield_get_int(field_in,AVS_FIELD_TYPE);

  *field_out = (AVSfield *)AVSfield_alloc (field_in,0);

  /* inquire nspace of the field_in */

  in_nspace = AVSfield_get_int (field_in,AVS_FIELD_NSPACE);

  /* inquire the dimensions of the field_in */

  in_ndim = AVSfield_get_int (field_in,AVS_FIELD_NDIM);
  in_dims = (float *)malloc( sizeof (float) * in_ndim);
  AVSfield_get_dimensions (field_in,in_dims);

  /* allocate space for the min and max extent */

  min_ext = (float *)malloc( sizeof(float) * in_nspace );
  max_ext = (float *)malloc( sizeof(float) * in_nspace );

  /* find out what the min and max extents are */

  in_map_type = AVSfield_get_int(field_in,AVS_FIELD_UNIFORM);

  coords = field_in->points;

  switch (in_map_type)
    {
    case UNIFORM:

      for (i=0; i < in_ndim; i++)
{
  min_ext[i] = 0.0;
  max_ext[i] = in_dims[i]-1;
}
      break;

    case RECTILINEAR:

      offset = 0;
      for (i=0; i < in_ndim ; i++)
{
for (j=0; j < in_dims[i] ; j++)
  {
    coord_value = coords[j + offset];

    if (j==0)
      {
min_ext[i] = coord_value;
```

```
max_ext[i] = coord_value;
      }
    else
      {
if (coord_value < min_ext[i]) min_ext[i] = coord_value;
if (coord_value > max_ext[i]) max_ext[i] = coord_value;
      }
  } /* end of j */

offset += in_dims[i];
      } /* end of i */

      break;

    case IRREGULAR:

      num_elems = 1;

      for (i=0; i < in_ndim ; i++)
num_elems *= in_dims[i] ;

      for (j=0; j < in_nspace ; j++)
{
  for (i=0; i < num_elems ; i++)
    {
      coord_value = coords[i + (j*num_elems)];

      if (i==0)
{
  min_ext[j] = coord_value;
  max_ext[j] = coord_value;
}
      else
{
  if (coord_value < min_ext[j]) min_ext[j] = coord_value;
  if (coord_value > max_ext[j]) max_ext[j] = coord_value;
}
    } /* end of i */
} /* end of j */

      break;
    }

  /* set the extents */

  AVSfield_set_extent (*field_out,min_ext,max_ext);

  /* copy across the data */

  in_veclen = AVSfield_get_int(field_in,AVS_FIELD_VECLEN);

  num_elems = 1;

  for (i=0; i < in_ndim ; i++)
    num_elems *= in_dims[i] ;

  switch (in_type)
```

```
    {
    case 0: /* byte data */

      ptr_char = ((*field_out)->field_union.field_data_char);

      for (i=0; i < (in_veclen*num_elems) ; i++)
*(ptr_char)++ = *(field_in->field_union.field_data_char)++;

      break;

    case 1: /* integer data */

      ptr_int = ((*field_out)->field_union.field_data_int_u);

      for (i=0; i < (in_veclen*num_elems) ; i++)
*(ptr_int)++ = *(field_in->field_union.field_data_int_u)++;

      break;

    case 2: /* float data */

      ptr_float = ((*field_out)->field_union.field_data_float_u);

      for (i=0; i < (in_veclen*num_elems) ; i++)
*(ptr_float)++ = *(field_in->field_union.field_data_float_u)++;

      break;

    case 3: /* double data */

      ptr_double = ((*field_out)->field_union.field_data_double_u);

      for (i=0; i < (in_veclen*num_elems) ; i++)
*(ptr_double)++ = *(field_in->field_union.field_data_double_u)++;

      break;

    }

  /* copy across the labels */

  if (AVSfield_get_labels (field_in,in_lab_str,in_lab_delim) == 1)
    {
      AVSfield_set_labels (*field_out,in_lab_str,in_lab_delim);
    }

  /* copy across the units */

  if (AVSfield_get_units (field_in,in_unit_str,in_unit_delim) == 1)
    {
      AVSfield_set_units (*field_out,in_unit_str,in_unit_delim);
    }

