Assignment #2:
Scalar Field Visualization I: Color Mapping, Iso-contouring
Due September 24th, before midnight

Goals:
The first goal of this assignment is to get familiar with the PLY surface mesh representation and the corresponding data structure that we are going to use in a few coming assignments including this one. You will be provided a skeleton code to start with. You will learn how to modify the PLY file format to include scientific data defined on the mesh. The second goal is to practice the popular and basis visualization techniques for 2D/2.5D scalar fields including color coding and iso-contour extraction (we will leave the iso-surfacing for the second project). The third goal is to design the first visualization interface that allows the users to explore the data with a number of operators including the value range selection. Finally, you will be required to design a simple 1D mapping function for efficient color usage.

Tasks:

1. Color Mapping
You will be provided a number of models in PLY format with scalar or vector fields defined on them. For the data with vector fields, simply compute their magnitude as the scalar field for this assignment.

To load the data value, you need to modify the ply loader as follows:

In the “Skeleton.h” file, modify the “Vertex” class as follows:

```c++
class Vertex {
public:
    double x,y,z;
    float s;
    float vx, vy, vz;
    int index;

    ...
};
```

In the “Geometry.cpp” file, modify the “Vertex_io” structure as follows

```c++
typedef struct Vertex_io {
    float x,y,z;
    float s;
    float vx, vy, vz;
    void *other_props; /* other properties */
} Vertex_io;
```

Modify the vertex property list as follows

```c++
PlyProperty vert_props[] = { /* list of property information for a vertex */
```

1
Modify the vertex property set up list of the routine “Polyhedron::Polyhedron(FILE *file)” as follows

/* set up for getting vertex elements */
setup_property_ply (in_ply, &vert_props[0]);
setup_property_ply (in_ply, &vert_props[1]);
setup_property_ply (in_ply, &vert_props[2]);

setup_property_ply (in_ply, &vert_props[3]);
setup_property_ply (in_ply, &vert_props[4]);
setup_property_ply (in_ply, &vert_props[5]);
setup_property_ply (in_ply, &vert_props[6]);

... /* copy info from the "vert" structure */
 vlist[j] = new Vertex (vert.x, vert.y, vert.z);
vlist[j]->other_props = vert.other_props;
vlist[j]->s = vert.s;
vlist[j]->vx = vert.vx;
vlist[j]->vy = vert.vy;
vlist[j]->vz = vert.vz;

1.1 Successfully loading the PLY file with the data values (10 points)

1.2 Implement the “rainbow” and “blue-white-red” color schemes. Implement at least one other color scheme (excluding the discrete colors) introduced in the class. You need to map the scalar values to HSV color space as shown in the class. Use the radio buttons to allow the user to select from different color schemes. See GLUI document (page 25) and the skeleton code on how to do so. (20 points)

Note that to enable color shading under lighting condition, you need to add a line in the display_shape() routine before you render your colored surfaces.

....
Case 0:
    glEnable(GL_COLOR_MATERIAL);
....

1.3 The color mapping for some of the provided data may not be efficient. That is, most of the values for that data have similar range, and thus are colored similarly. The color plot for this data will show only a few distinguishable colors. Find out this data and try to design a color mapping functions to map the majority values to large range of the colors, i.e. showing more colors. Explain your color function and why do you think it will work and will not introduce mis-leading information. Experiment with the parameters of your color function and report your results. (20 points)
2. Iso-contouring on a 2D regular grid

You are going to run a temperature simulation of a plane that has 4 heat sources. The function you are going to use is

\[ t(x, y) = \sum_{i=0}^{3} A_i e^{-br_i^2} \]

where \( r_i^2 = (x - X_i)^2 + (y - Y_i)^2 \), \( b = 5.5 \) (you can also play with different value of \( b > 1 \)) and

<table>
<thead>
<tr>
<th>( i )</th>
<th>( X_i )</th>
<th>( Y_i )</th>
<th>( A_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>0.00</td>
<td>90.00</td>
</tr>
<tr>
<td>1</td>
<td>-1.00</td>
<td>-0.30</td>
<td>140.00</td>
</tr>
<tr>
<td>2</td>
<td>-0.10</td>
<td>1.00</td>
<td>110.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.40</td>
<td>170.00</td>
</tr>
</tbody>
</table>

