Assignment #6
2D Vector Field Visualization—Arrow Plot and LIC
Due Oct.15th before midnight

Goal:
In this assignment, you will be asked to implement two visualization techniques for 2D steady (time-independent) vector fields. The first technique is the direct visualization with colors and arrows, the second one is a texture-based method -- line integral convolution (LIC). You will be given a set of steady vector field data stored in PLY format for this assignment. All of them are planar data and defined in a planar region [0,1]x[0,1].

Task:

1. Direct method (20 points)

1.1 Load the data as described in assignment #2.
You will need to modify your file loader because the data now contains only vector components and does not contain scalar value. That means you have to remove variable “s” from

```c
PlyProperty vert_props[] = { /* list of property information for a vertex */
    {"x", Float32, Float32, offsetof(Vertex_io,x), 0, 0, 0, 0},
    {"y", Float32, Float32, offsetof(Vertex_io,y), 0, 0, 0, 0},
    {"z", Float32, Float32, offsetof(Vertex_io,z), 0, 0, 0, 0},
    /*!"s", Float32, Float32, offsetof(Vertex_io,s), 0, 0, 0, 0},
    {"vx", Float32, Float32, offsetof(Vertex_io,vx), 0, 0, 0, 0},
    {"vy", Float32, Float32, offsetof(Vertex_io,vy), 0, 0, 0, 0},
    {"vz", Float32, Float32, offsetof(Vertex_io,vz), 0, 0, 0, 0},
};
```

You should also modify the other part that is related to data loading (refer to the description of assignment #2).

1.2 Visualize the vector field magnitude, vector angle (0~2pi), x component, and y component, respectively using color plots. That is, totally 4 different color plots. You should have the corresponding interface to toggle on and off certain plots. (10 points)

1.3 Visualize the vector field using arrow plots (10 points)
To create an arrow plot, draw an arrow at each vertex of the mesh.
Use the following routine to draw an arrow head with the given direction if you do not have one. You should be able to draw the other part of the arrow that is just a line segment pointing from the vertex location (x, y) to the direction according to the vector value defined at it. Note that you need to scale the arrows uniformly through the whole field in order to get reasonable visualization.

```c
void draw_arrow_head(double head[2], float direct[2])
{
    glPushMatrix();
    glTranslatef(head[0], head[1], 0);
    glRotatef(atan2(direct[1], direct[0])*360/(2*M_PI), 0, 0, 1);
    glScalef(0.03, 0.03, 1);
    glBegin(GL_TRIANGLES);
    glVertex2f(0, 0);
    glEnd();
}
```
2. Line integral convolution (80 points)
Implement the LIC technique as described in the class. There are a number of different strategies to implement LIC. You are welcome to implement your own LIC algorithm. The following is my approach for your reference.

2.1 Set up three texture arrays
```c
const int IMG_RES = 512;
unsigned char noise_tex[IMG_RES][IMG_RES][3];
unsigned char vec_img[IMG_RES][IMG_RES][3];
unsigned char LIC_tex[IMG_RES][IMG_RES][3];
```
Here, IMG_RES is the resolution of the output image. For instance, if the output image is 512x512, then IMG_RES = 512;

2.2 Create white noise texture
There are a number of different ways to generate a noise texture. For instance, you can use a 2D perlin noise as your input texture. The following use the random number generator provided by C/C++ to compute a noise texture.
```c
void gen_noise_tex ()
{
    for (int x = 0; x < IMG_RES; x++)
        for (int y = 0; y < IMG_RES; y++)
            {
                noise_tex[x][y][0] = noise_tex[x][y][1] = noise_tex[x][y][2] = (unsigned char) 255*(rand() % 32768) / 32768.0;
            }
}
```

2.3 Encode the 2D vector field into an image
This is for the simplicity of the later streamline computation. We encode the 2D vector field into an image with the exactly same resolution of the output image. This image will be created once for each field that is loaded.

First, assume the vector value is in the form of (vx, vy) at each vertex. Then, search the maximum and minimum x and y components of the vector values through the entire field, i.e. max_vx, min_vx, max_vy, and min_vy.
Second, render the vector field into an image such that the image encodes both the x and y components. Specifically, for a vector value (vx, vy) at a vertex, we compute the color at each mesh vertex as follows
float rgb[3];
rgb[0] = (vx - min_vx) / (max_vx - min_vx); // red channel
rgb[1] = (vy - min_vy) / (maxVy - minVy); // green channel
rgb[2] = 0;

You can create a routine as follows and call it right after you load the data and set up the initialization. Remember the LIC image need to compute only once and will be saved as a texture for later rendering.

void render_vec_img( Polyhedron *this_poly)
{
    glViewport(0, 0, (GLsizei) 512, (GLsizei) 512);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glClearColor(GL_COLOR_BUFFER_BIT);

    glDrawBuffer(GL_BACK);
    int i, j;