  /* reset min and max data values */

  AVSfield_reset_minmax (*field_out);
```

```
return(1);
}

/* ***********************************************************************/
/* Initialization for modules contained in this file.                    */
/* ***********************************************************************/
int ((*mod_list[])()) = {
add_extents_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}
```

9 Sobel Edge Detection

9.1 Description of the module

This module will import a 2D array of data with a single scalar byte at each data element. The data will then be processed to detect sharp changes in intensity i.e., edges by passing a sobel filter across the image.

9.2 Specification of the module

A specification of the module follows:

Module name:	edge_detection
Module class:	filter
Language:	C
Module type:	subroutine
Input port name 0:	input
Input port type 0:	"field 2D 2-space scalar uniform float"
Output port name 0:	output
Output port type 0:	"field 2D 2-space scalar uniform float"
File name:	`edge.c`

9.3 The exercise

The following equation shows how the resultant pixel P' is calculated from its neighbouring pixels.

B	C	D
A	P	E
H	G	F

$$X = (B + 2C + D) - (H + 2G + F)$$

$$Y = (D + 2E + F) - (B + 2A + H)$$

$$P' = \sqrt{X^2 + Y^2}$$

A module should be written to accept a 2D field of floats and peform the above filtering process on the image.

1. Include the file `ex8/edge.body` in the user defined code area of the `edge_detection_compute` routine and fill in the appropriate code where indicated. N.B., a header file has also been included in the appropriate user-supplied code area.

The source code for the `edge.body` file is shown below and a full listing of the answer can be found in appendix F.

9.4 Source Code (`edge.body`)

```
/* mod_gen Version 1                                                   */
/* Module Name: "edge_detection" (Filter) (Subroutine)                 */
/* Author: Naomi Hill,CGU (Summer Student),6095                        */
/* Date Created: Tue Aug  9 16:36:48 1994                              */
/*                                                                     */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as  */
/* mod_gen needs it in order to read module sources back in.           */
/*                                                                     */
/* input 0 "input" field 2D 2-space 1-vector uniform float REQUIRED    */
/* output 0 "output" field 2D 2-space 1-vector uniform float           */
/* End of Module Description Comments                                  */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/

############################## INSERT CODE HERE ##############################

/* <---- END OF USER-SUPPLIED CODE SECTION #1                           */
```

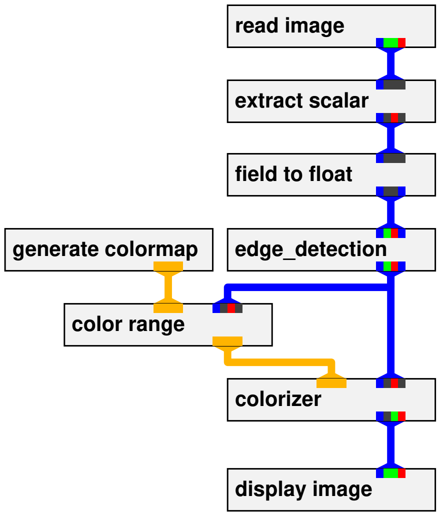
```
/* ****************************************/
/*  Module Description                   */
/* ****************************************/
int edge_detection_desc()
{

int in_port, out_port, param, iresult;
extern int edge_detection_compute();

AVSset_module_name("edge_detection", MODULE_FILTER);

/* Input Port Specifications             */
in_port = AVScreate_input_port("input",
"field 2D 2-space 1-vector uniform float", REQUIRED);

/* Output Port Specifications            */
out_port = AVScreate_output_port("output",
"field 2D 2-space 1-vector uniform float");

AVSset_compute_proc(edge_detection_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2                             */
return(1);
}

/* ****************************************/
/* Module Compute Routine                */
/* ****************************************/
int edge_detection_compute( input, output)
AVSfield_float *input;
AVSfield_float **output;
{

/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */

int dims[2], i, j;
float X, Y;