You are given the temperature data from the simulation at each node in a 2D grid. The coordinate range of the data volume is: \(-1.0 \leq x, y \leq 1.0\). The number of node points you place along each dimension is at least 50 but should also be adjustable by the user.

```c
#define NX ??? // should be larger and equal 50
#define NY ???
```

For the temperature range of the data, use: \( 0.0 \leq t \leq 100.0 \). The temperatures defined by the equation actually can go higher than 100 degrees in places, but don’t worry about it. After computing \( t \), just clamp it to 100.0:

```c
const float TEMPMIN = { 0.f };  
const float TEMPMAX = { 100.f };  
...  
if( t > TEMPMAX )  
t = TEMPMAX;  
```

2.1 Set up your data structure to store the 2D grid and the temperature value at each grid point. NOTE that this data structure is different from what you have used for task 1. (10 points)

2.2 Visualize the data as point clouds, i.e. visualize them using

```c
glColor3f(r, g, b);  
glVertex3f(x, y, z);  
```

Apply the color coding schemes you have implemented in task 1 to these points. (20 points)

2.3 Use three GLUI range sliders to allow the user to cull the data by displaying a subset in X, Y, and Temperature. Use a fourth GLUI range slider to allow the user to control the display based on the
absolute gradient at each point. The gradient at each point is a 2-component vector: \( \left( \frac{dT}{dx}, \frac{dT}{dy} \right) \).

The absolute gradient is \( \sqrt{\left( \frac{dT}{dx} \right)^2 + \left( \frac{dT}{dy} \right)^2} \). This will show where the temperature is changing quickly and where it is changing slowly. (20 points)

The \( x \) gradient at a point is obtained by taking the difference from the point before to the point after. This is called a two-sided gradient computation:

\[
\text{Nodes}[i][j].dTdx = \frac{(\text{Nodes}[i+1][j].T - \text{Nodes}[i-1][j].T)}{(\text{Nodes}[i+1][j].x - \text{Nodes}[i-1][j].x)};
\]

The \( y \) gradient is similar. Be sure to take into account when \( i \) and \( j \) are either 0 or at their maximum value. This is when you would use a one-sided gradient computation.

2.4 Extract iso-contours corresponding to the user specified scalar values. (50 points)

For the contour part of the assignment, write a routine that will generate the correct contours for one single quadrilateral. If you use structures, you could pass in pointers to each point:

```c
void ProcessQuad( struct node *p0, struct node *p1, struct node *p2, struct node *p3 )
{
    ...
    ... p0->x ...
    ... p1->s ...
    ... p3->s ...
    ...
}
```

For easy management, you should consider to define a data structure to store contour segments.

Add two spinners (see page 29 of the GLUI library) to your interface. One is for the user to specify the scalar value that he/she wants to extract the contour. The other is to specify the total number, say “\( k \)”, of contours that will be computed. The scalar value for the \( i \)th iso-contour is then computed as

\[
s = \text{TEMPMIN} + i*(\text{TEMPMAX} - \text{TEMPMIN}) / (k-1);\]

3. Iso-contouring on triangular mesh (50 points)

Extend your iso-contour extraction algorithm for regular grid onto triangular mesh. How many valid cases within a triangle can you see? Try to handle them robustly.

Add a checkbox to the interface, saying “enable discrete colors”. By enabling that, the discrete color mapping is used based on the current color coding schemes. The level of discretization is determined by the number of contours that are extracted (i.e. “\( k \)” in task 2.4).
### Grades:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Total points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>

### Suggestions

Here are some suggestions that you may find helpful for this assignment.

