CS 404: Lab 5: Adding Graphics with OpenGL

Getting Started

1. Boot into Linux. To do this, click on "shutdown" and then select "restart." The computer will begin restarting. You will eventually have the option to select either Win2K or Linux (use up and down arrows to switch, press enter to select).

2. Open a web browser and go to the course website[1]. Download RAD1Dgl.tar.

3. To unpack the tar-file, type tar xvf RAD1Dgl.tar in a terminal window. Cd into the directory "RAD1Dgl."

Adding Graphics to RAD1D

1. Open the Makefile. RAD1Dgl calls GLUT and OpenGL routines. To get the code to compile, we need to use the -I option to tell the compiler where to find the header files for GLUT and OpenGL. We also need to set the LIBPATH to point to /usr/X11R6/lib (the library files for the X-Window system, Version 11, Release 6). Look at all the libraries! If our code only calls GLUT (libglut.a) and OpenGL (libGL.a), then why must we call so many libraries?

2. Type make to build rad1dgl. When you run it, it will ask you for the usual command file, plus the number of time samples you want. Choosing "medium.cmnd" and "100" produces nice results. You may also specify both the file and number of samples on the command line (rad1d medium.cmnd 100). Run it a few times, play around with the parameters in the command file. Does adding graphics help you understand the meaning of the various parameters?

3. Now we'll see how it all works. Open main.c in an editor and scroll down to the subroutine main (at the bottom). Most of the subroutines called in main are in the GLUT library (they start with "glut"). I won't go into all of the options available for these calls, but will try to point out the general ideas. The first call, glutInit, tells GLUT to start up, and the second, glutInitDisplayMode, tells GLUT which

[1] www.cs.cornell.edu/Course/cs404/2002sp

graphics model we want. GLUT_RGB and GLUT_DEPTH are the two simplest possibilities. Another possibility is GLUT_DOUBLE which "enables double-buffering," a graphics model used for animation. After we set up GLUT, we create a window by describing its initial size and position. We then register the two callback functions: DrawC for general drawing, and ResizeIt to handle window resizing. We then call the routine InitGL (in GLout.c) which initializes OpenGL. InitGL sets the background color of the window to black and sets-up a standard view for 2D data. Finally, we call RAD1D, and when it finishes, we turn control over to GLUT by calling glutMainLoop. This is all pretty standard stuff. The best way to get started with OpenGL and GLUT is to find some sample code (like this, or check out www.opengl.org) and change it to fit your needs–that's what I did!

4. Scroll up and look at RunRad–it should look familiar. Find the start of the main while-loop. At the start of every iteration, RunRad checks whether it is time to sample C. If it is, it calls the function AddSample (in GLout.c). Where is the data stored? Which routines do you think can access this data? Does this seem like good programming?

5. From the point of view of GL, RunRad's only purpose is to get data for the plot. Look at DrawC in GLout.c. DrawC takes our data and plots $C(x, t_j)$ vs. x at several time levels. The outer for-loop iterates over the available time samples. For each one, we pick a color by giving levels of red, green, and blue to the OpenGL routine glColor3f. Color levels range from 0.0 to 1.0. We then tell OpenGL that we're going to plot a series of line segments. The points are placed using glVertex2d. When all the points are placed, we tell OpenGL we're done with the line. How are the points specified? What do A, B, C, and D do?

6. It is tempting to think that DrawC produces some static picture. Put a print statement inside DrawC (something like printf("In DrawC\n");) to let you know when you've entered that routine. Rebuild and run. Trying moving the graphics window around, minimizing it, moving other windows in front of it, etc. When does DrawC get called?