Table of Contents

1. Introduction

This course has been developed by the International AVS Centre (Manchester) in collaboration with the Manchester Visualization Centre, at the University of Manchester.

1.1. What is AVS/Express?

AVS/Express is an object-oriented visual development tool, that enables you to build reusable application components and sophisticated applications. It provides hundreds of application components for visualizing, analysing, manipulating, and displaying data. Further it provides an open interface that allows users to write and intergrate their own components into the application.

1.2. Who is this workbook aimed at?

This workbook and the accompanying transparencies, are aimed at those users who have a need to extend the functionality of the AVS/Express environment, so that they may visualize their data.

1.3. Are there any prerequisites?

The course is designed for those users who are already familiar with using the AVS/Express environment, and assumes that the attendees have at least some programming experience (C or C++).

2. About the Workbook

2.1. The Development Environment

To perform the exercises in this workbook and to develop your own modules for Express requires both C and C++ compilers. On UNIX machines the system compilers are used. On Windows machines Express requires Visual C/C++ of which version 5 is neccessary for use with Express 4.0, which is the current version of Express at time of writing.

Express projects create a lot of files during compilation and so it is suggested that users create a project directory from which to run Express before they save/compile a project. This also makes distribution of Express projects much more manageable.

2.2. Structure of the Exercises

The user is presented with an outline of the task containing information for the task to be completed. This is followed by hints and suggestions about how the user should set about completing the task, rather than black and white instructions to encourage the user to think about what they are doing. The exercises assume a reasonable knowledge of how to use Express and its general structure. They are designed to be performed in order as each builds on the experience gained from the previous exercise.

2.3. Solutions and Datafiles

Any solutions that are not presented as part of the exercises can be found in Chapter 8.
Due to the nature of programming these may not be exactly the same as the solutions produced by the user and are therefore provided as a guide and example of a possible solution.

Any data files are those present in the Express installation unless otherwise stated.

An Express project containing all the code built in the workbook can be found on the IAC web site at:

```
http://www.iavsc.org/express/courses/modwrite-xp/
```

3. The Add Module Tool

3.1. Exercise: AddFloats Module (Add Module Tool)

Using the **Add Module** tool create a module, **AddFloats**, that takes two *floats* as inputs and writes their sum to a *float* output. The module calls a C function method called *addflts* that runs in the *user* process. The source code for the method should reside in *addfloats*.c.

Select *Workspace_1* in *Library_Workspaces*. Open the **Add Module** tool from the Object menu.

Using the **Add Module** tool, rename the module and add the appropriate type of method. Give the method the correct name and make sure it runs when the module is instanced. Press **Done** and **Next** to finish adding methods.

Add the two input parameters and give them appropriate names. Make their input ports visible. Set the method behaviour to **read**, **req** and **notify** for each of these parameters. Repeat the process for the output parameter but the method dependency should be simply **write** access. Press **Done** and **Next** to finish adding parameters.

Set the compilation details, choosing the correct process and source code file.

Select **Edit Source** to generate skeleton code. Edit the source to add the two inputs and set the output to this value. It is not necessary to try and understand exactly what the code does but you should get a general overview of how Express accesses module parameters using its C API.

Save the source and choose **Compile** in the **Add Module** tool. When compilation has completed close **Add Module** and save the project. Test the module in an application with appropriate values.

3.2. Exercise: AddFloats Module (Network and Object Editor)

Create the same **AddFloats** module as before (though call it **AddFloats2**) but using drag'n'drop techniques to build the module in the **Network Editor**. Then use the **Object Editor** to set process, source files and method dependencies.

Open a *ScratchPad* application.

From the *Standard_Objects* library drag down an *om_module* which contains a single C function method. Open the module and rename and set the method appropriately.

Drag and drop *float* primitives from the same library to create the parameters, rename and export the ports to match the previous module.

Drag and drop the module into *Workspace_1*.

Select the module and open the **Object Editor**. From the 'sorce code' page set the process and source code (same as before).

For each parameter choose the 'method' page and set the method dependencies.

You do not need to edit or compile the source as the module is using the same function as before. Save the project with the new addition. Test the new module.

3.3. Exercise: Points2DDist Module (using groups in modules)

Create a **Point2D** group with members **x** and **y**. Use the **NE** properties to change the port colour for **Point2D**. Write a C++ module **point2Ddist** to calculate the distance between two points. The method code should reside in *point2Ddist.cxx* and run in the *user* process.

Open *ScratchPad*.

From *Standard_Objects* drag down a *group* object and rename. Use the **Properties** tool to set *NEcolor0* to `0x0000ff` (blue) and *NEcolor1* to `0x808000 (yellow).` Add two *floats* to the group to be **x** and **y**. Place the new datatype in a Workspace.

Create the module with a *omx* (C++) *method* by whatever means you prefer. Note, do not set the name of the function in the method as C++ uses the name of the module to generate the function name. Add two *point2D* object as inputs, change their reference mode to *by ref(&)*. Add a *float* as output. Edit the source code adding the following lines:

```
// assumes inputs called p1 and p2, output dist

float xd, yd;

xd = p1.x - p2.x;
yd = p1.y - p2.y;
dist = sqrt( xd*xd + yd*yd);
```

Save the source and project and then recompile. Open an application and test out the module.

Examine the *v/templ.v* file to get an idea of how the modules you have generated are defined. V code is covered in more detail later in the course.

4. Using the V Definition Language

4.1. Exercise: Creating and Examining Primitives (in the VCP)

Learn how to use V code to directly create, modify and examine primitive objects using the **VCP**.

This and the following exercises show only the text you need to type into the **VCP**, you will see in the **VCP** your location e.g.,`OM(Root)`, followed by the prompt `->`.

