Project 3 - 3D Transformation

For this project, you will learn how to load a 3D data file, present a 3D model, and rotate and shear it freely from scratch. The goal is to give you a first-hand experiences of triangle/polygon based 3D modeling, 3D transformation, as well as mouse/keyboard interaction with 3D objects. You are suggested to start with the example provided files "OpenGL_basics", which gives you the barebones of OpenGL.

Your Tasks:

(1)Load 3D data from a ".obj" file.

(2)Display the 3D object in an OpenGL window by rendering all the triangles that construct the 3D object shape.

(2)Rotate the 3D object by mouse drag (Left button drag).

(3)Shear the 3D object by mouse drag ("Shift" + Left button drag).

For all the above tasks, you can only use the basic OpenGL commands (as the example provided on Isidore "OpenGL_basics.cpp"). For example, to render a triangle, you can call "glBegin(GL_TRIANGLES)", which is a basic drawing functions in OpenGL. However, for those more advanced functions, such as "glRotate()" or "glShear()", they are not allowed to use. You should implement these functions yourself. The process can enhance your understanding on the 3D fundamentals. Below are the details for each task:

(1) Load a 3D data from a ".obj" file:

A ".obj" files is provided: "teddy.obj", which consists of many triangles and the corresponding vertices. Your program should be able to load the "teddy.obj" file and extract the triangles automatically. The basic format of the obj file is as follows:

….

v -0.0741957 0.255051 0.00482052

v -0.128779 0.302833 0.0741941

v 0.0209863 0.262466 0.0322468

……

f 353 223 187

f 73 237 223

f 234 187 317

There are two types of lines in the ".obj" file. Starting with the letter "v", it means this line represents a 3D coordinate of one vertex. Starting with letter "f", it means this line represents a triangle face of a 3D object. The rest of this line has the corresponding vertex index for each corner of the triangle. For example, "f 234 187 317" means the triangle has the three vertices with the indices 234, 187 and 317, which are stored in the vertex list. So to make your code more efficient, it is recommended you can create two structures: Vertex and Triangle. This can make your code cleaner in storing the data and make your later computation easier.

e.g.

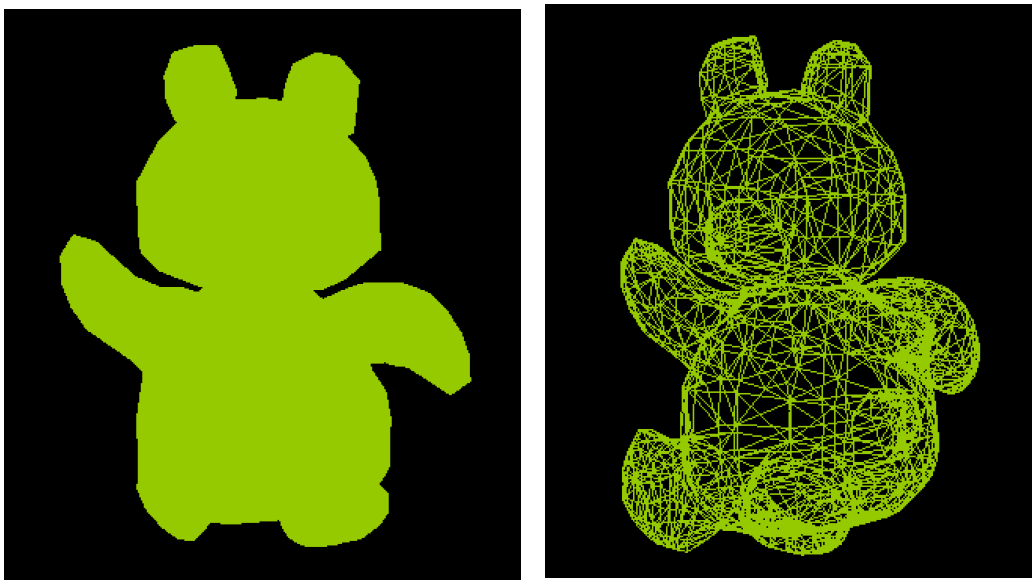
```
struct Vertex{
        float x,
        float y,
        float z};


struct Triangle{
        int v1,
        int v2,
        int v3
        };
```

So to store the data, you can create: vector<vertex> vt_list; vector<Triangle> tg_list; For each Triangle, the index v1, v2, and v3 actually refer to the positions of the corresponding vertices in vt_list. (Attention: you should use vt_list[v1-1] instead of vt_list[v1], as for a vector, it starts counting from 0).

