Graphics Programming Project 6 Report

Luke Miller
010785453

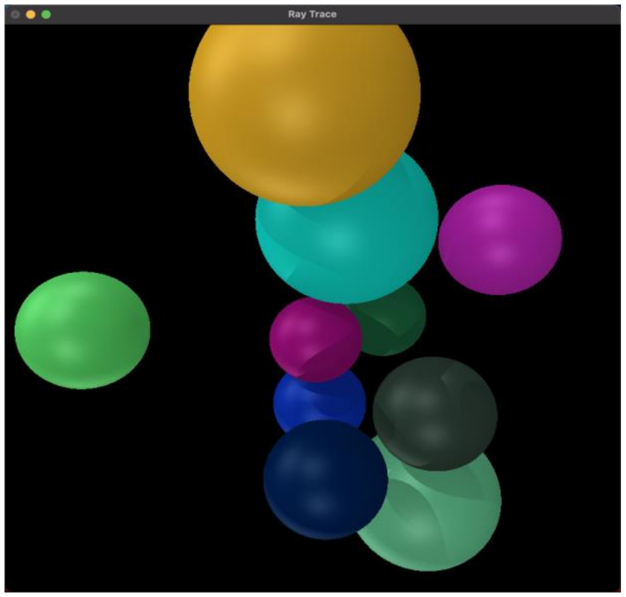
Problem Statement:

The goal of the programming assignment is to use OpenGL to extend an existing ray tracing program to learn more about ray tracing internals as you add several new features. It was required to add **two** features to the "ray_trace" program in the CSCE 4813 "src" directory. There are no inputs to the program other than starting it up and watching the shapes move. There are a few places of error handling that keep the shapes on the board and don't allow them to float off screen. For example, when a sphere starts to float off the screen it changes directions and turns the opposite way in all three dimensions.

Design:

When I designed my program, I first decided to duplicate the ray_trace2.cpp file to give me a good starting point for the added features. Then I needed to move some data structures, so they were globally accessible to everyone. These structures were the light source direction, the spheres, the colors for the spheres, a global flag, and finally a new Point3D array for the trajectory of the sphere's movements. Then I created two new functions that pulled the initialization for the spheres and the light sources into their own respective functions. This meant every time the ray tracing function was called those parts weren't reinitialized. Finally, I added the timer callback to give an interval of when to move the spheres, as well as a move spheres function that moved the spheres along their given trajectory and bounced them off walls if needed. This all made for relatively easy work, and once I knew how I was to design it, I pulled it off quite easily.

Implementation:



To get started I moved some things around in the ray trace 2 program to allow for easier access to global variables. The given code was very adaptable and modular to start. Everything I rearranged was from the ray trace function. They didn't need to be initialized every time, so have the spheres, their colors, the light direction, and their color all be global worked a lot better and may have helped with efficiency. The refactoring probably took me less than 20 minutes in total when I knew what I wanted to do. I wasted quite a bit of time debugging the wall bouncing, and my code was supper inefficient. I then remember in paradigms we did a cops and robbers game where the cops bounced off the edge of the screen, funny enough I used that old code for inspiration for the trajectory code, as I was already out of ideas late that night. After implementing that, the bouncing worked a treat. Then I wasted around 3 hours try to implement the cylinder shapes. I think I got really close, but the cylinders were the entire height of the screen, and it looked real bad. So then as to not waste any more time, I decided to pivot to feature 4. I had tried feature 4 before doing the cylinders but to no avail. I after trying again I got feature 4 to work within an hour! I was missing the part where you add all the colors together from the light.

Testing:

I tested my program consistently. After almost every change I made I would make sure that change worked as expected. The testing got kind of tedious as I was testing for the wall collisions. I finally get them work well after my inspiration from my paradigm's homework from over a year ago and used the testing to approximate where exactly the walls were so the bounces looked nice. Then I test the various light sources many of times. Often the lights were too bright resulting in some psychedelic looking pictures but toning the color of the lights down help a ton. In the end everything tested and worked well, except I never got the cylinders for feature 2 to work, I got feature 1 and 4 to work very well.

Conclusions:

In conclusion, this program was a success. Everything works as expected. I am very proud of the modularity of the program and I believe this was a great experience with ray tracing. Knowing what I know now I wouldn't have wasted time trying to undo the cylinders, and just tried harder with feature 4 out of the gate. If it weren't for that the assignment would have taken me less than 2 hours. Two hours with debugging isn't wholly that bad and I am proud of how it turned out.

P.S. Thank you for the wonderful semester. I know you're very tired of the remote stuff, but this class was fun. It has probably the most fun I have had programming in a long while. Thank you for that.