In the **NE** open a *DefaultApplication*. In the **VCP** move through the hierarchy to the application:

```
DefaultApplication {
```

Create some primative objects:

```
int a;
int b=4;
a c;
```

Notice that they appear in the **NE**. Type in commands to examine the objects:

```
$print
$int obj
$float obj
$list
```

Set some values:

```
a=1;
c=a+b;
```

Examine the objects again. Change some values:

```
a=2;
```

Notice that c doesn't change. Make it reference a and b:

```
c=>a+b;
a=3;
b=5;
```

Remember to examine the object after each change. Delete objects:

```
-a;
```

4.2. Exercise: Creating and Examining Groups (in the VCP)

Learn how to use V code to directly create, modify and examine group objects using the **VCP**.

Create some groups:

```
group grp1;
group grp2 {
    float x=7.2;
};
grp2 grp3;
```

Examine the groups you have created. Modify a group object:

```
grp1 {
    int y=4;
};
```

There are different ways of accessing groups:

```
grp2.x=10.4;
grp1 {
    y=2;
};
```

Groups cannot be connected directly like primitives:

```
group grp4=>grp1;
```

will return an error. Group references must be used instead. A group reference specifies the minimum number and type of object that a group must have for a connection to be made. The group reference will attain any additional members that the groups has:

```
group &grpref1 {
    float a;
};
group grpa {
    float a=4;
    int b=2;
};
grpref1=>grpa;
```

Examine the groups carefully and make sure you understand what is happening.

Objects can access other objects at different hierarchical levels using <-. to move up a level and .object to move down.

```
int a=2;
```

```
int b;
group grp4{
    int a1;
    int b2=7;
};

grp4.a1=><-.a;
b=grp4.b2;
```

4.3. Exercise: Creating and Examining Arrays (in the VCP)

Learn how to use V code to directly create, modify and examine array objects using the **VCP**.

Creating primative arrays:

```
float a[2];
float b[]={1,3,2};
a c;
```

As before examine the objects as you make changes to see what's happening. To examine arrays use:

`$get_array` *`obj`*

Setting the values:

```
float x=3.5;
float y=7.8;
a={x,y};
x=6.3;
```

Making an array reference its values:

```
a=>{x,y};
y=8.3;
```

You can use arrays of group references as well:

```
Mesh m1;
Mesh m2;
Mesh &marr[2]=>{m1, m2};
```

4.4. Exercise: Animated Filename Module (V code module)

Use the *loop* module to create a *macro*, **AnimFileName**, that generates a **filename** from supplied **prefix** and **postfix** strings as well as the number from the **loop** module, i.e., `prefix###postfix`.

Open a *ScratchPad* and try using the **VCP** to create a *group* called **AnimFileNameGen**. Add two *string* parameters for the **prefix** and **postfix**, then add an *int* for the file **number**.

To create the **filename** add another *string* to the group, set the strings value using the built in *str_format* function:

```
=>str_format("%s%.3d%s", prefix, number, postfix)
```

because a reference is used **filename** will change each time a parameter changes. The '`.3`' tells the function to pad the number with zeros to 3 digits.

Set the ports to be exported properly. Try to do this for a parameter with the **NE** and then examine the object in the **VCP**. Use the **VCP** to set the export property on the remaining parmeters.

Drag down a *loop* module, found in *Main.Data_IO*, and connect its output to **AnimFilenameGen**. Cut and paste these modules into a *macro* called **AnimFilename**. Store finished macro in a Workspace and save.

In an *application* test the macro is working by setting values for the parameters.

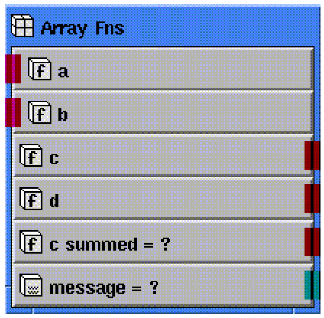
4.5. Exercise: Calculating Values form Arrays (V code module)

Create the V code module **ArrayFns** in the **VCP** to demonstrate how to constuct new arrays from parts of other arrays and how to calculate values from arrays using V's inbuilt functions. Taking two *float* arrays as inputs construct a new array from the first four values of each. Calulate the *sum* and *mangnitude* of this array. Generate an array of the *maximum* values of the input arrays.

Open an application in the **NE** and then move to this in the **VCP**.

Create two arrays of floats as data for the module give each at least four values.

Create the group **ArrayFns** in the **VCP** so that it look like this:



in the **NE**. Connect the data arrays to **ArrayFns**' inputs.

To create the array **c** from the first four values of each input set it to:

```
=>{a[0:3],b[0:3]};
```

Examine the array **c**.

Calulate its *sum* by setting to appropriate member of **ArrayFns** to:

```
=>sum(c);
```

Set array **d** to hold to the *maximum* values of the inputs using

```
=>{max_array(a), max_array(b)};
```

this shows how functions can be nested.

The *str_format* function, which has a syntax similar to C's *printf* function, can be used to say something about the data created.

```
=>str_format("The sum of the %d values in c is %f",
array_size(c), c_summed);
```

Examine carefully the V code for the module and then change the data values to see if the calculated values respond as you would expect.

5. Integrating C and C++ Code

5.1. Julia Module (creating Node_Data objects)

Create an *om_module* called **Julia** that calls a function *julia_set* located in *julia_set.c*. The module excepts parameters to set the Mandelbrot point at which the set is to be generated, the maximum depth of recursion and the x and y resolution of the data created. The module will then output a 2D uniform *Field* containing the Julia set..

> The *Field_Unif* template can be found in the *Accessories* library under *Field_Schema*.
>
> When you have generated the code you will need to set several varibles before they are used such as dims, nspace and points for the *Mesh* part and then ncomp, veclen and data_type for the *Node_Data* part. Read the skeleton code carefully as it gives hints where variables and data need to be set.
>
> See **Chapter 8 Solutions** for source code.
>
> Test the module with values of your choice by connecting i to a *Uviewer2D* via a *DataObject*.

