

**Fall 2009**  
**22C:151 Introduction to Computer Graphics**  
**Assignment 5**

**Due: Wednesday, October 7th at 11:59 pm**

**Goal:** Setup a 3D OpenGL program with perspective and simple geometry, and utilize matrices to manipulate objects.

**Problem 1 (20 points):** Write an OpenGL/GLUT program that renders 3D geometry into a  $640 \times 480$  window.

- **Part A (3 points):** Set the projection matrix for 3D instead of 2D. Set your view frustum using `gluPerspective()` with a near plane at a distance of 1, a far plane at distance 100, and a field of view of  $90^\circ$ . Set the aspect ratio correctly. *Hint: If you are building on existing code, this should replace `gluOrtho2D()`. Also note the aspect ratio should not be 1 – make sure you are not using integer division!*
- **Part B (1 point):** Set your eye point (using `gluLookAt()`) to have the eye at (0,5,5), looking at (0,0,0) with an up vector of (0,1,0).
- **Part C (2 points):** Remember that in OpenGL you can equivalently think of “moving the eye” or “moving the scene.” A single translation and a single rotation can give you a matrix equivalent to that generated by `gluLookAt()`. What translation and what rotation are required? Write the answer in your README.

*Hint: The answer may not be your first thought. You can check your answer by comparing your answer with the matrix `gluLookAt()` creates. You can get this matrix by reading back the modelview matrix using the following code:*

```
GLfloat gluLookAtMatrix[16];
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt( <...fill in correct values...> );
glGetFloatv(GL_MODELVIEW_MATRIX, gluLookAtMatrix );
printf("%f %f %f %f\n", gluLookAtMatrix[0], gluLookAtMatrix[4], <...etc...>);
printf("%f %f %f %f\n", gluLookAtMatrix[1], gluLookAtMatrix[5], <...etc...>);
<...etc...>
```

- **Part D (3 points):** Write a function (with no parameters) that draws a cone of height 2 with a radius of 1 at the base.
  - The tip of the cone should be at (0,0,2) and the base should be on the  $z = 0$  plane (so (1,0,0), (-1,0,0), (0,1,0), and (0,-1,0) are all on the cone’s base).
  - The tip of the cone should be **green** and the base should be **red** at (-1,0,0) and **blue** at (1,0,0). Along the base of the cone, you should smoothly interpolate between red and blue (e.g., the color at points (0,1,0) and (0,-1,0) should be a dark purple color (0.5,0,0.5)).
  - This function will be called from inside your display function, so it should not call `glClear()`, `glFlush()`, or `glutSwapBuffers()`.
  - *Suggestion: Use two triangle fans. One for the triangles on the base of the cone (in the  $z = 0$  plane) and one for the conical part. Since your cone will be made up of triangles, use at least 40 triangles to make it appear smooth.*
- **Part E (2 points):** Draw your cone. You should be able to call your cone function from Part D in your `display()` function. Rotate it so that the tip of the cone is at (0,2,0) and the base lies on the  $y = 0$  plane. **Do not change your cone function from Part D to do this! You must have OpenGL apply a rotation matrix.**

- **Part F (3 points):** Setup an idle callback that causes the cone in Part E to rotate around the  $y$ -axis by a small amount each frame. Initially, the colors on this cone will exhibit a strange behavior during rotation. In order to eliminate this problem, you must enable `GL_DEPTH_TEST` so OpenGL tracks which side of the cone is on top. Add a keyboard callback (with the 'd' key) that toggles back and forth between enabling and disabling `GL_DEPTH_TEST`.

– *Note: In order for depth testing to work correctly, you must add `GLUT_DEPTH` to the list of modes passed to `glutInitDisplayMode()`!*

- **Part G (2 points):** Draw another cone with base centered at (3,1,0) with tip at (3,1,2). Reuse your cone function from Part E. You should have a single cone function that you call twice, once for Part F, once for Part H. This cone should not rotate!
- **Part H (4 points):** Draw a **yellow** teapot and place it somewhere (other than the origin) so that it's visible in the scene. Make sure it does not overlap with the cones. (*Hint: Use `glutSolidTeapot()`, which you can read about in the *OpenGL Programming Guide*.*)

#### NOTES:

- A "README" file is required in order to get full credit. It must include compiling instructions.
- Posting images of your assignment on your webpage is required in order to get any credit.
- You may add additional geometry to your scene, as long as it doesn't overlap with the cones or teapot.
- A sample executable is available on the web page for you to see my expectations.