II. Display the 3D object

After loading the 3D data, you can display it in the window by calling OpenGL basic command. To display the 3D data with a single color, you can use glColor3f() to set a fixed color for the triangle followed by calling glBegin() and glVertex() to render each vertex for a triangle. When you first time render this 3D object, you may feel it is not like a 3D since you have not done any illumination or shading yet. All the vertices have the same color. To have a better 3D view, you can use "glBegin(GL_LINE_LOOP)" instead of "glBegin(GL_TRIANGLES)" to generate a mesh version of the 3D object. You are required to implement a keyboard callback function that allows the user to hit "m" or "M" key to switch between the triangle surface view and the mesh view, as the figure shows below:



III Rotation and Shearing:

For this two functions you cannot use the OpenGL provided commands. Instead, you should manually create a 4x4 matrix for the rotation or shearing operation. Then apply the implemented matrix to times each vertex to get a new position for the vertex. You are recommended to create a matrix structure and reload the multiplication operator, which can simplify your future computation. The basic structure can be as follows:

```
struct Mat{

  float elem[4][4];



  /*Mat times another Mat function*/

  const Mat operator *(const Mat& right) const
```

```
  {
      Mat result;
      …
      return result;
  }

 /*Mat times a vertex and return another vertex*/
 const Vertex operator *(const Vertex& v) const
 {
      Vertex result;
      …
      return result;
 }
};

/*After you have implemented this matrix and the operators, you can use it to transform a 3D point easily, below is a simple example*/

Mat R_x, R_y, R_z; //rotation matrices

Mat Shear; //shearing matrix

… //Assign values to R and T

Mat P = T * R_x * R_y;  //for example: P

Vertex v_new = P * v; //rotate the vertex v about y axis, then rotate it about x axis, and finally shear it to a new shape.
```

Rubrics:

1 Correctly load the .obj file (20%)

Your program should be able to load the 3D data and store them into your designed data structure, such as vertices and triangles.

2 Display the 3D object in an OpenGL window (20%)

Display the triangle view and mesh view for the Teddy bear object. For both views, you can assign a fixed color value for all the triangles or lines. Users can hit the "M" or "m" key to switch between the two views.

3 Rotate the 3D object by mouse dragging (40%)

You need to implement the rotations about x and y axis. Here are the rules:

(a) drag the mouse horizontally - rotate the object by y axis.

(b) drag the mouse vertically - rotate the object by x axis.

(c) drag the mouse horizontally & press the "z" key – rotate the object by z axis

(Hint: for all the above cases, we use left button down for the mouse drag. The way to map the mouse drag to the rotation is actually mapping the mouse movement in pixels to the angle change in degrees. So what is the relationship? From my experiment, you can use:

mouse_position_change / 100 = angle_degree_change;

This can achieve an appropriate rotation speed. Of course, you can adjust it to fit your preferred speed.

4 Shear the 3D object by mouse dragging + SHIFT (20%)

Similar to the rotation drag, you are required to implement the shearing along the x and y axis by mouse dragging together with pressing "SHIFT" key. You can also use the relationship "mouse_position_change / 100 = coordinate_change", which works pretty well on my computer.

(a) drag the mouse horizontally & press the "Shift" key - shear the object along y axis.

(b) drag the mouse vertically & press the "Shift" key" - shear the object along x axis.

Submission:

1. As previous projects, only source files are needed to be submitted via Isidore. You can create as many files as needed.
2. Please provide a .docx or .pdf file showing several screen captured images of your running results: the Teddy bear can be sheared or rotated to different directions.