Lab 4 OpenGL II

CS123 Fall 2010

1 Introduction

Last time we explored the basics of OpenGL. You can create simple shapes using GLUT, translate, color, light and shade your shapes, and position and orient the camera. Last time we only rendered a still image. This time, we will compose multiple transformations and create simple animation.

The purpose of this lab is to get you accustomed to the types of transformations you'll be implementing from scratch in the two upcoming assignments. In *Camtrans*, you'll implement camera transformations (including the perspective transformation). In *Sceneview*, you'll implement the object transformations. We'll cover the gory math in lecture and in the algo homework. Your goal for tonight is to discover how all these things work at a higher level.

2 Setting Up

Type git pull to get the latest version of the lab code. Resolve any versioning conflicts and commit your changes before beginning. By now you should be getting familiar with Git; if you have any problems please ask a TA for help. You will do most of the work for this lab in Lab04.cpp and Lab04.h which can be found in the *labs* folder.

Now that you have finished Shapes, you should put your design to the test. In this lab (and from now on), you should use your own Shapes classes instead of the GLUT toolkit functions we used last time. It was intentional to incorporate all of the code for this class in one project; it should be trivial at this point to retrofit your Shapes code into your lab work. If this requirement makes it apparent that your Shapes aren't flexible enough to reuse, you should fix the problem before moving on. If you haven't finished Shapes yet (maybe you are using a late day?), you can use GLUT this last time. Note that you'll be required to use your Shapes in the Camtrans and Sceneview assignments!

Thanks to the magic of polymorphism, your OpenGL initialization code (including your camera code) is inherited from Lab 3. Compare Lab04.h and Lab03.h to see what's inherited. If you drastically altered Lab03.h last week, then you may need to make the corresponding adjustments to Lab04.h. This lab does require a working implementation of Lab 3, so if you haven't finished that yet, you should do so before starting this lab.

3 What to Do

3.1 More Objects and Transformations

In the last lab, the only transformation you used was translation. In this lab, you will learn about rotation and scaling. OpenGL provides you with two functions to accomplish rotation and scaling: glRotatef[1] and glScalef.

For now, copy over your paintGL method from last week into Lab04.cpp. Modify paintGL so that a rotation and a translation is applied to your shapes, and then try reversing the order of the transformations. You'll see that the order matters.

Specifically, you might notice that the transformation you apply last is actually the first one that gets applied to your object. So if you wanted to scale and then rotate you'd first make a call to rotate, and then to scale. Also, remember that matrix multiplication is, in general, not commutative. Since each successive transformation is represented as a matrix multiplication, transformations are non-commutative. This concept is easy to see in another way. Imagine yourself standing, turning right, then walking forward. Return to your original position and orientation, walk forward, and then turn right. The end result, of course, is different.

Now replace the translation with a scaling. Try reversing the order of the transformation once again. What happens?

3.2 Simple Object Animations

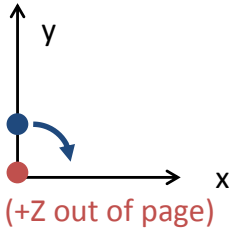
In the support code we gave you last time, we declared a timer, and in the paintGL method, a variable: *time*. This variable corresponds to the time elapsed since we started running the program (in seconds). We will use time as the basis for our animation. Recall that paintGL actually gets called every time a frame is drawn to the screen (normally 30 times a second). You probably didn't notice this last time because your objects were stationary. We are using OpenGL in immediate mode, which means that your scene, including all transformations, must be respecified at each call to paintGL. Thus, a transformation like glTranslate3f(0.5, 0, 0) results in a static translation, not a translation that continually

[1] glRotatef operates in *degrees*, while sine and cosine operate in *radians*.

increases by 0.5 in the x direction, even though glTranslate3f is being called 30 times a second.

Think about how the transformations you perform will look with respect to the camera's position. Your camera should be aligned with the negative Z axis, facing the XY plane, with the positive Y axis pointing up (just like we did in last week's lab).

Now, make it so your scene contains a single sphere at (0, 0.5, 0). Make your object orbit about the Z axis at a radius 0.5, starting from the initial position described above. Note that you will need to compute your transformations as a function of time, and you will need a translation and a rotation. You may have to play around with your parameters to make your translation and rotation look nice. The following diagram might be helpful; below, we are looking down the –Z axis (into the page). The blue dot represents your object, and the blue arrow represents the direction of motion. Before you start coding, think about the order in which you are going to do the transformations.



3.3 Animating the Camera

Your camera can be transformed just like any other object. The proper way to translate the camera is to modify m_camera in your paintGL method and then call updateCamera (from the paintGL method) when you've made all your changes. You should only call updateCamera once from paintGL!

You can change where the camera is located and where the camera is looking. The aspect ratio is determined by the display size, and you also probably aren't going to want to adjust the near and far parameters (we'll learn about those in lecture soon!). Use the camera functions you explored last time (gluLookAt, gluPerspective, etc.) to effect your camera changes. Make the camera rotate around the Y axis while always looking at the origin.

You have reached Checkpoint 1. Show a TA your work before moving on.

3.4 Putting it all together

Now that you are familiar with the basics of transformations in OpenGL, we'd like you to *make something cool*. Experiment with different transformations and try to compose a scene. Use animation to make your scene interesting. We're not asking for something incredibly complex. As a guideline, your scene should contain between 3 and 30 instances. Here are some ideas:

- Animate multiple objects in the scene. Compose translation, scaling, and rotation to create complex transformations.
- Try using a loop to create many objects; group objects between calls to glPushMatrix and glPopMatrix to apply transformations to only those objects.
- Add color to your scene by using functions like glColor3f.

You will be working with OpenGL extensively in upcoming assignments, and you may want to use OpenGL for your final project. Although we're not expecting that much from you now, you should make a good-faith effort to familiarize yourself with these basic aspects of OpenGL.

4 Finished?

When you've finished your lab, commit all your changes to your local git repository and show a TA your work. The TA will give you a unique code that you will enter during the lab survey, which can be accessed at **http://vxcv.com/8wrvh**. You must complete the lab survey and enter your completion code in order to receive credit for this lab. You have until next Thursday to be checked off.
