

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 of 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 triangles, 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.

Then, 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); 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 the window. The center of the window should roughly overlap with the center of the object. One way to ensure this is to set up the view and apply a transformation to the model so that the center of the *bounding box* of the model projects to the center of the screen. The bounding box is the smallest parallelepiped 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 more specific things you have to do (but not all of them - see the Interface section) to get full credit:

1. Turn **on** back-face culling
2. Use *display lists*

3. 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 independently.

2 Interface

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 left 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

zoom out Zoom out a little

animate/freeze model Toggle between animated (e.g. rotated like the cube in the sample code) and still image of the model.

slower (model) - animate model slower

faster (model) - animate model faster

animate/freeze light - Toggle between light rotating around the model and light not changing its location

slower (light) - animate light slower

faster (light) - animate light faster

toggle orientation Reverse the orientations of the triangles this should cause the front-facing triangles not to be drawn - basically, you will only see the 'back' of the model; may look and behave somewhat weirdly. Choosing this menu item again should restore the original orientation [that's meant to show the effect of wrong orientation of triangles on the back face culling algorithm]

Important points:

1. The light animation should be independent of the model animation. If you select to freeze or animate the model, this should not in any way affect the trajectory of the light source. And the other way around, if you choose to freeze/animate the light, the model should just continue to move or stay still undisturbed.

2. You should not have any discontinuities in your animation
3. Use the normals as described in the 'Normals' section below

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 normal of each of the triangles (i.e. 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, use either `glScale` or 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`.

5 Due date and Grading

The projects are due on Friday, February 28 on 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. The name of the executable should be 'project2', 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:

```
% project2 -d window_size < input_file_name
```

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

Grading:

1. line/triangle/point modes: 10
2. gouraud/flat modes: 10
3. animation: 30
4. correct normals: 20
5. display lists used correctly: 10
6. back face culling and toggle orientation: 10
7. clarity of the code: 10