/* Free old field data                */

############################## INSERT CODE HERE ##############################


/* Allocate space for new field output                                   */

/* set the dims using MAXX and MAXY */

############################## INSERT CODE HERE ##############################


/* create and allocate memory for *output using AVSdata_alloc */

############################## INSERT CODE HERE ##############################


/* check to ensure field allocation successful if not return */
```

```
############################## INSERT CODE HERE ##############################



/* Compute the sobel algorithm                                          */

for (i=0; i<MAXX(input); i++)
for (j=0; j<MAXY(input); j++)

/* check to see if the current pixel is an edge pixel */

############################## INSERT CODE HERE ##############################

/* if so redefine the pixel to 0.0 */

############################## INSERT CODE HERE ##############################


/* for non edge pixels perform the filtering process */

############################## INSERT CODE HERE ##############################

/* calculate the X value */

############################## INSERT CODE HERE ##############################


/* calculate the Y value */

############################## INSERT CODE HERE ##############################


/* redefine the pixel, using the algorithm, to P' */

############################## INSERT CODE HERE ##############################




/* <---- END OF USER-SUPPLIED CODE SECTION #3                           */
return(1);
}

/* ********************************************************************/
/* Initialization for modules contained in this file.                   */
/* ********************************************************************/
static int ((*mod_list[])()) = {
edge_detection_desc
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}
```

```
/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                                             */
```

9.5 Testing the module

The following network will allow you to test the module:



If 'red' is chosen from the `extract_scalar` parameters the result should be similar to the following.

A Answer to Threshold (ex1)

A.1 Source Code (threshold.c)

```
/* mod_gen Version 1                                                      */
/* Module Name: "do_threshold" (Filter) (Subroutine)                      */
/* Author: Steve Larkin,MCC CGU,6096,                                     */
/* Date Created: Fri Mar 19 11:24:55 1993                                 */
/*                                                                        */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as    */
/* mod_gen needs it in order to read module sources back in.             */
/*                                                                        */
/* input 0 "input" field 3D 3-space 1-vector uniform float REQUIRED      */
/* output 0 "output" field 3D 3-space 1-vector uniform float             */
/* param 0 "pmin" dial 0.00000 0.00000 1.00000                           */
/* param 1 "pmax" dial 1.00000 0.00000 1.00000                           */
/* End of Module Description Comments                                     */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1                             */

/* ***************************************/
/*  Module Description                  */
/* ***************************************/
int do_threshold_desc()
{

int in_port, out_port, param;
extern int do_threshold_compute();

AVSset_module_name("do_threshold", MODULE_FILTER);

/* Input Port Specifications            */
in_port = AVScreate_input_port("input",
"field 3D 3-space 1-vector uniform float", REQUIRED);

/* Output Port Specifications           */
out_port = AVScreate_output_port("output",
"field 3D 3-space 1-vector uniform float");
```

```
/* Parameter Specifications                */
param = AVSadd_float_parameter("pmin", 0.00000, 0.00000, 1.000);
AVSconnect_widget(param, "dial");
param = AVSadd_float_parameter("pmax", 1.000, 0.00000, 1.000);
AVSconnect_widget(param, "dial");

AVSset_compute_proc(do_threshold_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2                          */
return(1);
}

/* ****************************************/
/* Module Compute Routine                  */
/* ****************************************/
int do_threshold_compute( input, output, pmin, pmax)
AVSfield_float *input;
AVSfield_float **output;
float *pmin;
float *pmax;
{

/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */

    int i, j, k;
    float min = *pmin;
    float max = *pmax;

    /* allocate the output field to have same structure as input */

    *output = (AVSfield_float *)AVSfield_alloc (input,0);

    /*
     * We use a triply nested loop to traverse the field.  The macros MAXX,
     * MAXY, and MAXZ determine the maximum extent of the field in each of
     * the three dimensions.  We know this will be a 3-dimensional
     * field because of the declaration in the description routine, so
     * we don't need to check.  When we want to reference an element of the
     * field we use the I3D macro which picks an element of a 3D field.
     * Note that the first index (i) varies the fastest in memory, so we
     * make that the innermost loop.
     */

     for (k = 0; k < MAXZ(input); k++)
     for (j = 0; j < MAXY(input); j++)
     for (i = 0; i < MAXX(input); i++)
    if (I3D(input, i, j, k) > max) {
   I3D(*output, i, j, k) = 0.0;
    } else if (I3D(input, i, j, k) < min) {
   I3D(*output, i, j, k) = 0.0;
    } else {
   I3D(*output, i, j, k) = I3D(input, i, j, k);
    }


