Peter Bou Saada

CS 330: Comp graphic and visualization

Prof. Brian Holbert

December 12th, 2021

**Reflection:**

I chose cubes and planes because they are the easiest objects to start with. I chose cylinders because it allows me to dynamically allocate memory on the heap and calculate the locations of all vertices, textured coordinates, colors, and indices. This would help me prove that I understand the material of this course well. I chose to add the pyramid last second to test how easy it would be, it could be done in as little as 2 lines of code.

I modularized my code a lot and separated everything into its own classes. Had separate classes for my window, my renderer, vertex array, vertex buffer, index buffer. Then I used inheritance to create a header only class named Shape. This class served as a container for the different shapes I have like the cubes, planes, pyramids, and cylinders. Then I created a RenderObject class that held a vector of shapes. This class is mostly there for organizational purposes. However, it is also used as the parent object to have objects rotate around the center of it, instead of their own center.

Navigation around the scene is made by manipulating the projection and the view matrices. We pass those 2 matrices to the shader which then calculates the position of objects. Contrary to popular belief, there is no such thing as a camera that moves around the scene, we simulate that by moving all the objects opposite to the camera movement. For example, if the camera “moves” along the positive x axis. All the objects must move along the negative x axis to simulate the camera moving correctly. We combine this thought process for x, y, and z axis, and we get camera movement. We use the mouse position to calculate the camera tilt, up, down, right, and left. We pass these to the view matrix. The controls to move forward, backward, right, and left respectively are W, S, D, A. The controls to go up and down respectively are E and Q.

There are 3 major functions in my application. The constructor for the Window class, the constructor for the Renderer class, and the Render function in the renderer class. The Window class is responsible to create an OpenGL context if none exists, then create a window and set all the correct callback functions for controls. The Renderer class constructor is responsible for creating the VAOs, VBOs, and IBOs for every single object.

The render function loops through all the objects, and clears the buffers, recalculates all the matrices that need to be recalculated, binds the right shader, vao, texture, and makes the draw call for each shape. Then unbinds the VAO.

All of these methods are reusable; however it is probably a good idea to make the renderer a singleton to prevent having more than 1 renderer, or creating a new gl viewport to render to. The Window class can be reused to create new windows.