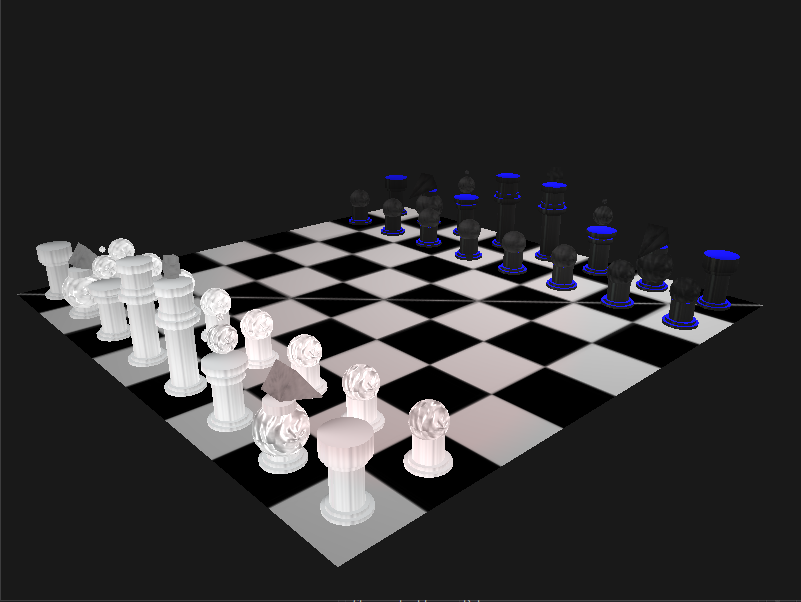
John Elbogen

CS 330

4/17/2021

**Project 2**

For my 3d scene, the following is the outcome of my recreation of a 2d scene in 3d:



My development choices were quite simple, I chose to first create the basic pawn shape. This piece is the staple of a chess set, and the simplest shape of all the pieces. To create a pawn, I used two basic shapes, a cylinder and sphere. For the base I used two layered cylinders, which matches the scene on the left. To accomplish this, two cylinders were created, and I utilized a cylinder.cpp file but changed my function inputs to have a slightly smaller radius and height. I did the same for the length of the pawn structure with a third cylinder. To re-use objects, I re-drew the same base for each piece, rather than create entirely new cylinders each time. This saves time and coding. Next, I utilized a shapegenerator.cpp file to create a sphere. This created the head of the pawn.

I now had a pawn piece created, and kept this code organized together with comments. To texture the pawn, I bound a white marble texture found online, using glActiveTexture, and glBindTexture. All that was left was to create a loop, which duplicate and changes the positioning of the pawn 8 times. This was a simple loop, that changed the pawn position on each iteration, as well as created a new pawn on each iteration. This was the basic format for all pieces but, I will comment specifically on the knight, and kind piece. For the knight I chose a pyramid shaped head, and I decided to write my own pyramid function. The same process was done for the cross above the king’s head, but a rectangular prism function. These both were based on the plane function that I wrote for this project and were written similarly.

These design choices allowed me to edit shapes individually, and group together each piece in an organized way.

This 3d scene can be navigated easily. The input device in mind is a keyboard, therefore any input device which can map to a keyboard function could be utilized. A, W, S, D, keys are used to move the camera in a direction, while the Q, E, navigate upward and downward. Using a mouse, the user can look around and move the camera. The mouse moves the camera position which also uses a spotlight that updates to the camera’s coordinates. Moving the mouse will allow the user to shine a red spotlight wherever they look at. By pressing the P key, the user can change perspectives from normal to ortho mode.

To elaborate on custom functions used, I created several functions to organize and easily edit my custom shapes. For example, My UCreatePlane() function takes mesh data, then defining vertex data, setting the vertex positions of the shape, generates a vertex array, and binds it. Next it creates a vertex buffer array, and finally creates vertex attribute pointers. Simply put, it sets up the shape which can then be rendered. Next, we consider the URender() function, which takes in a texture, translation coordinates, and radians. We pass, texture to allocate our chosen texture, coordinates to move the shape around, and a radian amount, which rotates the shape accordingly on the z axis. All this code is now modularized and easily called in the render loop. It can be called as many times as necessary and passed any data needed. This take in more variables but is currently customized to fit the needs of the scene. An example is potentially passing in, x, y, z coordinates for the axis of rotation, which I did use in my render function for the rectangular prism.