/* <---- END OF USER-SUPPLIED CODE SECTION #3                          */
```

```
return(1);
}

/* ***********************************************************************/
/* Initialization for modules contained in this file.                   */
/* ***********************************************************************/
int ((*mod_list[])()) = {
do_threshold_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                            */
```

A.2 Source Code (makefile)

```
#
# Makefile for threshold
#

INC_FILE=$(ROOT)/usr/avs/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(ROOT)/usr/avs/lib
INC = $(ROOT)/usr/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
DEBUGFLAGS = -I$(INC) -g
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

threshold: threshold.o
$(CC) $(CFLAGS) -o threshold threshold.o $(CFLOWLIBS)

threshold.o: threshold.c
$(CC) $(CFLAGS)  -c threshold.c

debug:
$(CC) $(DEBUGFLAGS)  -c threshold.c
$(CC) $(DEBUGFLAGS) -o threshold threshold.o $(CFLOWLIBS)
```

B Answer to Read Temps (ex2)

B.1 Source Code (read_temps.c)

```
/* mod_gen Version 1                                                    */
/* Module Name: "read_temps" (Input) (Subroutine)                       */
/* Author: Steve Larkin,MCC CGU,6096,                                   */
/* Date Created: Fri Mar 19 13:22:53 1993                               */
/*                                                                      */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as   */
/* mod_gen needs it in order to read module sources back in.            */
/*                                                                      */
/* output 0 "temp_field" field 2D 2-space 1-vector uniform float        */
/* param 0 "datafile" browser "" "" ".dat"                              */
/* End of Module Description Comments                                   */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1                           */

/* ****************************************/
/*  Module Description                   */
/* ****************************************/
int read_temps_desc()
{

int in_port, out_port, param;
extern int read_temps_compute();

AVSset_module_name("read_temps", MODULE_DATA);

/* Output Port Specifications            */
out_port = AVScreate_output_port("temp_field",
"field 2D 2-space 1-vector uniform float");

/* Parameter Specifications              */
param = AVSadd_parameter("datafile", "string", 0, 0, ".dat");
AVSconnect_widget(param, "browser");

AVSset_compute_proc(read_temps_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
```

```
/* <---- END OF USER-SUPPLIED CODE SECTION #2                              */
return(1);
}

/* ***************************************/
/* Module Compute Routine               */
/* ***************************************/
int read_temps_compute( temp_field, datafile)
AVSfield_float **temp_field;
char *datafile;
{
/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */

  FILE *file_ptr;
  int  dims[2];
  int  i,j;
  float datavalue;

  /* check to see if a datafile has been supplied */

  if (!datafile) return (1);

  /* check the datafile is readable */

  if (!(file_ptr = fopen (datafile,"r")) )
      {
AVSerror ("Error in read_temps cannot open file\n %s\n",datafile);
return(0);
}

  /* read in the dimensions of the field */

  fscanf (file_ptr,"%d%d",&dims[0],&dims[1]);

  /* check to see if the field has been previously allocated */

  if (*temp_field != NULL)
  {
   /* if so then free the space */

   AVSfield_free (*temp_field);
  }

  /* allocate enough space for the output field */

  *temp_field = (AVSfield_float *)
             AVSdata_alloc ("field 2D 2-space scalar uniform float",dims);

  /* read the data from the file and add to the field */

  for (i=0; i < dims[0]; i++)
    for (j=0; j < dims[1]; j++)
      {
/* add the data to the field */

fscanf (file_ptr,"%f",&datavalue);
I2D(*temp_field,i,j) = datavalue;
```

```
      }

/* <---- END OF USER-SUPPLIED CODE SECTION #3                            */
return(1);
}

/* ********************************************************************/
/* Initialization for modules contained in this file.                   */
/* ********************************************************************/
int ((*mod_list[])()) = {
read_temps_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                            */
```

B.2 Source Code (`makefile`)

