**Programming Project 3 Report**

Student Name: Robert Johnson

Student UAID: 010796992

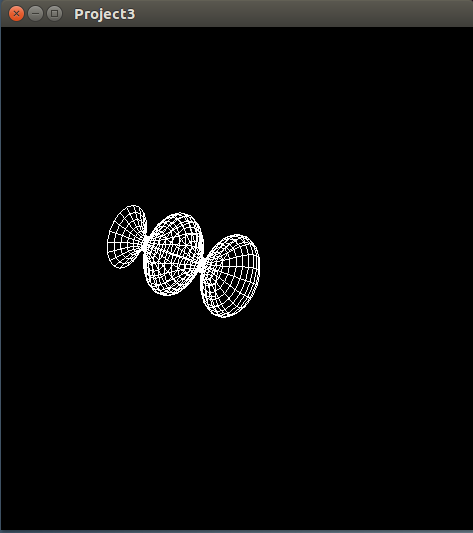
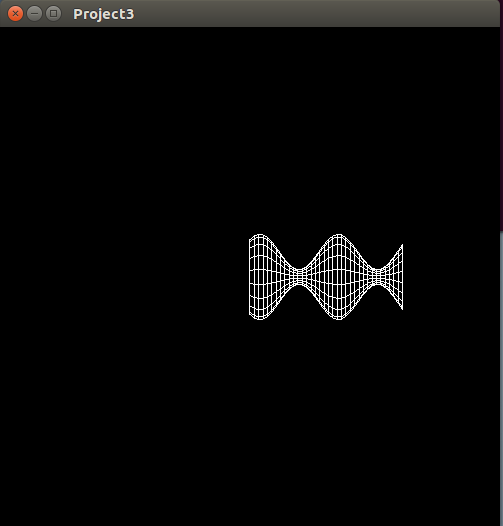
**Problem Statement:**

The goal of this project was to start working with 3d objects. The first two projects consisted of working with a 2d Tetris game and starting to learn OpenGL. With this project, the z-axis was introduced. The motion callback was also something new we used with this project. The problem we had to complete was to draw a y=f(x) curve and turn it into a 3d object by rotating it over the x-axis. This was used with some fancy formulas we learned earlier. Then we used the mouse and motion callbacks to let the user draw their own curve and then when the mouse is released, it will turn that curve into a 3d object. You can use the keyboard callback to rotate the objects over the x,y,z axes to look at them in awe.

**Design:**

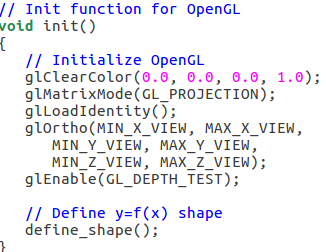
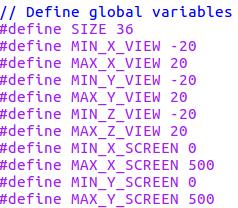
When looking where to start my design process, I saw that the object.cpp sample code on the class repository looked very helpful with learning how to rotate a 2d line/curve over an axis and making it 3d by introducing the z-axis. I then found the formulas on how to rotate over the x-axis which was asked of us. Then for task 3 when asked to let the user draw a curve, I saw that the rectangles.cpp sample code was going to be quite helpful on learning how to use the motion callback and defining the 2d arrays and other variables that were going to be useful.

I decided to use -20 to 20 for all of my x,y,z axis on the glOrtho() command. The y=f(x) formula I came up with was y = cos(x) + sin(x) + 2 where x = [0-4pi]. I also decided to use 36 as the number of sample points, I would have when creating my “wire-frame” for the objects. A good number that is a multiple of 360 degrees. Below is what it looks like when the program first runs and what it looks like when you rotate it around.



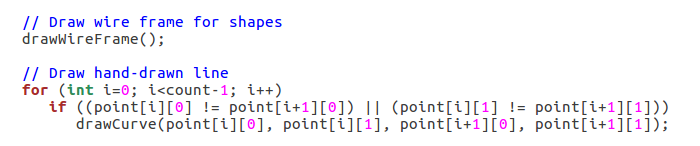
**Implementation:**

For the implementation process, I used the object.cpp sample code as the base that I worked from. One of the first things I noticed I was going to have to do for the hand drawn line was to define a MIN and MAX x,y,z views and window sizes in order to correctly use the mouse and motion callbacks so that it picks up the correct (x,y) coordinates.