Create a macro **JuliaSet** that encapsulates the module with a *DataObject* module to create a renderable object.

> Build your macro in *ScratchPad*. The *macro* object can be found in *Standard_Objects*.
>
> Use *link* objects to make the macro more stable and usable. Place the finished macro in a Workspace and remember to save the project.

5.2. Cube Module (creating Mesh objects)

Create a **Cube** module that outputs a mesh containing the geometry of a wireframe cube that has sides of unit length. The method calls the C function *create_cube* located in *create_cube.c*.

> The basic *Mesh* object can be found in the *Accessories* library under *Field_Schema.Mesh_Schema*. Make the module run when instanced.
>
> When you have generated the code you will again need to set several varibles before they are used, the skeleton code gives indications where variables and data need to be set.
>
> See **Chapter 8 Solutions** for source code.
>
> Test the module in a 3D application, use a *DataObject* from *Accessories.Graph-*

ics.Modules to convert the geometry into a renderable object.

Extend the **Cube** module by adding 3 inputs to scale the cube (cuboid) along its axes.

Create a *macro* containing the **Cube** module and *DataObject* using a *link* object aswell.

6. UI Kit

6.1. Exercise: Julia User Interface

Create a user interface for the **Julia** module created previously to allow the user to change the threshold range.

For the UI to appear as you build it you need to use an executing enviroment such as *Application* rather than *ScratchPad*.

Create a macro object called **JuliaUI** to contain the UI objects.

Change to the *User_Interface* library and from *Containers* drag down a *UImod_panel*. This will make sure that the UI appears in the correct place when using a *SingleWindowApp* for example. Change the name of the panel to Julia, this will be used by module lists in applications.

Drag down a *UIlabel* and a *UIfield* rename the UI field to **mandelx**. Connect the green *parent* inputs to the panel. Notice how Express adds each widget below the previous. Repeat this for the **mandely**, **resx** and **resy** parameters

Drag down a *UIslider* from *Widgets*. Change its name to **max_iters** Open it and set its range (*min* and *max*) to 0-256 and make its mode 'integer'so that the UI is more user friendly.

Export the widget values from the macro.

UI macros must think that they are in the express process so place the **JuliaUI** in an empty Workspace and make sure that the *process* property for the Workspace is set the 'express'. If you wish to recompile your modules that are in the 'user' process make sure that this Workspace is not selected.

Drag the **JuliaSet** macro that you generated before into a *ScratchPad*. Add **JuliaUI** to this connecting the parameter outputs to the inputs on the **Julia** module. Replace the improved **JuliaSet** macro in the Workspace and save the project.

Test the macro within an application.

Optional: try improving the UI by tidying it up, from the *UI_Builder* menu select *Layout_Editor*. Widgets can now be moved around by selecting and dragging with the third mouse button. Remember to turn the *Layout_Editor* off when you have finished.

7. Project Organization

7.1. Exercise: Overlay Module (IAC Project Organisation)

Create a project which provides a user macro **Overlay** that takes two images (*Mesh_Unif+Node_Data*) of equal size and overlays the second image over the first using a user specified ARGB value as being transparent. This will involve creating an **overlay_image** module, an **OverlayUI** UI macro, the **Overlay** user macro and an example application.. The project should be put together under the **IAC Project Organisation GuideLines**.

Read the **Project Organisation GuideLines** at:

```
http://www.iavsc.org/express/pog.html
```

Create a new directory and dowload the **IAC** project base into this from:

```
http://www.iavsc.org/express/iac.tar.gz
```

Uncompress this using:

```
gunzip iac.tar.gz
tar xvf iac.tar
```

You may wish to download an install the example **threshold** project the help get an idea of the organisation of the files.

```
http://www.iavsc.org/express/thresh.tar.gz
```

First thing to do is to examine the field that **ReadImage** creates using the **VCP** or **Object Editor**. Information will be required about the how the *points* array is filled and *node_data[].values* datatype, also notice how many values are automatically generated from others (such as the *coordinates.values*) and how the dimensions of some arrays are defined by other member objects which if set will change the array size.

The only parameter needed is an array containing the ARGB values to be used as the transparent colour in the second image, using the **POG** parameter block method can therefore be dispensed with. This can be done by checking that the *nnodes* is non-zero in each.

Here are some hints about coding the module.

Firstly the module should make sure that it has two valid images as inputs; group references will cause the module to be notified if they are connected to an appropriate group even though that group may not be valid itself.

Make sure that any values that specify the dimensions of an array are set before you try to access the array. Also when generating an array if the array is of type *prim* you should set it's type first, in C use e.g.,:

```
OMret_typed_array_ptr(array_id,
OM_GET_ARRAY_WR,OM_TYPE_XXXX, &array_size)
```

and in C++ use e.g.,:

```
array.ret_typed_array_ptr(OM_GET_ARRAY_WR,
OM_TYPE_XXXX, &array_size)
```

The output *node_data* can be composed using the following code

```
// overlay image2 on image1 in compos
for(i=0; i<compos.nnodes; i++)
    for(j=0; j<4; j++)
        compos_node_data[i*4+j] =
        (image2_node_data[i*4+j]==mask_arr[j]) ?
        image1_node_data[i*4+j]:image2_node_data[i*4+j]
;
```

Once the module is complete test it with images **mvc.gif** and **iac.gif** located at:

`http://www.iavsc.org/express/courses/modwrite-xp/`

the background colour is `(0,0,0,255).`

Constuct a simple UI using *UIsliders*, setting the parametrs to be compatible with the image data (the *mode* parameter can be set to the correct dataype). The float values from the sliders will need the be combined into an array using *mlink* before being connected to the overlay transparency parameter.