```
#
# Makefile for read_temps
#

INC_FILE=$(ROOT)/usr/avs/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(ROOT)/usr/avs/lib
INC = $(ROOT)/usr/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

read_temps: read_temps.o
$(CC) $(CFLAGS) -o read_temps read_temps.o $(CFLOWLIBS)

read_temps.o: read_temps.c
$(CC) $(CFLAGS)  -c read_temps.c
```

C Answer to Read Conc (ex3)

C.1 Source Code (read_conc.c)

```
#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1                                   */

/* ****************************************/
/*  Module Description                    */
/* ****************************************/
int read_conc_desc()
{

int in_port, out_port, param;
extern int read_conc_compute();

AVSset_module_name("read_conc", MODULE_DATA);

/* Output Port Specifications             */
out_port = AVScreate_output_port("conc_field",
"field 1D 3-space 1-vector irregular float");

/* Parameter Specifications               */
param = AVSadd_parameter("datafile", "string", 0, 0, ".dat");
AVSconnect_widget(param, "browser");

AVSset_compute_proc(read_conc_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2                                   */
return(1);
}

/* ****************************************/
/* Module Compute Routine                 */
/* ****************************************/
int read_conc_compute(conc_field, datafile)
AVSfield_float **conc_field;
char *datafile;
{
/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */
```

```
  FILE *file_ptr;
  int  dims[1];
  int  i,j;
  float xpos,ypos,zpos,datavalue;
  float *coords;

  /* check to see if a datafile has been supplied */

  if (!datafile) return (1);

  /* check the datafile is readable */

  if (!(file_ptr = fopen (datafile,"r")) )
      {
AVSerror ("Error in read_conc cannot open file\n %s\n",datafile);
return(0);
}

  /* read in the dimension of the field */

  fscanf (file_ptr,"%d",&dims[0]);

  /* check to see if the field has been previously allocated */

  if (*conc_field != NULL)
  {
   /* if so then free the space */

   AVSfield_free (*conc_field);
  }

  /* allocate enough space for the output field */

  *conc_field = (AVSfield_float *)
             AVSdata_alloc ("field 1D 3-space scalar irregular float",dims);

  /* read the data from the file and add to the field */

  coords = (*conc_field)->points;

  for (i=0; i < dims[0]; i++)
    {
      /* add the data to the field */

      fscanf (file_ptr,"%f%f%f%f",&xpos,&ypos,&zpos,&datavalue);
      *I1DV(*conc_field,i) = datavalue;

      /* add the coordinate information */

      coords[i] = xpos;
      coords[i + dims[0]] = ypos;
      coords[i + (2*dims[0])] = zpos;

      }

/* <---- END OF USER-SUPPLIED CODE SECTION #3                            */
return(1);
```

```
}

/* ***********************************************************************/
/* Initialization for modules contained in this file.                    */
/* ***********************************************************************/
int ((*mod_list[])()) = {
read_conc_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                            */
```

C.2 Source Code (makefile)

```
#
# Makefile for read_conc
#

INC_FILE=$(ROOT)/usr/avs/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(ROOT)/usr/avs/lib
INC = $(ROOT)/usr/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

read_conc: read_conc.o
$(CC) $(CFLAGS) -o read_conc read_conc.o $(CFLOWLIBS)

read_conc.o: read_conc.c
$(CC) $(CFLAGS)  -c read_conc.c
```

D Answer to Scatter Plot (ex4)

D.1 Source Code (scatter_plot.c)

```
#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>
#include <avs/geom.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #1                                    */

/* ****************************************/
/*  Module Description                    */
/* ****************************************/
int scatter_plot_desc()
{

int in_port, out_port, param;
extern int scatter_plot_compute();

AVSset_module_name("scatter_plot", MODULE_MAPPER);

/* Input Port Specifications              */
in_port = AVScreate_input_port("scatter_data",
"field 1D 3-space 1-vector irregular float", REQUIRED);

/* Output Port Specifications             */
out_port = AVScreate_output_port("scatter_obj", "geom");

/* parameter specifications */

param = AVSadd_float_parameter("pradii", 0.0, 0.0, 0.2);
        AVSconnect_widget(param, "dial");