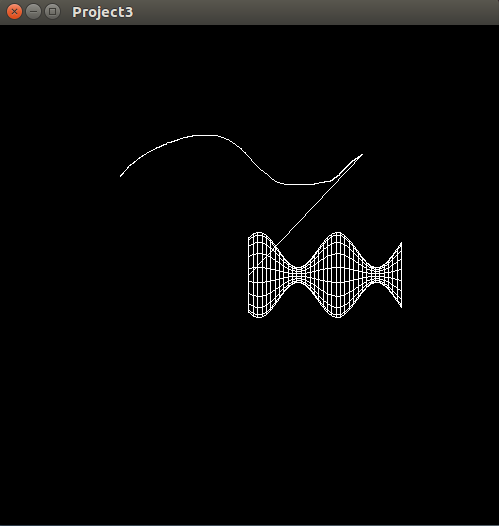
In the above screenshots you see how I implemented that into the glOrtho() function and you also use those global variables in the mouse and motion callback in order to store the correct (x,y) values when the user is drawing their line. This was not too hard, I got most of this from the rectangles.cpp sample code and implemented it into mine.

The next task that took up most of my time was then setting up how to correctly use the mouse and motion callbacks to draw and show the line correctly. Eventually I realized I would need a 2d array to store values when the user moves the mouse. I had to ask help from Professor Gauch, and it was just small things I had wrong like not incrementing my count value in the correct spots and my for loop needed to be i<count-1 instead of just i<count. Below is the screenshot of what I am talking about.

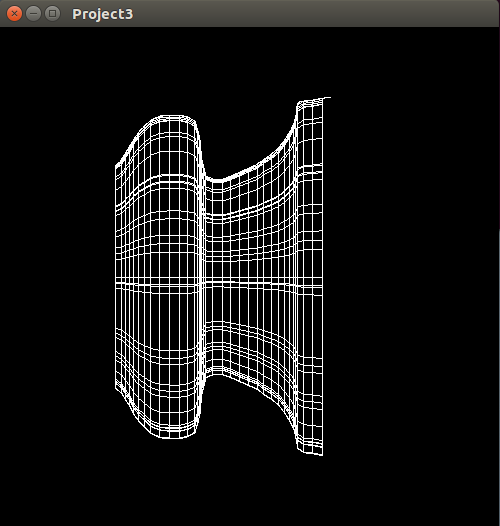


**Testing:**

Once again, I used a Virtual Machine running Ubuntu to compile and test my code. Testing and completing tasks 1 and 2 were not too hard, no big bumps in the road there. Task 3 is where I got most of my issues when testing. For example, like I mentioned in the implementation section, I had a wrong condition in my for loop that was making it go out of bounds and it left this nasty line from (0,0) to wherever the end of the curve I drew was. Also, you will see in this screenshot, I had not yet figured how to remove the original y=f(x) object after drawing the curve and rotate the new hand-drawn curve.



Below is what it looks like after adding/changing two lines of code and drawing a custom curve:



However, the above screenshot still was not my finished product. This was just after drawing the curve, and although it looks good from this angle, once you started rotating it, you would have noticed the wire frame was not working as intended. I learned this was because I had changed something for theta that was not supposed to be changed when doing the object of revolution. Also, I had some wrong conditionals in some of my for loops. But in the end, I got the whole program up and running as intended. I didn’t do the bonus part to smooth out the hand drawn objects of revolution but I think everything else was a success and looks to be working as it should.

**Conclusions:**

Overall, this project was not too hard or time consuming. The sample codes we have were a lot of help and kind of laid out the groundwork on how to get all of this started. Just picking out bits and pieces from a few sample codes and changing a few lines got most of it done to be honest. I really enjoyed finally moving into working with three dimensions and figuring out how to use the mouse and motion callbacks was very interesting as well. I think this project made me learn a lot about OpenGL very quickly. As a programmer just looking at code is not always easy to pick up on things but with sample codes to look at and getting help from someone who is very experienced with it, once you are shown the mistakes in your own code, everything just clicks, and you feel very dumb for not realizing how simple some of your mistakes were. Also, it was nice to use some of the annoying things I learned in Calculus that made me angry because I thought I would never use any of it. Also, as someone who is not artistic or very creative, and is used to being the computer science math nerd, it is cool to be able to use math to create such neat things to look at.