The user macro can now be created, remember to use *link*s and to add a *DataObject*

Finally create an example application using the images provided..

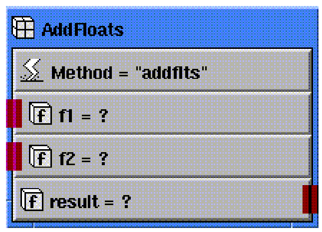
IMPORTANT: do not use *Project.Save* as this will destroy the V file structure used by the **IAC**. When you have completed objects use *Objects.Save_Objects* to save the individual objects then edit them to create the `xxxx_mods.v` and `xxxx_macs.v` files that contain the modules and macros.

8. Solutions

8.1. The Add Module Tool

1. AddFloats Module (Add Module Tool)

The **AddFloats** module built should look like:



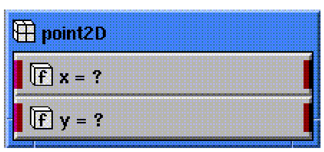
The C source code for **addfloats.c**:

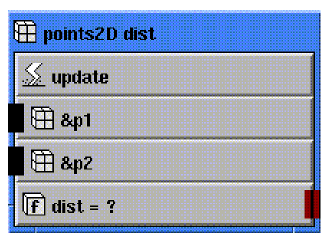
```
#include "user.h"

int
addflts(OMobj_id AddFloats_id, OMevent_mask event_mask,
int seq_num)
{
   /**********************/
   /*  Declare variables  */
   /**********************/
   double  f1;
   double  f2;
   double  result;

   /**********************/
   /*  Get input values   */
   /**********************/
   /* Get f1's value */
   if (OMget_name_real_val(AddFloats_id,
OMstr_to_name("f1"), &f1) != 1)
      f1 = 0.0;

   /* Get f2's value */
   if (OMget_name_real_val(AddFloats_id,
OMstr_to_name("f2"), &f2) != 1)
      f2 = 0.0;
```

```
   /***********************/
   /* Function's Body     */
   /***********************/
   ERRverror("",ERR_NO_HEADER | ERR_INFO,"I'm in func-
tion: addflts generated from method: AddFloats.Meth-
od\n");


   /***********************/
   /*  Set output values  */
   /***********************/
   /* Set result's value */
   result=f1+f2;
   OMset_name_real_val(AddFloats_id, OMstr_to_name("re-
sult"), result);
   return(1);
}
```

2. Points2DDist Module (using groups in modules)

The **point2D** group and **poin2Ddist** module are simply:





The C++ source code for **points2d_dist.cxx** is:

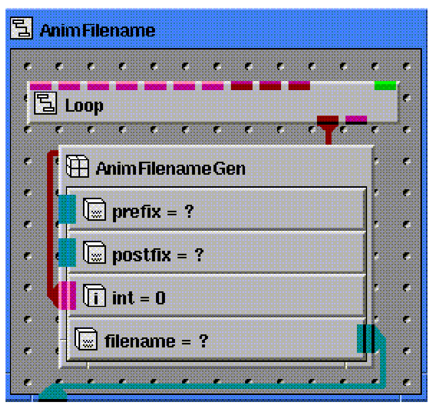
```
#include <math.h>
#include "user.h"

int
WORKSPACE_2_points2D_dist::update(OMevent_mask
event_mask, int seq_num)
{
   // p1 (point2D read req notify)
   // p2 (point2D read req notify)
   // dist (OMXfloat write)

   /***********************/
   /* Function's Body     */
   /***********************/
```

```
  float xd, yd;

  xd=p1.x-p2.x;
  yd=p1.y-p2.y;

  dist=sqrt(xd*xd-yd*yd);

   ERRverror("",ERR_NO_HEADER | ERR_PRINT,"I'm in meth-
od: WORKSPACE_2_points2D_dist::update\n");

   // return 1 for success
   return(1);
}
```

8.2. Using the V Definition Language

1. Animated Filename Module (V code module)

The **AnimFilename** Macro looks like:



8.3. Intergrating C an C++

1. JuliaModule (creating Node_Data objects)

This is the full source code for **julia_set.c** , additions are in **bold**.

```
#include "/home/cluster/zzcgujp/projects/modwrite1/./
user.h"

int julia_set(OMobj_id julia_id, OMevent_mask
event_mask, int seq_num)
{
```