AVSset_compute_proc(scatter_plot_compute);

/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2                                  */
return(1);
}

/* ****************************************/
/* Module Compute Routine                 */
/* ****************************************/
```

```
int scatter_plot_compute( scatter_data, scatter_obj, pradii)
AVSfield_float *scatter_data;
GEOMedit_list *scatter_obj;
        float *pradii;
{

/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */

  GEOMobj *spheres;
  int dims[1],i;
  float *centres,*coords,*ptr_centres,*ptr_colours,*colours,
        *radii,*ptr_radii;
  float minval,maxval,midval;

  /* get the number of points */

  AVSfield_get_dimensions (scatter_data,dims);

  /* allocate enough space for each sphere */

  centres = (float *)malloc(sizeof(float)*dims[0]*3);
  colours = (float *)malloc(sizeof(float)*dims[0]*3);
  radii   = (float *)malloc(sizeof(float)*dims[0]);

  /* get the coords for each sphere */

  coords = scatter_data->points;
  ptr_centres = centres;
  ptr_colours = colours;
  ptr_radii   = radii;

  /* find the min and max data value */

  AVSfield_get_minmax(scatter_data,&minval,&maxval);

  /* calculate the mid range */

  midval = (maxval + minval) / 2;

  for (i=0; i < dims[0] ; i++)
    {
      *radii++ = *pradii; /* radius from dial parameter */

      *centres++ = coords[i];
      *centres++ = coords[i + dims[0]];
      *centres++ = coords[i + (2*dims[0])];

      if ( *I1DV(scatter_data,i) < midval )
{
  *colours++ = 0.0; /* red   */
  *colours++ = 0.0; /* green */
  *colours++ = 1.0; /* blue  */
}
      else
{
  *colours++ = 1.0;
  *colours++ = 0.0;
```

```
  *colours++ = 0.0;
}
    }

  centres = ptr_centres;
  colours = ptr_colours;
  radii   = ptr_radii;

  /* create the edit list */

  *scatter_obj = (GEOMedit_list *)GEOMinit_edit_list(*scatter_obj);

  /* create the list of spheres */

  if ( *pradii == 0.0 )
    {
      /* if radii is 0.0 then create "real" dots */

  spheres = GEOMcreate_sphere (NULL,       /* extent */
       centres,
       GEOM_NULL,  /* radii = 0 (dots) */
       NULL,       /* normals */
       NULL,       /* colours */
       dims[0],    /* number of spheres */
       GEOM_COPY_DATA);
    }
  else
    {

  spheres = GEOMcreate_sphere (NULL,       /* extent */
       centres,
       radii,
       NULL,       /* normals */
       NULL,       /* colours */
       dims[0],    /* number of spheres */
       GEOM_COPY_DATA);
    }

  GEOMadd_float_colors (spheres,colours,dims[0],GEOM_COPY_DATA);

  GEOMauto_transform (spheres);

  /* now add spheres to the edit list */

  GEOMedit_geometry(*scatter_obj,"scatter_obj",spheres);

  /* free up the spheres */

  GEOMdestroy_obj(spheres);

  /* free the allocate arrays */

  free (centres);
  free (colours);
  free (radii);

/* <---- END OF USER-SUPPLIED CODE SECTION #3                            */
```

```
return(1);
}

/* ***********************************************************************/
/* Initialization for modules contained in this file.                    */
/* ***********************************************************************/
int ((*mod_list[])()) = {
scatter_plot_desc,
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                            */
```

D.2 Source Code (`makefile`)

```
#
# Makefile for scatter_plot
#

INC_FILE=$(ROOT)/usr/avs/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(ROOT)/usr/avs/lib
INC = $(ROOT)/usr/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

scatter_plot: scatter_plot.o
$(CC) $(CFLAGS) -o scatter_plot scatter_plot.o $(CFLOWLIBS)

scatter_plot.o: scatter_plot.c
$(CC) $(CFLAGS)  -c scatter_plot.c
```

E Answer to Discs (ex6)

E.1 Source Code (discs.c)

