CS 354
Project 4 - object viewer
Due Wednesday, November 7, 11:59 pm

Overview

For this assignment you will implement a viewer for mesh files in Wavefront obj format. The Wavefront obj file format is described here: `http://en.wikipedia.org/wiki/Wavefront_.obj_file`. It contains meshes along with material properties. Note: most, but not all features of obj files will be implemented.

The purposes of this assignment are:

1. Deepen understanding of transformations
2. Understand lighting and texturing

There are four major components to this assignment:

1. Implement arc ball-style mouse rotation and also zoom
2. Render obj files with proper lighting and material properties, including texturing
3. Create a custom obj file
4. Create a scene using multiple obj files

Specifications

1. The initial view shows the entire mesh.
2. Material properties are determined by the obj file.
3. At least one light source is in a reasonable position.
4. Dragging the left mouse button rotates using an arc ball.
5. Dragging the right mouse button up and down zooms in and out.
6. A custom obj file called `original.obj/mtl`. You can use existing jpg files for your textures, but the obj and material file must be original.
7. A screen shot of a custom scene called `original.png`. Do NOT submit additional obj files used to create this scene.
8. Creativity points are based on the custom obj file and on the custom scene.
9. If you added special features to your viewer, list them in README.txt. Also, list all obj files used in your custom scene and where you got them. Also describe how you created your custom obj file.
10. Submit your work by typing `turnin --submit edwardsj project4 *` at the command-line. Do NOT submit any of the sample data. The only data you should submit is the obj/mtl/jpg files you create. If you include any of the sample data you will lose points. You are responsible to ensure that what is submitted is correct. A helpful tool is `turnin --verify`. Type `man turnin` at the command-line for details.

Suggested approach

1. Implement rotation. See arc ball instructions.
2. Implement zoom.
3. Get familiar with code presented in class on lighting and textures.

4. Add data structures to store vertices and polygons as they are read in from the obj file. This will be done by adding code to mesh.cpp/h. Look for TODO comments.
5. Render the obj polygons without lighting.
6. Compute the vertex normals. Polygon normals are computed using the cross product of two vectors in the plane. Vertex normals are computed using the sum of normals of each incident polygon.
7. Render the normals to make sure you have them right. This can be done using `GL_LINES`.
8. Enable default lighting and make sure polygons render correctly.
9. Add material properties to your meshes.
10. If the variable `scene_lighting` is true, then any light sources should remain fixed with respect to the world frame. If it is false, then keep the light source(s) fixed with respect to the camera.
11. Create custom obj and mtl files. Call them original.obj/mtl. You can use any jpg texture you wish. Vertices in the obj file can be entered by hand or you can write a program to do something more sophisticated.
12. Create a custom scene. When viewer is run with the -s option you should render your scene.
13. Use turnin to submit your work. DO NOT submit any data files other than original.obj with any associated mtl and jpg files. We may request data files you use for your scene, but do not submit them.

Arc ball

To implement rotation with the mouse we'll take the approach of an arc ball. Imagine there's a ball in the world centered at the origin.

1. When the user clicks on the screen, compute the location of the click normalized to a square of length 2 centered at the origin. For example, $x_n = 2 \cdot x/\text{window\_width} - 1$. Remember to account for the upside-down coordinate system in y.
2. Now intersect the ray parallel to the z axis that passes through the point with the unit sphere centered at the origin. Call the intersection point p'. p' will have a positive z value. If the clicked point lies outside of the unit sphere just move it to the closest point.
3. As the user drags the mouse, use the same approach to compute a point q'. The axis of rotation for your world will be a vector normal to the plane defined by the points O, p' and q', where O is the origin. The angle of rotation will be the angle between vectors $p' - O$ and $q' - O$.
4. To account for multiple rotation mouse gestures, you'll need to maintain all previous rotations in a matrix. The OpenGL functions `glLoadMatrixf` and `glGetFloatv` will be useful in doing this. You can either write your own matrix multiply or piggyback on the implicit matrix multiply when you call glRotatef. It is best if the code to update this rotation matrix is not in your display function.

Scoring

1. 25% - Arc ball rotation
2. 10% - Zoom
3. 25% - Mesh renders with correct normals and lighting; `-l` option works as specified
4. 20% - Mesh renders with correct material properties
5. 5% - Custom obj file
6. 5% - Custom scene
7. 10% - Coding quality and style given by report from check-code and visual inspection.

Notes

1. You may use the standard C++ library (e.g. vector). You *may not* use any other third-party libraries.
2. `./check-code` is run automatically when you build.
3. You can submit your work as frequently as you like – only the most recent submission will be retained. Suggestion: submit first thing to get familiar with how it works and submit occasionally during development. This way there won't be any surprises when you're up against the deadline.