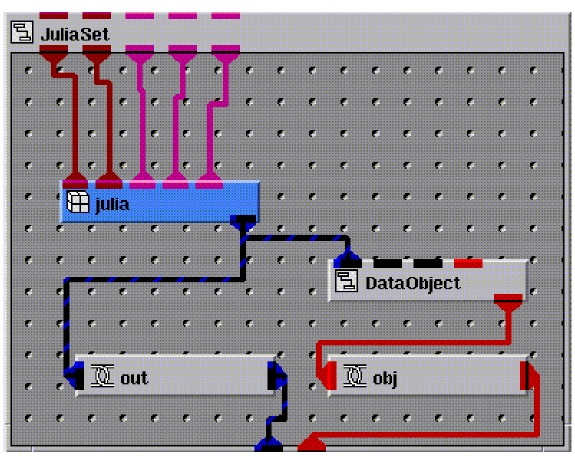
```
/**********************/
/*  Declare variables  */
/**********************/
double  mandelx;
double  mandely;
int  max_iter;
int  resx;
int  resy;
OMobj_id out_id;
int out_ndim, *out_dims, out_nspace, out_npoints;
float *out_points;
int out_ncomp, out_comp_count, out_veclen;
int    out_data_type, out_ndata;
unsigned char  *out_data; /* outputing byte data */

int i, j, k, index;
double dx, dy;
double zx, zy;
double tmp;
double posy;
int max_k;

 /**********************/
 /*  Get input values   */
 /**********************/
 /* Get mandelx's value */
if (OMget_name_real_val(julia_id, OMstr_to_name("man-
delx"), &mandelx) != 1)
   mandelx = 0.0;

 /* Get mandely's value */
if (OMget_name_real_val(julia_id, OMstr_to_name("man-
dely"), &mandely) != 1)
   mandely = 0.0;

 /* Get max_iter's value */
if (OMget_name_int_val(julia_id,
OMstr_to_name("max_iter"), &max_iter) != 1)
   max_iter = 0;

 /* Get resx's value */
if (OMget_name_int_val(julia_id,
OMstr_to_name("resx"), &resx) != 1)
   resx = 0;

 /* Get resy's value */
if (OMget_name_int_val(julia_id, OMstr_to_name("re-
sy"), &resy) != 1)
   resy = 0;
```

```
    /**********************/
    /* Function's Body     */
    /**********************/
  ERRverror("",ERR_NO_HEADER | ERR_INFO,"I'm in func-
tion: julia_set generated from method: julia.gener-
ate\n");

  if (max_iter<256) {
    max_k = max_iter;
  } else {
    max_k = 255;
  }

  dx = 4 / (double)(resx-1);
  dy = 4 / (double)(resy-1);

  index=0;

    /**********************/
    /*  Set output values  */
    /**********************/
     /* Set out uniform mesh */

    /*  Get mesh id */
  out_id = OMfind_subobj(julia_id, OMstr_to_name("out"),
OM_OBJ_RW);

  /* Set mesh dimensionality, out_ndim can be 1,2 or 3 */
  out_ndim=2;
  FLDset_ndim (out_id, out_ndim);

    /* Set mesh dims array */
  out_dims = (int *)ARRalloc(NULL, DTYPE_INT,
     out_ndim, NULL);

  /*** fill in dims array with your values ***/
  out_dims[0]=resx;
  out_dims[1]=resy;

  FLDset_dims (out_id, out_dims);
  if (out_dims)
    ARRfree((char *)out_dims);

    /* Set mesh nspace, out_nspace can be 1,2 or 3 */
  out_nspace=2;
  FLDset_nspace (out_id, out_nspace);

    /* Set mesh extents */
  out_npoints = 2*out_nspace;
```

```
  out_points = (float *)ARRalloc(NULL, DTYPE_FLOAT,
 out_npoints, NULL);

 /*** fill in points array with values for 2 points: low
left and high right corners of the mesh ***/

  out_points[0]=-2;
  out_points[1]=-2;
  out_points[2]=2;
  out_points[3]=2;

  FLDset_points (out_id, out_points, out_npoints,
 OM_SET_ARRAY_FREE);
  /* set  out node data */

   /* Get field id */
 out_id = OMfind_subobj(julia_id, OMstr_to_name("out"),
OM_OBJ_RW);

  /* Set number of node data components */
  out_ncomp=1;
  FLDset_node_data_ncomp (out_id, out_ncomp);

   /* For each node data component set veclen, type and
data arry itself */
  for (out_comp_count=0; out_comp_count < out_ncomp;
out_comp_count++) {

    /* Set veclen, assign out_veclen before next call */
    out_veclen=1;
    FLDset_node_data_veclen
(out_id,out_comp_count,out_veclen);

    /* Set data array */
    /* data_type should be set to one of the following:
       DTYPE_BYTE, DTYPE_CHAR, DTYPE_SHORT,
       DTYPE_INT, DTYPE_FLOAT, DTYPE_DOUBLE) */

    out_data_type = DTYPE_BYTE;

    /* allocate out_data array first */
    /* assume float array and out_ndata is set to number
of nodes */
    out_ndata=resx*resy;
    out_data = (unsigned char *)ARRalloc(NULL,
out_data_type,out_veclen*out_ndata, NULL);
    for( j=0; j<resy; j++ )
    {
        posy = (double)(dy * j) - 2;
        for( i=0; i<resx; i++ )
```

```
        {
          zy = posy;
          zx = (double)(dx * i) - 2;

          for( k=0; k<max_k; k++ )
          {
            tmp = (zx*zx) - (zy*zy) + mandelx;
            zy = 2 * zx * zy + mandely;
            zx = tmp;

            if( ((zx*zx) + (zy*zy)) > 4 )
              break;
          }

          out_data[index++] = k;
        }
      }

   FLDset_node_data (out_id, out_comp_count, out_data,
out_data_type,
      out_ndata*out_veclen, OM_SET_ARRAY_FREE);

    /*  Other useful calls:
FLDset_node_data_id()
FLDset_node_null_data()
FLDset_node_data_label()
*/
  }

  return(1);
}
```

The **JuliaSet** macro should look like:



2. Cube Module (creating Mesh objects)

The full C source code for **cube.c**, additions are in **bold**.

```
#include "user.h"

int
create_cube(OMobj_id cube_id, OMevent_mask event_mask,
int seq_num)
{
/***********************/
/*  Declare variables  */
/***********************/
OMobj_id geom_id;
int geom_nspace, geom_nnodes, geom_size;
int  geom_nsets, geom_set_count, geom_ncells;
int  geom_cell_nnodes, *geom_node_connect;
OMobj_id geom_cell_set;
float *geom_coord;
double  scale_x;
double  scale_y;
double  scale_z;

int i,j,k, coord_index;

/***********************/
/* Function's Body     */
/***********************/
ERRverror("",ERR_NO_HEADER | ERR_INFO,"I'm in function:
create_cube generated from method: cube.create\n");


/***********************/
/*  Set output values  */
/***********************/
/* set  geom mesh */

/*  Get mesh id */
geom_id = OMfind_subobj(cube_id, OMstr_to_name("geom"),
OM_OBJ_RW);

/* Set mesh nnodes */
geom_nnodes=8;
FLDset_nnodes (geom_id, geom_nnodes);

/* Set mesh nspace */
geom_nspace=3;
```

```
FLDset_nspace (geom_id, geom_nspace);

/* Set mesh coordinates */
/* First allocate geom_coord array */

geom_coord = (float *)ARRalloc(NULL, DTYPE_FLOAT,
                        geom_nspace*geom_nnodes, NULL);

/*** Fill in geom_coord array with X[,Y,Z] values at each
node ***/
for(k=0; k<2; k++)
    for(j=0; j<2; j++)
        for(i=0; i<2; i++)
        {
            coord_index=i*3+j*6+k*12;
            geom_coord[coord_index]=i;
            geom_coord[coord_index+1]=j;
            geom_coord[coord_index+2]=k;
        }

FLDset_coord (geom_id, geom_coord,
geom_nspace*geom_nnodes, OM_SET_ARRAY_FREE);

/* Set_number of cell sets to 0 initially, the real
number of cell_sets should be assigned to geom_nsets var-
iable */
FLDset_ncell_sets(geom_id, 0);
geom_nsets=1;

/* For each cell set set information about cells */
for (geom_set_count=0; geom_set_count<geom_nsets;
geom_set_count++) {

/* add cell set, the name can be one of the following:
Point, Line, Polyline, Tri, Polytri, Polhedron,
Tet, Hex, Prism, Pyr,
*/
FLDadd_cell_set(geom_id, "Line");

/* Get cell set id */
FLDget_cell_set(geom_id, geom_set_count,
&geom_cell_set);

/* Set number of cells */
geom_ncells=12;
FLDset_ncells(geom_cell_set, geom_ncells);

/* Set node connectivity list */
/* First allocate geom_node_connect */
```

```
FLDget_cell_set_nnodes(geom_cell_set,
&geom_cell_nnodes);
geom_node_connect = (int *)ARRalloc(NULL, DTYPE_INT,
geom_ncells*geom_cell_nnodes, NULL);

/*** fill in  geom_node_connect array with node indecies
for each cell ***/
    i=0;
    geom_node_connect[i++]=0;
    geom_node_connect[i++]=2;
    geom_node_connect[i++]=2;
    geom_node_connect[i++]=3;
    geom_node_connect[i++]=3;
    geom_node_connect[i++]=1;
    geom_node_connect[i++]=1;
    geom_node_connect[i++]=0;
    geom_node_connect[i++]=4;
    geom_node_connect[i++]=5;
    geom_node_connect[i++]=5;
    geom_node_connect[i++]=7;
    geom_node_connect[i++]=7;
    geom_node_connect[i++]=6;
    geom_node_connect[i++]=6;
    geom_node_connect[i++]=4;
    geom_node_connect[i++]=1;
    geom_node_connect[i++]=5;
    geom_node_connect[i++]=4;
    geom_node_connect[i++]=0;
    geom_node_connect[i++]=7;
    geom_node_connect[i++]=3;
    geom_node_connect[i++]=2;
    geom_node_connect[i++]=6;

FLDset_node_connect(geom_cell_set, geom_node_connect,
geom_ncells*geom_cell_nnodes, OM_SET_ARRAY_FREE);


/***  NOTE: for POLY cell_sets such as Polyline, Polytri
and Polhedron use:
FLDset_npolys() instead of
FLDset_ncells()
FLDset_poly_connect()    instead of
FLDset_node_connect()
***/
}
return(1);
}
```

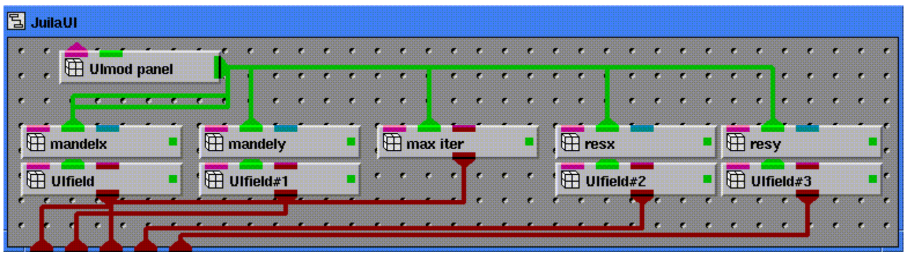
The further addtions to add scaling are:

```
:
:
/***********************/
/*  Get input values   */
/***********************/
/* Get scale_x's value */
if (OMget_name_real_val(cube_id,
OMstr_to_name("scale_x"), &scale_x) != 1)
      scale_x = 0.0;

/* Get scale_y's value */
if (OMget_name_real_val(cube_id,
OMstr_to_name("scale_y"), &scale_y) != 1)
scale_y = 0.0;

/* Get scale_z's value */
if (OMget_name_real_val(cube_id,
OMstr_to_name("scale_z"), &scale_z) != 1)
scale_z = 0.0;
:
:
:
/*** Fill in geom_coord array with X[,Y,Z] values at each
node ***/
for(k=0; k<2; k++)
    for(j=0; j<2; j++)
        for(i=0; i<2; i++)
        {
            coord_index=i*3+j*6+k*12;
            geom_coord[coord_index]=i*scale_x;
            geom_coord[coord_index+1]=j*scale_y;
            geom_coord[coord_index+2]=k*scale_z;
        }
:
:
```

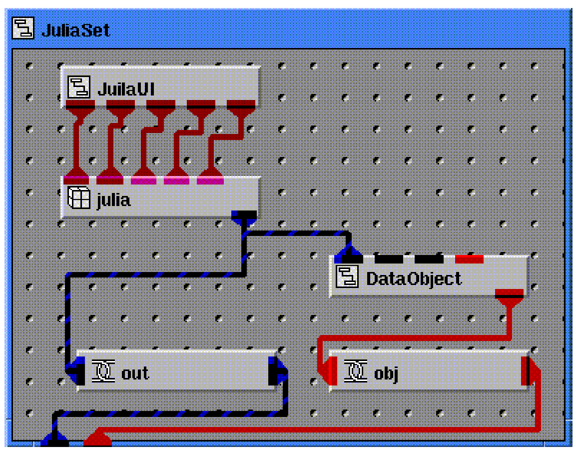
8.4. UI Kit

1. Julia User Interface

The **JuliaUI** should look like:



The **JuliaSet** macro should now contain the UI:



8.5. Project Organisation

1. Exercise: Overlay Module (IAC Project Organisation)

The **overlay_images** module should look similar to this:

The **over_mods.v** file:

```
flibrary OverlayMods <build_dir="iac_proj/overlay",
                      process="express",
                      libdeps="GD FLD",
                      cxx_hdr_files="fld/Xfld.h",
                      out_hdr_file="gen.h",
                      out_src_file="gen.cxx"> {

  module overlay_images<src_file="overlay_images.cxx">
{
      int mask<NEportLevels={2,0}>[4];
      Mesh_Unif+Node_Data &image1<NEportLevels={2,0}>;
      Mesh_Unif+Node_Data &image2<NEportLevels={2,0}>;
      Mesh_Unif+Node_Data compos<NEportLevels={0,2}>;

      cxxmethod+notify_inst+req overlay(
         mask+notify+read+req,
         image1+notify+read+req,
         image2+notify+read+req,
         compos+write);
   };
};
```

The C++ source code (**overlay_images.cxx**) for which is:

```
#include "iac_proj/overlay/gen.h"
#include "fld/Xfld.h"

int Overlay_OverlayMods_overlay_images::over-
lay(OMevent_mask event_mask, int seq_num)
{
  // check images are valid objs
```

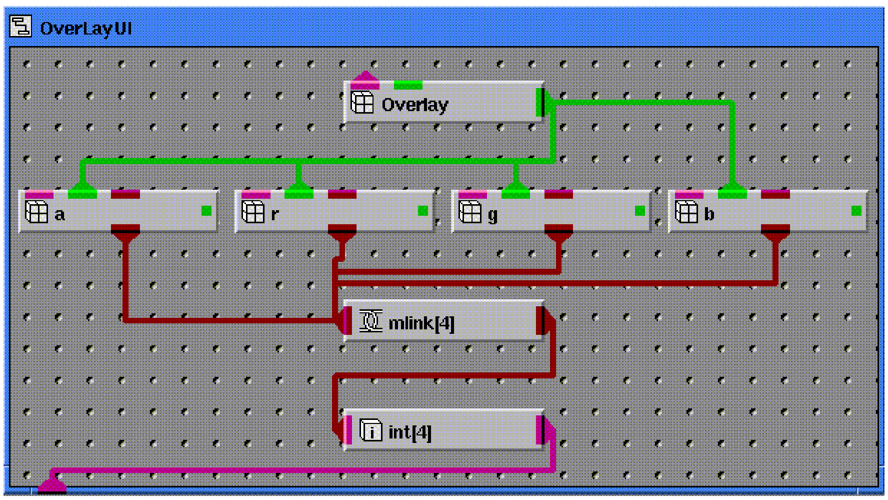
```
  if( !(image1.nnodes && image2.nnodes))
    return 1;

  // mask (OMXint_array read req notify)
  int mask_size;
  int *mask_arr;

  // image1 (Field_Unif read req notify)

  // image1.ndim (int)
  // image1.dims (int [])
  int *image1_dims = (int
*)image1.dims.ret_array_ptr(OM_GET_ARRAY_RD);
  // image1.nspace (int)
  // image1.npoints (int)
  // image1.points (float [])
  // image1.nnodes (int)
  // image1.nnode_data (int)
  int  image1_data_size, image1_data_type;
  // image1.node_data[image1_data_comp].veclen (int)
 // image1.node_data[image1_data_comp].values (char [])
  char *image1_node_data = (char
*)image1.node_data[0].val-
ues.ret_array_ptr(OM_GET_ARRAY_RD, &image1_data_size,
&image1_data_type);

  if (!image1_node_data) {
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay:
can't access image1 node_data\n");
    return 1;
  }

  // image2 (Field_Unif read req notify)

  // image2.ndim (int)
  // image2.dims (int [])
  int *image2_dims = (int
*)image2.dims.ret_array_ptr(OM_GET_ARRAY_RD);
  // image2.nspace (int)
  // image2.npoints (int)
  // image2.points (float [])
  // image2.nnodes (int)
  // image2.nnode_data (int)
  int  image2_data_size, image2_data_type;
  // image2.node_data[image2_data_comp].veclen (int)
 // image2.node_data[image2_data_comp].values (char [])
  char *image2_node_data = (char
*)image2.node_data[0].val-
ues.ret_array_ptr(OM_GET_ARRAY_RD, &image2_data_size,
&image2_data_type);
```

```
  if (!image2_node_data){
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay:
can't access image2 node_data\n");
    ARRfree((char *)image1_node_data);
    return 1;
  }

  // check images are both argb
  if( image1.node_data[0].id!=669 ||
image2.node_data[0].id!=669  ){
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay: im-
ages must be argb\n");
    return 1;
  }

  // check images are the same size
  if( image1_dims[0]!=image2_dims[0] ||
image1_dims[1]!=image2_dims[1] ){
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay: im-
ages must be of identical dims\n");
    return 1;
  }

  // compos (Field_Unif write)

  // compos.ndim (int)
  // compos.dims (int [])
  compos.ndim=2;
  int *compos_dims = (int *)com-
pos.dims.ret_array_ptr(OM_GET_ARRAY_WR);
  // compos.nspace (int)
  compos.nspace=2;
  // compos.npoints (int)
  compos.npoints=2;
  // compos.points (float [])
  float *compos_points = (float *)com-
pos.points.ret_array_ptr(OM_GET_ARRAY_WR);
  // compos.nnodes (int)
  // compos.nnode_data (int)
  int  compos_data_size, compos_data_type;
  // compos.node_data[compos_data_comp].veclen (int)
  // compos.node_data[compos_data_comp].values (char [])

  // set compos dims
  compos_dims[0]=image1_dims[0];
  compos_dims[1]=image1_dims[1];
  compos.nnodes=compos_dims[0]*compos_dims[1];

  // set points
```