```
#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/geom.h>

#define MAX_DISCS 100

/* ****************************************/
/* Coroutine Main Routine                */
/* ****************************************/
int main(argc, argv)
int argc;
char **argv;
{

        int num_discs;
int discs_desc();
int sleep = 1;
GEOMobj *obj = NULL;
GEOMedit_list output_discs = NULL;
float verts[3*MAX_DISCS] ;
float radii[MAX_DISCS] ;
float xpos,ypos;

double drand48();

verts[0] = 0.0;
verts[1] = 0.0;
verts[2] = 0.0;
radii[0] = 0.1;

num_discs = 1;
xpos = 0.0;
ypos = 0.0;

srand48(1.0);

AVScorout_init(argc,argv,discs_desc);

/* start of the main loop forever */

while (1) {
```

```
  /* check if the sleep parameter is false */
  /* this will cause corour_wait to exit   */

  if ((!sleep) && (num_discs < MAX_DISCS)) AVScorout_mark_changed() ;

  AVScorout_wait ();

  /* get the new value of the sleep parameter */

  AVScorout_input(&sleep);

  obj = GEOMcreate_sphere (NULL,
   verts,
   radii,
   NULL,
   NULL,
   num_discs,
   GEOM_COPY_DATA);

  /* create the edit list */

  output_discs = (GEOMedit_list) GEOMinit_edit_list(output_discs);

  /* add the disc object to the edit list */

  GEOMedit_geometry (output_discs,"disc",obj);

  /* send the edit list to the output port */

  AVScorout_output(output_discs);

  /* update the disc position */

  num_discs++;
  xpos = xpos + 0.2;
  ypos = ypos + (float)( 0.2 * drand48() );
  ypos = ypos - (float)( 0.2 * drand48() );
  verts[(num_discs-1)*3] = xpos;
  verts[(num_discs-1)*3+1] = ypos;
  verts[(num_discs-1)*3+2] = 0.0;
  radii[num_discs-1] = 0.1;

}

}

/* ****************************************/
/*  Module Description                    */
/* ****************************************/
int discs_desc()
{

int in_port, out_port, param;

AVSset_module_name("discs", MODULE_DATA);
```

```
/* Output Port Specifications               */
out_port = AVScreate_output_port("output_discs", "geom");

/* Parameter Specifications                 */
param = AVSadd_parameter("sleep", "boolean", 1,0,1);
AVSconnect_widget(param, "toggle");

return(1);
}
```

E.2 Source Code (`makefile`)

```
#
# Makefile for discs
#

INC_FILE=$(ROOT)/usr/avs/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(ROOT)/usr/avs/lib
INC = $(ROOT)/usr/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

discs: discs.o
$(CC) $(CFLAGS) -o discs discs.o $(CSIMLIBS)

discs.o: discs.c
$(CC) $(CFLAGS)  -c discs.c
```

F Answer to Edge Detection (ex8)

F.1 Source Code (edge.c)

```
/* mod_gen Version 1                                                    */
/* Module Name: "edge_detection" (Filter) (Subroutine)                  */
/* Author: Naomi Hill,CGU (Summer Student),6095                         */
/* Date Created: Tue Aug  9 16:36:48 1994                               */
/*                                                                      */
/* This file is automatically generated by the Module Generator (mod_gen)*/
/* Please do not modify or move the contents of this comment block as   */
/* mod_gen needs it in order to read module sources back in.            */
/*                                                                      */
/* input 0 "input" field 2D 2-space 1-vector uniform float REQUIRED     */
/* output 0 "output" field 2D 2-space 1-vector uniform float            */
/* End of Module Description Comments                                   */

#include <stdio.h>
#include <avs/avs.h>
#include <avs/port.h>
#include <avs/field.h>

/* ----> START OF USER-SUPPLIED CODE SECTION #1 (INCLUDE FILES, GLOBAL VARIABLES)*/

/* include file math.h needed for sqrt */
#include <math.h>

/* <---- END OF USER-SUPPLIED CODE SECTION #1                           */

/* *****************************************/
/*  Module Description                    */
/* *****************************************/
int edge_detection_desc()
{

int in_port, out_port, param, iresult;
extern int edge_detection_compute();

AVSset_module_name("edge_detection", MODULE_FILTER);