The following code can be used to compute the scalar fields defined on the 3D grids. However, you can always develop your own code.

```c
struct sources
{
    double xc, yc, zc;       // heat source location for this assignment zc=0
    double a;                // temperature value of the source
} Sources[] =
{
    {  1.00f,  0.00f,  0.00f,  90.00f },
    { -1.00f, -0.30f,  0.00f, 140.00f },
    { -0.10f,  1.00f,  0.00f, 110.00f },
    {  0.00f,  0.40f,  0.00f, 170.00f },
};

// The following function is going to be used for the next assignment as well
double Temperature( double x, double y, double z )
{
    double t = 0.0;

    for( int i = 0; i < 4; i++ )
    {
        double dx = x - Sources[i].xc;
        double dy = y - Sources[i].yc;
        double dz = z - Sources[i].zc;
        double rsqd = dx*dx + dy*dy + dz*dz;
        t += Centers[i].a * exp( -5.*rsqd );
    }

    if( t > TEMPMAX )
       t = TEMPMAX;

    return t;
}
```

You can use the following data structure to store your 2D and 3D scalar fields defined at the regular grids.

```c
struct node
{
    float x, y, z;          // location
    float T;                // temperature
    float r, g, b;  // the assigned color
```
float rad;              // radius
float dTdx, dTdy, dTdz; // can store these if you want, or not
float grad;             // total gradient
}

Using the GLUI Range Sliders

We have added range sliders to the GLUI library. Here is how to use them.

For each variable, define a text-display format, a 2-element array to hold the low and high end of
the range, a pointer to the created slider, and a pointer to the created text-display:

// in the global variables:
#define TEMP 0
const float   TEMPMIN = {   0. };  
const float   TEMPMAX = { 100. };  
const char *  TEMPFORMAT = { "Temperature: %5.2f - %5.2f" }; 
float   TempLowHigh[2];           
GLUI_HSlider *  TempSlider;        
GLUI_StaticText * TempLabel;       
.
.
// in the function prototypes:
void Buttons( int );
void Sliders( int );
.
.
// in InitGlui( ):
char str[128];
.
TempSlider = Glui->add_slider( true, GLUI_HSLIDER_FLOAT, TempLowHigh, 
    TEMP, (GLUI_Update_CB) Sliders );
TempSlider->set_float_limits( TEMPMIN, TEMPMAX );
TempSlider->set_w( 200 );  // good slider width
sprintf( str, TEMPFORMAT, TempLowHigh[0], TempLowHigh[1] );
TempLabel = Glui->add_statictext( str );
.
.
The arguments to Glui->add_slider( ) are, in order:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range_slider</td>
<td>true means this is a 2-edged range slider</td>
</tr>
<tr>
<td>type</td>
<td>Use GLUI_HSLIDER_FLOAT</td>
</tr>
<tr>
<td>array</td>
<td>2-element float array to store values in</td>
</tr>
<tr>
<td>id</td>
<td>unique id to be passed into the callback routine (0, 1, 2, ...)</td>
</tr>
<tr>
<td>callback</td>
<td>callback routine to call when a slider is used</td>
</tr>
</tbody>
</table>
The arguments to `TempSlider->set_float_limits()` are the minimum and maximum values on that slider.

The argument to `TempSlider->set_w()` is the width, in pixels, of that slider in the GLUI window. 200 is a good number.

The argument to `Glui->add_statictext()` is the text string to display.

**The Button Callback Routine**

The buttons callback routine needs to be modified to re-do all the text strings if the Reset button is selected:

```c
void
Buttons( int id )
{
    char str[256];
    switch( id )
    {
        case RESET:
            Reset();
            sprintf( str, TEMPFORMAT, TempLowHigh[0], TempLowHigh[1] );
            TempLabel->set_text( str );
            ...
    }
}
```

**The Slider Callback Routine**

All range sliders can use the same callback routine:

```c
void
Sliders( int id )
{
    char str[32];
    switch( id )
    {
        case TEMP:
            sprintf( str, TEMPFORMAT, TempLowHigh[0], TempLowHigh[1] );
            TempLabel->set_text( str );
            break;
            ...
    }
    glutSetWindow( MainWindow );
    glutPostRedisplay( );
}
```

There are other color mapping you can find in the internet, here is one good site [http://dept.astro.lsa.umich.edu/~msshin/science/code/matplotlib_cm/](http://dept.astro.lsa.umich.edu/~msshin/science/code/matplotlib_cm/). Be creative and have fun!