

Fall 2005
22C:151 Introduction to Computer Graphics
Assignment 5

Due: Wednesday September 28th at 11:59pm

Goal: Setup a 3D OpenGL program with a perspective viewing window, and some simple geometry. Get a feel for how OpenGL manipulates objects in 3D.

Problem 1 (20 points): Write a OpenGL/GLUT program which renders 3D geometry into a 640×480 window. Use the whole window as the viewport (i.e., a 640×480 viewing area).

- **Part A (3 points):** Set the projection matrix for 3D instead of 2D. Setup a viewing frustum with a near plane at a distance of 1, a far plane with a distance of 100, and a field-of-view of 90° . Set the aspect ratio correctly.
 - If you are building off your previous code, this should replace your `gluOrtho2D()` call.
 - Hint: the aspect ratio should not be 1.
- **Part B (1 point):** You can use either `gluPerspective()` and `glFrustum()` to do Part A. In your README file, write the two calls (to `glFrustum()` and `gluPerspective()`) that give the projection matrix required for Part A.
- **Part C (1 point):** Set your eyepoint using `gluLookAt()` to have the eye at (0,5,5) looking at (0,0,0) with an up vector of (0,1,0).
- **Part D (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. This is a bit tricky. 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 E (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 color of 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 cone’s base, you should smoothly interpolate between red and blue (i.e., the color at both points (0,1,0) and (0,-1,0) should be a dark purple of value (0.5,0,0.5))
 - This function will be called from inside your display function, so it **should not** call `glutSwapBuffers()` or `glFlush()`.
 - *Hint: Use two triangle fans. One for 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 F (2 points):** Draw your cone. You should just be able to call your cone function from Part E in your display function. Rotate it so that the tip is at $(0,2,0)$, with base on the $y = 0$ plane. **Do not change your cone function from Part E to do this!**
- **Part G (2 points):** Setup an **idle callback** that causes the cone in Part F to rotate by a small amount each frame around the y -axis.

Note: You will get strange or unexpected behavior as this cone rotates around the y -axis. This is fine. Fixing this problem is the extra credit.

- **Part H (2 points):** Draw another cone with base centered at $(x,y,z) = (3,1,0)$, with tip at $(3,1,2)$. Reuse your cone function from Part E, but **do not change the function!** You should have a single cone function that you call twice, once for Part F, once for Part H.
- **Part I (4 points):** Draw a **yellow** teapot and place it somewhere (other than the origin) so it's visible in the scene. Make sure it does not overlap with the cones.

Hint: Use `glutSolidTeapot()`, which you can read about on page 688.

Extra Credit (2 points): Why do the cones drawn in Parts G and H look strange? Put the answer in your README. Fix the problem in your program, and add a keyboard callback (using the 'x' key) to switch back and forth between the fixed and unfixed versions. To get credit, the discontinuity apparent in the cones must go away.

NOTES:

- A "README" file is required in order to get full credit!
- When submitting, please submit *a single* file (as a .tar.gz or .zip archive).
- Posting images of your assignment on your webpage is required in order to get full credit.
- You can add additional geometry, 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 for this homework.