/* Input Port Specifications              */
in_port = AVScreate_input_port("input",
"field 2D 2-space 1-vector uniform float", REQUIRED);
```

```
/* Output Port Specifications              */
out_port = AVScreate_output_port("output",
"field 2D 2-space 1-vector uniform float");

AVSset_compute_proc(edge_detection_compute);
/* ----> START OF USER-SUPPLIED CODE SECTION #2 (ADDITIONAL SPECIFICATION INFO)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #2                            */
return(1);
}

/* ****************************************/
/* Module Compute Routine                 */
/* ****************************************/
int edge_detection_compute( input, output)
AVSfield_float *input;
AVSfield_float **output;
{

/* ----> START OF USER-SUPPLIED CODE SECTION #3 (COMPUTE ROUTINE BODY)   */

int dims[2], i, j;
float X, Y;


/* Free old field data                  */

if (*output) AVSfield_free(*output);



/* Allocate space for new field output                                   */

/* set the dims using MAXX and MAXY */
dims[0] = MAXX(input);
dims[1] = MAXY(input);

/* create and allocate memory for *output using AVSdata_alloc */
*output = (AVSfield_float *)
AVSdata_alloc("field 2D 2-space 1-vector uniform float", dims);

/* check to ensure field allocation successful if not return */
if (*output == NULL)
{
AVSerror("Allocation of output field failed.");
return(0);
}



/* Compute the sobel algorithm                                           */

for (i=0; i<MAXX(input); i++)
for (j=0; j<MAXY(input); j++)
{
/* check to see if the current pixel is an edge pixel */
if ( (i==0) || (i==MAXX(input)-1) ||
     (j==0) || (j==MAXY(input)-1)   )
```

```
{
/* if so redefine the pixel to 0.0 */
I2D(*output, i, j) = 0.0;
}
/* for non edge pixels perform the filtering process*/
else
{
/* calculate the X value */
  X =    ( I2D(input, i-1, j-1)       +
    I2D(input, i,   j-1) * 2.0 +
    I2D(input, i+1, j-1)       -
   I2D(input, i-1, j+1)       -
    I2D(input, i,   j+1) * 2.0 -
    I2D(input, i+1, j+1)        );

/* calculate the Y value */
Y =    ( I2D(input, i+1, j+1)       +
    I2D(input, i+1, j  ) * 2.0 +
    I2D(input, i+1, j-1)       -
   I2D(input, i-1, j+1)       -
    I2D(input, i-1, j  ) * 2.0 -
    I2D(input, i-1, j-1)        );

/* redefine the pixel, using the algorithm, to P' */
I2D(*output, i, j) = ( sqrt( X*X + Y*Y ) );
}
}


/* <---- END OF USER-SUPPLIED CODE SECTION #3                            */
return(1);
}

/* ********************************************************************/
/* Initialization for modules contained in this file.                 */
/* ********************************************************************/
static int ((*mod_list[])()) = {
edge_detection_desc
};
#define NMODS (sizeof(mod_list) / sizeof(char *))

AVSinit_modules()
{
AVSinit_from_module_list(mod_list, NMODS);
}

/* ----> START OF USER-SUPPLIED CODE SECTION #4 (SUBROUTINES, FUNCTIONS, UTILITY ROUTINES)*/
/* <---- END OF USER-SUPPLIED CODE SECTION #4                            */
```

F.2 Source Code (makefile)

```
#
# Makefile for edge
#

AVS_PATH=/usr/avs
INC_FILE=$(AVS_PATH)/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(AVS_PATH)/lib
INC = $(AVS_PATH)/include
BASELIBS=-lgeom -lutil -lm $(LASTLIBS)
CFLAGS = -I$(INC) $(AOPTCFLAGS) $(LOCAL_CFLAGS) $(G)
CFLOWLIBS=-L$(AVS_LIBS) -lflow_c $(BASELIBS)
CSIMLIBS=-L$(AVS_LIBS) -lsim_c $(BASELIBS)

edge: edge.o
$(CC) $(CFLAGS) -o edge edge.o $(CFLOWLIBS)

edge.o: edge.c
$(CC) $(CFLAGS)  -c edge.c
```