

Lab Assignment 4 – CSE 332/564, Spring 2008

Due: Thursday, March 26, 11:59pm

This lab will introduce you to 3D rendering. Continue to use the code you developed for lab 1. Upon completion of the assignment you will submit the following, via blackboard:

- the complete software (all that is needed to build the executable: source code, project files, etc)
- an executable (.exe file) of your work
- a comprehensive report that illustrates with images and narrative text all aspects of your work
- any images not part of the provided collection that you have used for testing your software

All of these components are equally important. Please note the policies posted on the class webpage.

The overall theme of this project is to create functionality to render a box made of 6 polygons. This may not be an overly interesting object, but in lab 5 you will insert some more interesting content into it. For now, you will learn how to build the box, how to 3D render it, and how to map textures onto it.

1. Update your software

Visit the lab page for new code that will make the task easier. First, use the solutions file, build an executable and run it. You will see a wireframe square in the display window, which you can rotate and zoom in/out by left/right clicking the mouse in the display window. Now check out the code base. The rotation is enabled with a trackball interface.

Integrate this new code into your existing software and see if it can still perform the functions you had before. All that should change is that you can now rotate the polygon with the trackball interface.

2. Build the box

Add a button to the File menu, such as 'Read Volume' or the like. You will use it later to read in volume data files. For now you will just build a box. There are a number of volume data files posted on the lab page. The header file is similar to the ppm header. It is as follows:

```
P7 -- indicates a volume data file
nx ny nz – the dimensions
255 – the density resolution
data (there will be one byte per data element, not the three (RGB) bytes you had for the images)
```

Use the identifier P7 to decide on read time that you will be dealing with a volume (as opposed to an image). For now, just read these first 3 lines. Build a wireframe box according the (nx, ny, nz) dimensions. Make sure that you can also read and handle data files where the box is not a simple cube (such as lobster.vol). Use the trackball interface to rotate and zoom into/out of this box.

3. Shade the box

Now do some lighting and shading. Have a look at the class notes and enable both faceted and smooth shading. Also enable the depthTest. Play with varying the vertex colors and appreciate their interpolation

on the rendered box faces.

4. Play with the GL Depth Test

Check out the web page

http://www.opengl.org/documentation/specs/man_pages/hardcopy/GL/html/gl/depthfunc.html

It describes the GL z-buffer (depth buffer) depth test. Default is GL_LESS which means that the nearest polygons (as seen from each pixel) get displayed. If you set GL_GREATER you will display the polygons most distant from the viewer. Since these polygons will be back facing they will not be lit. Increase the ambient lighting to see them better (or reverse the polygon normal vectors).

5. Read the rendering results

For the next lab assignment you will use the results obtained with projecting the near and far polygons for rendering. For this you will need to read the rendering results into your program, using `glReadPixels()`. Note that this will only produce bytes, for 32-bit floating point precision you will need to use Frame Buffer Objects (FBO). We shall leave this for later as an option.

5. Texture map some images onto the box

Implement a hybrid of your previous and this current lab assignment. Read in an image file and build a box based on its dimensions. Now map this image on all the six faces (this requires texture mapping, see class notes). Now you can use the program you already have to render it.

For CSE 564 students (10% extra credit for CSE 332 students)

1. Implement functionality that allows the user to rotate the light source just like the view point. You will need to add two toggle buttons on the canvas to decide what is currently controlled with the mouse interactions.
2. Add perspective viewing by adding `glFrustum()` – right now the code only supports `glOrtho()` for orthographic viewing. Use the provided slider to change the eye-distance (closer eye positions cause greater perspective distortions). Fix the view plane near the volume such that the volume does not shrink too much when the eye moves closer.

Extra credits (10% each)

1. Implement translation, using the middle mouse button. You can calculate the offset in screen space, multiply this by the model view matrix to get the 3D offset, and add this 3D offset to the trackball.
2. Implement a dash board with sliders where the user can set the light's RGB, the ambient RGB, and the k_a , k_d , and k_s constants used in the OpenGL illumination (see class notes on how this can be set in OpenGL).