Project 2

1 Goal

This project is meant to familiarize you with OpenGL, and see the rendering pipeline concepts and algorithms 'in action'.

First, you will need to read in a 3D triangle model. Its format is going to be the following. The first two entries will be integers, the triangle count and the vertex count. Then, there will follow a triangle table: a number of lines, each listing three integers (ID's ov vertices bounding a triangle). Finally, there will be the vertex table: coordinates of all the vertices. For example, here is an input file that represents a tetrahedron:

```
4 4
0 1 2
2 1 3
2 3 0
0 3 1
0.000000 0.000000 1.000000
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 0.000000
```

The first two numbers say that you have 4 vertices and 4 triangles (mind the order!), the coordinates of the vertices are listed in the final 4 lines and the other lines tell you which triples of vertices bound a triangle.

You will need to draw the model you have read in. We are not setting very strict requirements as to how to set the lighting conditions or viewing parameters (see below though); do whatever pleases your eyes, just make sure the image looks clean and you don't do anything crazy like putting the light source inside the model or such. Also, make sure that in the initial image you draw the model shows up and it roughly fits inside the window. The center of the window should roughly overlap with the center of the object. One way to ensure this is to apply a transformation (translate and scale) to the model so that the center of the *bounding box* of the model projects to the center of the screen and the entire bounding box is visible. The bounding box is the smallest axis-oriented paralleliped enclosing the model. To compute the coordinates of its center you can, for example, calculate the maximum and minimum x-, y- and z-coordinates of a vertex and then compute the average of the minimum and maximum x, minimum and maximum y and minimum and maximum z. After the program is started, a still image should be displayed.

Here are the the first two specific things you have to do (see the Interface section for more) to get full credit:

1. Turn **on** back-face culling

2. Place the light somewhere a little bit above the viewpoint (typically, the upper part of the visible part of the model should appear brightest).

You don't need to build triangle strips or fans, just draw the triangles as 'soup', independently (`GL_TRIANGLES` mode).

2 Interface

2.1 Rotating the model: Virtual trackball

The main part of the user interface will be a virtual trackball. Basically, the idea is to give the user the ability to grab (by pressing the left mouse button) an imaginary (invisible) ball centered at the middle of the displayed model and let her rotate it by moving the mouse with the left button down. Of course, the displayed model should rotate together with the ball. This requires proper handling of mouse events (see `glutMouseFunc`, `glutMotionFunc`).

2.2 Menu

You should be able to select the following items from a menu (just as in the enclosed code, the menu should be attached to the right mouse button):

lines Line mode: draw only the edges of the triangles

points Point mode: draw only the vertices

triangles Triangle mode: draw 'filled' triangles

toggle Gouraud/flat Toggle between Gouraud shading (smooth shading in OpenGL nomenclature) and Flat shading [see 'normals' section]

zoom in Zoom in a little (we want to be able to iterate this until we see just this little triangle in the middle!)

zoom out Zoom out a little (also here, we need to be able to iterate this)

W/M-fixed light - Toggle between light having a fixed location in the world and having a fixed location relative to the model

toggle orientation Switch from back-face culling to front-face culling and back; note that this is equivalent to reorienting all triangles [that's meant to show the effect of wrong oriantation of triangles on the back face culling algorithm]

For zooming in and out, I suggest changing the field of view in `gluPerspective` call. Although scaling the model may also be an option, it may be less convenient: you need to be careful about the model growing so big that it contains the origin or the front of it being clipped away by the front clipping plane.

The current modelview matrix is applied to the light source location at the moment you specify it. This is the key to implementing the light behavior correctly.

3 Normals

You will have to define a normal vector for each vertices you specify with `glVertex`.

In the flat shading mode, use the true normals of the triangles.

In the smooth (Gouraud) shading mode, compute the vertex normals by averaging the normals of neighboring faces. More precisely, first compute the triangle's normal as the cross product of vectors running along the edges. Then, add all of the normals of incident faces to get the normal at a vertex. This will yield the area-weighted average normal.

4 OpenGL

There are not too many commands you will need except for the ones already used in the skeleton code. Implementing rotation requires understanding of `glMulMatrix`, `glMatrixMode`, `glLoadIdentity` perhaps `glRotate`. Remember that the current modelview matrix is applied to the light source location at the moment you specify the light.

For line, point and triangle mode, take a look at `glPolygonMode`.

For zooming in/out, manipulate the field of view in `glPerspective`.

For switching between shading models, see `glShadeModel`.

Commands important for specifying vertices: `glBegin`, `glEnd`, `glVertex`, `glNormal`. To specify materials and light, you can use (or modify) the routines I provided.

Commands you may need to deal with display lists: `glNewList`, `glGenLists`, `glEndList`, `glDeleteLists`.

For turning on options like back-face culling, automatic normal vector normalization, depth test and an other settings: `glDisable`, `glEnable`.

Setting which faces are culled: `glCullFace`.

You can use glGet to grab the current transformation matrix or read other components of the OpenGL state. Like the modelview matrix – you might want to choose this to help you with the trackball (note that this might be sufficient in this 'toy' project, but if you were to sell your code to someone... I would advise doing this in software – see the writeup on trackball).

5 Due date and Grading

The projects are due on Monday, February 20 midnight. Late penalty: as in the first project (3^{n-1}, where n is the number of days late). Send your code (tar file containing the makefile and all source files) to `cs4451@cc.gatech.edu`. All files should extract to the current directory and compile by simply typing 'make'. The name of the executable should be 'proj2', and should read the input file from the standard input and provide a way of specifying the window size exactly like the provided piece of code. We'll be running your programs like this:

```
% proj2 < input_file_name
```

We'll use linux. Make sure your code compiles and runs on the lab machines. Programs that do not won't get credit.

6 Grading

We'll run your code for the posted examples.

1. line/triangle/point modes: 5

2. gouraud/flat modes: 25 (includes normals; 15 point penalty for a superlinear implementaton of normal vector calculation)

3. back face culling and toggle orientation: 10

4. zooming in/out: 10

5. Trackball: 50

Most of the grading will be done by just running your code and trying things out.

Make sure that the time between the code is started and the model shows up in your window is only about a few seconds and there are no delays when switching between modes etc. As always, we need to be able to *see* that something works to give credit for it. By 'works' I mean 'works in general', i.e. for all input models rather than just one or two.

When grading the trackball, we'll check if it gives reasonable immediate feedback (meaning reasonable refreshing of the view in response to the mouse-moving-with-button-down events) and if it is reasonably intuitive to use. Most importantly, the rotation direction/axis/angle should be naturally linked to the mouse cursor movement.

You are welcome to discuss high-level implementation details or compare the output/interface of your code. However, you have to write the code yourself.