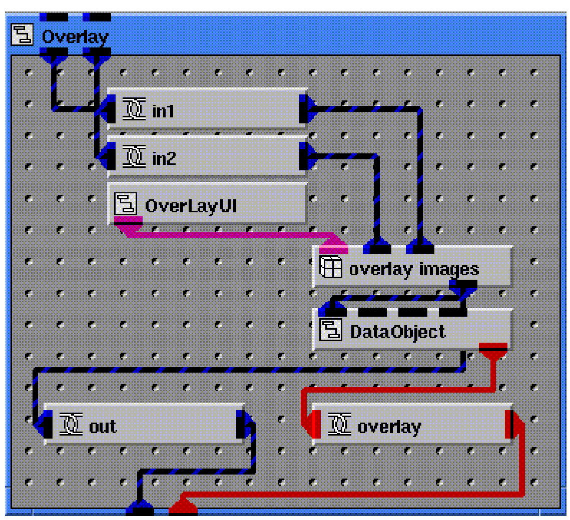
```
  compos_points[0]=0;
  compos_points[1]=0;
  compos_points[2]=compos_dims[0]-1;
  compos_points[3]=compos_dims[1]-1;
  ARRfree((char *)compos_points);

  // set single node_data
  compos.nnode_data=1;
  compos.node_data[0].id=669; // argb data
  compos.node_data[0].veclen=4;
 // need to use ret_typed_array_ptr as the values array
is instanced as a prim
  char *compos_node_data = (char *)com-
pos.node_data[0].val-
ues.ret_typed_array_ptr(OM_GET_ARRAY_WR, OM_TYPE_BYTE,
&compos_data_size);

  if(!compos_node_data){
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay:
can't access compos node_data\n");
    ARRfree((char *)image1_node_data);
    ARRfree((char *)image2_node_data);
    return 1;
  }

  /**********************/
  /* Function's Body    */
  /**********************/

  mask_arr = (int
*)mask.ret_array_ptr(OM_GET_ARRAY_RD,&mask_size);
  if (mask_arr)
  {
    int i,j;

    // overlay image2 on image1
    for(i=0; i<compos.nnodes; i++)
      for(j=0; j<4; j++)
compos_node_data[i*4+j] =
  (image2_node_data[i*4+j]==mask_arr[j]) ?
image1_node_data[i*4+j] : image2_node_data[i*4+j];

    // free all the arrays
    ARRfree((char *)mask_arr);
    ARRfree((char *)image1_dims);
    ARRfree((char *)image2_dims);
    ARRfree((char *)compos_dims);
    ARRfree((char *)image1_node_data);
    ARRfree((char *)image2_node_data);
    ARRfree((char *)compos_node_data);
```

```
  }
  else
    ERRverror("",ERR_NO_HEADER | ERR_INFO,"overlay:
can't access mask\n");

  // return 1 for success
  return(1);
}
```

The **OverlayUI** macro looks like:



and the final **Overlay** usr macro like:



The V code for the macros in **overlay_macs.v**:

```
flibrary OverlayMacs  {
  macro OverLayUI<NEx=88.,NEy=88.> {
      int int<NEportLevels={1,2},NEx=418.,NEy=352.>[4]
=> .mlink;
      UImod_panel Overlay<NEx=132.,NEy=66.> {
         height = 249;
         option {
            set = 1;
         };
      };
      UIslider a<NEx=154.,NEy=132.> {
         y = 0;
         parent => <-.Overlay;
         min = 0.;
         max = 255.;
         value = 0.;
         mode = "integer";
         increment = 1.;
      };
      UIslider r<NEx=297.,NEy=132.> {
         y = 60;
         parent => <-.Overlay;
         min = 0.;
         max = 255.;
         value<NEportLevels={2,2}> = 0.;
         mode = "integer";
         increment = 1.;
      };
      UIslider g<NEx=440.,NEy=132.> {
         y = 120;
         parent => <-.Overlay;
         min = 0.;
         max = 255.;
         value<NEportLevels={2,2}> = 0.;
         mode = "integer";
         increment = 1.;
      };
      UIslider b<NEx=583.,NEy=132.> {
         y = 180;
         parent => <-.Overlay;
         min = 0.;
         max = 255.;
         value<NEportLevels={2,2}> = 255.;
         mode = "integer";
         increment = 1.;
      };
```

```
      mlink mlink<NEportLevels=1,NEx=286.,NEy=253.> =>
{.a.value,.r.value,
         .g.value,.b.value};
   };

   macro Overlay {
      link in1<NEportLevels={2,1},NEx=88.,NEy=66.>;
      link in2<NEportLevels={2,1},NEx=88.,NEy=33.>;
      link out<NEportLevels={1,2},NEx=44.,NEy=275.> =>
.overlay_images.compos;
     link overlay<NEportLevels={1,2},NEx=231.,NEy=275.>
=> .DataObject.obj;
      OverLayUI OverLayUI<NEx=88.,NEy=88.>;
      IAC_PROJ.Overlay.OverlayMods.overlay_images
overlay_images<NEx=220.,NEy=110.> {
         image1 => <-.in2;
         image2 => <-.in1;
         mask => <-.OverLayUI.int;
      };
      GDM.DataObject DataObject<NEx=231.,NEy=220.> {
         in => <-.overlay_images.compos;
      };
   };

};
```