Spring ’11 CIS 441/541 Assignment 2 – 120/100 points possible – Due Tuesday, 5-3, 11:59 PM

The goal of this assignment is to provide experience creating simple 3D objects in OpenGL and shading those objects using fixed-function lights and materials. The exercises from the book are intended to allow practice in working through vector-math problems and to provide an understanding of the math underlying the graphics pipeline.


2. [10] Exercise 5.21, page 287. Write your answer in the form of two gluLookAt() calls, assuming that the monoscopic eye position is \((x, y, z)\), the center of interest is \((0, 0, 0)\), and the up vector is \((0, 1, 0)\).


5. [10] Create assignment2.c with an int main() function which creates a GLUT window with title “Assignment 2” and dimensions 800x600. The window should use double buffering and enable GL_DEPTH_TEST and GL_SMOOTH shading. Also turn on backside-face culling using glEnable(GL_CULL_FACE) and glCullFace(GL_BACK), which will avoid rendering the back faces of all polygons for improved performance and easier debugging of lighting (we’ll discuss in class).

6. [10] Write a function which draws a unit cube (i.e., side length 1.0) centered at the origin.

7. [10] Write a function which draws a unit tetrahedron (i.e., side length 1.0) centered at the origin.

8. [10] Set up a perspective projection to view your scene with a 50-degree field of view and near/far clip planes appropriate for your scene. Add a reshape callback so that this projection is maintained when the window is resized. 410: You may create the perspective projection using gluPerspective(). 510: Create and load your own perspective-projection matrix. You should now be able to view your cube and tetrahedron.

9. [10] Add mouse handling to your application such that horizontal mouse movements orbit the camera about the Y axis at the origin and vertical mouse movements orbit the camera about the X axis at the origin when the left mouse button is down. Use glVertex() and glRotatef() but build the modelview matrix, but do not use gluLookAt(). You should now be able to rotate around your cube and tetrahedron.

10. [10] Write a function which will generate a unit (i.e., length of 1.0) normal vector for a given polygon using a cross-product operation. Use this function to create normals for your cube and...
tetrahedron drawing functions. Also enable GL_LIGHTING and GL_LIGHT0. You should now be able to view lit versions of your cube and tetrahedron. You may use the default parameters for the light or specify your own light parameters. You may use additional lights if you’d like, but I suggest getting a single light working before adding additional lights as it’s easy to oversaturate a scene.

11. [10] Add an idle callback to your application and animate a scene consisting of two cubes orbiting about the Y axis at the origin and two tetrahedrons orbiting about the Y axis of each cube. Use glPushMatrix() and glPopMatrix() to store/restore intermediate versions of the modelview matrix. Assign materials consisting of ambient, diffuse, and specular components to these objects to give them some distinction. Your final result should look something like:

![Assigned 2](image)

12. [+20] Write a function which draws an approximation of a sphere. You may approximate the sphere either by recursively subdividing a tetrahedron or using bands consisting of triangle or quad strips and caps consisting of triangle fans. Do not use gluSphere(). Replace the cubes in your scene with spheres created using your sphere-approximation function.
Zip your source file(s) and problem-set solution document and upload the .zip file to Blackboard.