Assignment #4:

Scalar Field Visualization 3D: Cutting Plane, Wireframe Iso-surfacing

Due Sept. 27, before midnight

Goals:

With the results from your assignments #2 and #3, the first goal of this assignment is to get familiar with the slicing technique in 3D volumetric data visualization. You will need to design an interface to allow the volume slicing along the X, Y and Z axis directions, respectively. The second goal is to perform a simple iso-surfacing like visualization technique which is a natural extension of the 2D iso-contouring. All the visualization in this assignment will **re-use** the **coloring schemes** that you have implemented in Assignment #2, and **extend the interface** you developed in the past two assignments.

Tasks:

1. Data generation (10 points)

The function you are going to use is

\[ t(x, y, z) = \sum_{i=0}^{3} A_i e^{-5r_i^2} \]

where \( r_i^2 = (x - X_i)^2 + (y - Y_i)^2 + (z - Z_i)^2 \) and

<table>
<thead>
<tr>
<th>i</th>
<th>X_i</th>
<th>Y_i</th>
<th>Z_i</th>
<th>A_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>90.00</td>
</tr>
<tr>
<td>1</td>
<td>-1.00</td>
<td>0.30</td>
<td>0.00</td>
<td>120.00</td>
</tr>
<tr>
<td>2</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>120.00</td>
</tr>
<tr>
<td>3</td>
<td>0.00</td>
<td>0.40</td>
<td>1.00</td>
<td>170.00</td>
</tr>
</tbody>
</table>

You are given the temperature data from the simulation at each node in a 3D uniform grid. The coordinate range of the data volume is: \(-1.0 \leq x,y,z \leq 1.0\). The number of node points you place along each dimension is at least 50 but may also be adjustable by the user.

```cpp
const int NX = 50;  // Note: you can have an interface to change the data dimension
const int NY = 50;
const int NZ = 50;
```

For the temperature range of the data, use: \(0.0 \leq t \leq 100.0\). The temperatures defined by the equation actually go higher than 100 degrees in places, but don't worry about it. After computing \(t\), just clamp it to 100.
const float TEMPMIN = { 0.f }; const float TEMPMAX = { 100.f }; ... if( t > TEMPMAX )
    t = TEMPMAX;

Please refer to the later “Suggestions” for a suggested data structure to store the information on the individual nodes. You will need to create a 3D array to store this synthetic 3D temperature field.

node_grid_3d[NX][NY][NZ];

2. Volume slicing and exploration (50 points)

Use a few sliders to allow the user to cull the data by displaying a subset in X, Y, Z, and Temperature (i.e., showing slices). See the right figure for a demonstration. Note that you may see different interface styles if you the latest GLUI or AntTweakBar.

You will need to prepare three orthogonal cutting planes that are parallel to the XY, YZ, and XZ planes, respectively. Use three checkboxes to enable the user to toggle on or off certain cutting planes.

You need to compute the 2D scalar values on each plane and make a color plot for each plane using the color scales you have implemented in Assignment#2. The user should be able to move the respective cutting planes along the corresponding axes (see the right figures for an example using the GLUI interface)

Hint: You can organize the 3D grid points into the respective XY, YZ, and XZ planes so that the user can choose which XY, YZ and/or XZ planes to display.

Use a fifth slider to control the display based on the magnitude of the gradient at each point. The gradient at each point is a 3-component vector: \( \frac{dT}{dx}, \frac{dT}{dy}, \frac{dT}{dz} \). The magnitude of the gradient is computed as \[ \sqrt{\left(\frac{dT}{dx}\right)^2 + \left(\frac{dT}{dy}\right)^2 + \left(\frac{dT}{dz}\right)^2}. \] This will show where the scalar value is changing quickly and where it is changing slowly.

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 (or central difference) computation:

\[ \text{Nodes}[i][j][k].dTdx = \frac{\text{Nodes}[i+1][j][k].T - \text{Nodes}[i-1][j][k].T}{\text{Nodes}[i+1][j][k].x - \text{Nodes}[i-1][j][k].x}; \]
The y and z gradients are similarly computed. Be sure to take into account when i, j, or k are either 0 or at their maximum value (i.e., those boundary vertices). This is when you would use a one-sided gradient computation.

3. Wireframe Iso-surfacing (40 points)

Implement the wireframe iso-surfacing technique as introduced in the class (slides 5-6 in lecture 8). This should be done based on the iso-contours computed on those volume slices that are parallel to the XY, YZ, and XZ planes, respectively. Make sure you implement an efficient iso-contouring algorithm in Assignment #3.

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>50</td>
</tr>
<tr>
<td>3</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;       //
    double a;                // temperature value of the source
} Sources[] =
{
    {  1.00f,  0.00f,  0.00f,  90.00f },
    { -1.00f, -0.30f,  0.00f, 120.00f },
    {  0.00f,  1.00f,  0.00f, 120.00f },
    {  0.00f,  0.40f,  1.00f, 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;
```
```plaintext
double rsqd = dx*dx + dy*dy + dz*dz;
t += Sources[i].a * exp(-5.*rsqd);
}
if( t > TEMPMAX )
t = TEMPMAX;
return t;
}

In JS Version
var Sources = {
xc: [1.0, -1.0, 0.0, 0.0],
cy: [0.0, 0.3, 1.0, 0.4],
zc: [0.0, 0.0, 0.0, 1.0],
A: [90.0, 120.0, 120.0, 170.0]
};
/**
 * Generate the temperature data for assignment 4 based on the provided
 * equation
 * @param {*} x -1.0 <= x <= 1.0
 * @param {*} y -1.0 <= y <= 1.0
 * @param {*} z -1.0 <= z <= 1.0
 */
function Temperature(x, y, z) {
    var t = 0.0;
    for (var i = 0; i < 4; i++) {
        var dx = x - Sources.xc[i];
        var dy = y - Sources.yc[i];
        var dz = z - Sources.zc[i];
        var rsqd = dx*dx + dy*dy + dz*dz;
        t += Sources.A[i] * Math.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
};
```
**In JS Version**

You can create a Javascript class with a similar structure as follow

```javascript
class Node {
    constructor() {
        // location
        this.x = 0.; this.y = 0.; this.z = 0.;

        // temperature
        this.T = 0.;

        // the assigned color
        this.r = 0.; this.g = 0.; this.b = 0.;

        // radius
        this.rad = 0;

        // can store these if you want, or not
        this.dTdx = 0.; this.dTdy = 0.; this.dTdz = 0.

        // total gradient
        this.grad = 0;
    }
}

let myNode = new Node;
```

**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:

```javascript
// 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>Type</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>
</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:
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( );
}

In JS Version

We have added the slider library for the Javascript version

<!-- Bootstrap input slider-->
<script src="js/bootstrap-slider.min.js"></script>
<link rel="stylesheet" type="text/css" href="css/bootstrap-slider.min.css">

To create a slider, you first need to add an empty input in the html file

<
    <div class="col"><input id="x_slider" type="text" /></div>

then call .slider() on it:

/**
 * X slider
 */
```javascript
var xSlider = $('"#x_slider"').slider({ min: -1, max: 1, step: 0.02, value: [-1, 1], focus: true });
```

In the above example, we created a slider with values in the range [-1,1] and the step size is 0.02. You can find the full list of options to set for the slider at this [https://github.com/seiyria/bootstrap-slider](https://github.com/seiyria/bootstrap-slider)

Here is the example for slider callback routine:

```javascript
xSlider.on("change", function () {

    // Print out the current values
    console.log("Min: "+ xSlider.slider('getValue')[0] + " Max: " + xSlider.slider('getValue')[1]);

});
```