Assignment #3:

2D Scalar Field Visualization (continued): Iso-contouring

Due September 24th, before midnight

Goals:

This is the second part of the project on 2D scalar field visualization.

The goal of this assignment is to practice the iso-contouring technique introduced in the class. You will also be asked to design a proper user interface using the GLUI library to fulfill to aid the data exploration. You will need to generate a 2D scalar field data for some parts of this assignment.

Tasks:

1. Prepare the data (10 points)

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 above simulation at each node in a 2D regular 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.

#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:

const float TEMPMIN = { 0.f };

1
const float TEMPMAX = { 100.f };  
...
if( t > TEMPMAX )
    t = TEMPMAX;

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 Assignment 2.

2. **Visualize the data using color plots (20 points)**

2.1 Visualize them using point cloud (10 points), using

```c
glBegin(GL_POINTS);
    for (...)  
        glVertex3f(x, y, z);
    ...  
    glEnd();
```

2.2 Visualize them with continue color plots (10 points), using

```c
glBegin(GL_QUADS);
    glVertex3f(x0, y0, z0);  
    glVertex3f(x1, y1, z1);  
    glVertex3f(x2, y2, z2);  
    glVertex3f(x3, y3, z3);
    glEnd();
```

You need to be careful about the ordering of the grid points in order to form a valid quad!

You should apply the color coding schemes you have implemented in Assignment 2 for this task.

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.

4. Extract iso-contours corresponding to the user specified scalar values. (60 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 ...
    ... 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 ith iso-contour is then computed as

$$s = \text{TEMPMIN} + i*(\text{TEMPMAX}- \text{TEMPMIN})/(k-1);$$
5. Iso-contouring on triangular mesh (40 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>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>40</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;
        t += dy * dy + dz * dz;
    }
    return t;
}
```
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.

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 do 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();

Be creative and have fun!