OpenGL Conversion Narrative

Artifact Description

Computer Science 330 introduces graphics and visualizations with an overview of the basic used of the OpenGL graphics library. The class is based in C++ and is restricted to the Windows platform due to the specific versions of the libraries required for the class. The final project in the class is to create a representation of a mundane object using OpenGL which meets certain requirements. It must meet levels of complexity, incorporate a texture, and implement some level of navigation to enable camera and light movement.

Justification for inclusion

As a developer, I can work with C++. However, it is not my "native" language. While working through this class I was able to understand the code I was writing well enough to get it working and to do well in the class. However, I did not understand the language well enough to develop truly modular and reusable code. I'm not proud of the fact that the entire program is wrapped up in a single .cpp file that is very long, unwieldy and difficult to read.

I'm a Java developer, I've been one for more than a decade, and I understand the language far better than C++. In my mind I can very easily create a more modular and reusable application. While the specific OpenGL libraries are implemented slightly differently, they are functionally equivalent. In addition to creating a more portable application that can run on any platform, it is an opportunity to learn more about lighting and add some minor features that will be interesting.

Coverage of Course Objectives

The changes included with this reengineering effort will meet several of the suggestions outlined for the work to be performed in this category. The new application will run on new operating systems which were not possible in the C++ version due to the way Java functions in a run-time environment as compared to C++ which is linked directly to the OS after being compiled. In a manner of speaking, creating a more modular design in Java will add a level of complexity while simultaneously creating a more maintainable application.

In addition to creating a more portable and modular codebase conversion of the application to Java created a better learning opportunity for me. While it is relatively easy for me to understand C++, it's not natural and takes some effort. Much of the time I could have spent learning OpenGL was spent figuring out how to make things work in C++. In Java, I am able to concentrate far more on OpenGL as the syntax is natural for me. Thus, in the Java version I'm able to implement new features like moving spotlights, better reflection and motion of the object, and a few more features.

In addition to the new OpenGL features, I'm able to make more generic functionality in the Java code that can support varying OpenGL options. For example, an OpenGL object can identify as either needing a texture mapping, or a color palate and the appropriate logic for that rendering is called. This means

the OpenGL environment in my Java implementation can render different types of objects in the same space without much effort.

Reflection

Working through the class, I was continually frustrated by the class being specific to Windows and working in a language with which I'm not all that familiar. After submitting the final, I had a desire to implement the application in Java as it is far more familiar and may allow me to learn the concepts of OpenGL graphics rather than being frustrated by the limitations of the language. Ultimately this proved to be true, I understand shaders and texture definitions better as well as vertex definitions and more.

Conversion of a simple application from C++ to Java is a straightforward process. Having a decent understanding of both languages it is essentially a process of recreating functionality from one to another. One level of complexity comes when the application being ported uses a third-party library. In this case, OpenGL creates a native C++ library which can be imported and used in the code. However, they do not create a native Java library. OpenGL libraries for Java are implemented and provided by third parties which can adds a level of complexity to the conversion. Rather than using direct datatypes, they are often named with slight variations. The re-engineering of this application required a fair bit more research that I'd originally anticipated.

Another interesting affect is the difference in rendering the graphics. I'm unsure if the rendering differences are due to the platform or due to differences in the library implementations. The textures and sizes are identical between the two renderings. While there is a slight change in the lighting, that change should not be enough to create such a difference between the two renderings of the object. It is something that warrants further investigation.

The images below illustrate the differences between the rendering, the left is C++ in Windows, the left Java on Mac