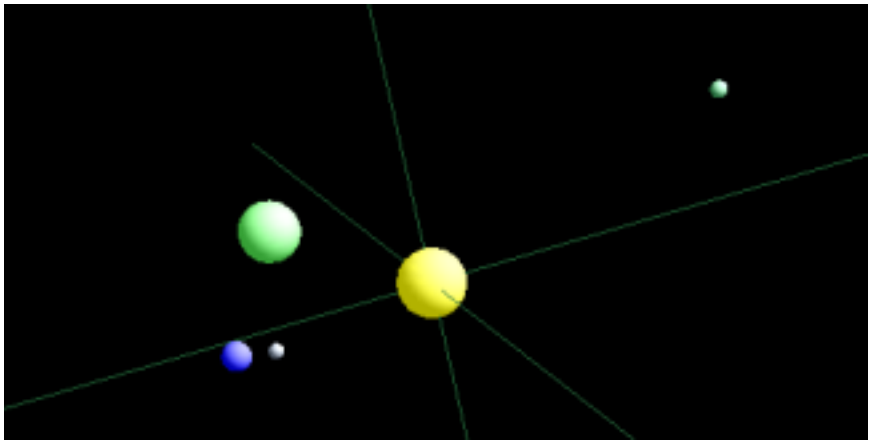
Orbiting Bodies Model in OpenGL



My concept was a model of orbiting bodies quite similar to our solar system. It was implemented using Python-2.5.2, PyOpenGL-3.0.0, and GLUT-3. My personal goals for this project were to increase my conceptual understanding and my practical understanding of 3D modeling using the OpenGL API.

To address my first goal I began by reading over the first chapters of the OpenGL SuperBible. I fond that immensely helpful because of how it introduced me to to modeling. It started by explaining how spacial arrangement affects the final perspective of a scene. It lead into the basics of 2D modeling and then 3D modeling. Along the way it introduced relevant snippets of OpenGL code.

I didn't stick to the SuperBible only though. I also downloaded code from other people on the internet. With their code I was able to play with the elements of an already working program and see how it affected the model. I found that quite helpful.

My final project began from a simple model at first. The Cartesian plane used in my model, as well as the viewing angle, are original from my first real “play time” with OpenGL. I began with the Cartesian plane and then graphed a square using primitives. From there I added orthogonal faces to the square to create a cube. The viewing angle present in the final project is the angle chosen to see the cube from a 3D perspective, instead of a single flat face.

The most important conceptual thing I learned was how the code of an OpenGL program is run through the API. Unlike in a non-graphical application where a method is only executed when called I found the “display()” method in an OpenGL application is called again and again – as soon as the scene has been rendered the display() method is run again. Abstractly, this is functionally equivalent to having a non-terminating loop in your application. With that concept grasped the rest of my model was rather simple to implement.

I created a class “Planet” to store the data for a major satellite and cut down on code clutter. I also created a child class “Moon” that contained a modified “render()” method which did not call glPushMatrix() or glPopMatrix() as the parent Planet class would. Each time the display() code was executed the Planet instances would update their positions and in turn call their own renderMoon() methods for each of their registered moons.

Some helpful side effects of the Planet and Moon classes were that the “Sun” is actually just a Planet Instance with an orbit diameter value of 0. Also, because Moons are also Planets it would require very little effort to modify the Moon class to allow them to have their own registered Moons.