    // first search the max_vx, min_vx, max_vy, min_vy through the entire field
    // render the mesh
    for (i=0; i<this_poly->ntris; i++) {
        Triangle *temp_t = this_poly->tlist[i];
        float rgb[3];
        rgb[2] = 0.5;
        glBegin(GL_TRIANGLES);
        for (j=0; j<3; j++)
        {
            Vertex *v = temp_t->verts[j];
            //determine the color for this vertex based on its vector value...
            glVertex2f (v->x, v->y);
        }
        glEnd();
    }

    // save the rendered image into the vec_img
    glDrawBuffer(GL_BACK);
    glReadPixels(0, 0, 512, 512, GL_RGB, GL_UNSIGNED_BYTE, vec_img);
}

2.4 Compute LIC image

The basic framework for computing a LIC image is as follows. For each pixel

//Compute a streamline using the vector field image in forward and backward direction (the streamline computation is terminated when the desired number of pixels is reached)
//Accumulate the color values from the pixels obtained in the previous step

As you have encoded the vector field into an image with the same resolution of the output in the previous step. The streamline tracing and the search of the pixels that it passes can be combined. Here is the basic idea of how these two can be combined.
Given a current pixel \((i, j)\) with \(i, j\) being its integer indexes corresponding to its row and column positions, respectively.

We start from the center of this pixel

```c
float y = i+.5;
float x = j+.5;
```

Let

```c
int next_i = i;
int next_j = j;
```

and extract the vector value from the `vec_img` as follows:

```c
vx = min_vx + (max_vx - min_vx) * vec_img[next_i][next_j][0]/255.0;
v y = min_vy + (max_vy - min_vy) * vec_img[next_i][next_j][1]/255.0;
```

This vector \((vx, vy)\) is normalized to be a unit vector so that you can move to one of its eight neighboring pixels.

The next position for the streamline computation can then be estimated using Euler integration.

```c
x = x + vx;
y = y + vy;
```

Then,

```c
next_i = int (x);
next_j = int (y);
```

is the next pixel the streamline will pass. Save it into an array.

We repeat the above process until

1) we hit the boundary of the domain, i.e. \(next_i<0\) or \(next_j<0\) or \(next_i>=IMG_RES\) or \(next_j>=IMG_RES\),

or 2) the current vector value is smaller than some threshold, i.e. reaching a fixed point,

or 3) the total number of pixels that we have visited is \(L/2\) (Assume \(L\) is the kernel size).

The backward tracing can be similarly implemented.

NOTE that this is a really coarse estimation of the streamline and will introduce large error. You can consider to resort to high-ordered integrators to improve the accuracy.

Then the color of the current pixel can be computed as the weighted sum of the color values of those pixels extracted in the streamline tracing. In this assignment, a simple average of those colors is sufficient.

### 2.5 Render the LIC image through texture mapping

Replace your `Display()` function with the following one since we are working on 2D vector field now.

```c
void Display()
{
    glViewport(0, 0, (GLsizei) IMG_RES, (GLsizei) IMG_RES);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);

    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_TEXTURE_2D);
glShadeModel(GL_FLAT);

/// Test noise texture (for debugging purpose)
//glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, IMG_RES, IMG_RES, 0,
//    GL_RGB, GL_UNSIGNED_BYTE, noise_tex);

/// Test vector field image (for debugging purpose)
//glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, IMG_RES, IMG_RES, 0,
//    GL_RGB, GL_UNSIGNED_BYTE, vec_img);

// Display LIC image using texture mapping
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, IMG_RES, IMG_RES, 0,
    GL_RGB, GL_UNSIGNED_BYTE, LIC_tex);

glBegin(GL_QUAD_STRIP);
glTexCoord2f(0.0, 0.0); glVertex2f(0.0, 0.0);
glTexCoord2f(0.0, 1.0); glVertex2f(0.0, 1.0);
glTexCoord2f(1.0, 0.0); glVertex2f(1.0, 0.0);
glTexCoord2f(1.0, 1.0); glVertex2f(1.0, 1.0);
glEnd();

// Add you arrow plot here and use a checkbox to enable its visualization

// IF you are using AntTweakBar, please call the following to draw it
TwDraw();

glutSwapBuffers();
glFlush();
}

You can also add a checkbox to your interface to toggle on and off this rendering mode.

If you use AntTweakBar, please modify your Reshape() function as follows

void Reshape(int width, int height)
{
    // Set OpenGL viewport and camera
    glViewport(0, 0, (GLsizei)width, (GLsizei)height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 1, 0, 1);
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Send the new window size to AntTweakBar
    TwWindowSize(width, height);
}

3. Enhance the original LIC (extra)
You can enhance the obtained LIC result from the previous task by doing the following.

3.1 Visualize vector magnitude with the LIC image (5 points)
You can use OpenGL blending function to blend the color plot showing the vector magnitude with the LIC image. Another option is to vary the kernel length (e.g. L) for each pixel according to the magnitude of the vector valued defined at the current pixel. Try to explore that by yourself.

3.2 Enhanced LIC (5 points)
Implement a simplified version of Enhanced LIC. After getting the initial LIC in Task 2, use it as the input (i.e. replace the original white noise) to perform one more LIC process.

Grades:

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Total Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>10 (extra)</td>
</tr>
</tbody>
</table>

Here are the results for the five data sets you should expect to see.