Ryan Reames

CS-330-T3336

2/15/2023

CS 330 Project Reflection

For my project, I created a scene by photographing some painting supplies that I keep for my occasional hobby of oil painting. I recreated these objects, consisting of a desktop, canvas, two brushes, a pen, and three paint tubes, using Blender and then imported the objects to OpenGL. By selecting these objects, I was able to implement a variety of shapes and textures to create (in my opinion) an interesting scene that exceeds the requirements laid out in the project rubric. While the objects were more complicated than cubes, cylinders, and spheres, they still incorporated these basic shapes to some degree, such as the large brush including a cube for the bristles, paint tubes using cylinders for the main body, and so on.

To create the scene in OpenGL, I was able to implement various aspects of the CS 330 tutorial code. Code that I adapted from the tutorials includes the basic window, startup function, main loop, mesh creation, and shaders (both shader code and generation). I also adapted controls provided in the tutorials to meet the movement requirements of the project. Specifically, using W, A, S, and D allows the user to move forward, left, backwards, and right. Q elevates the camera position and E lowers it. Using the mouse changes the direction of the camera, and by scrolling up the user can increase W, A, S, and D movement speed (or slow it by scrolling down). An additional movement feature I implemented is on left mouse button click, the user’s camera position is reset to the original location and angle.

By implementing mesh creation, rendering, and shader creation outside of the main loop, I was able to create the program in such a way that multiple objects can be rendered with little in a way of additional code. If one were to want to create additional objects, they would simply need to call these functions in the main method to do so (though the code could be refactored for positioning, rather than needing to add additional lines for each object position).

The most complicated part of this project was implementing the Blender objects I created into the OpenGL render. Originally, I began by creating an OBJ file in Blender, then reading the vertex coordinates and copying / pasting them into the create mesh function. This led to issues in that there were thousands of vertices and indices being pasted into the source code, creating a rather large file. Once textures were required to be used, this strategy did not work as Blender OBJ files use different indices for vertices, normals, and texture coordinates. To solve this, I researched online and found the site <http://www.opengl-tutorial.org/> which had various tutorials related to using Blender objects in OpenGL.

This site was particularly useful, in that they had an object loader similar to what I was using before, but rather than just reading vertex coordinates and saving them in a TXT file, their code loaded the vertex coordinates, normals, and texture coordinates into arrays to be used elsewhere (particularly in creating the mesh for my use case). This still left the issue of determining indices for normals and texture coordinates, but they also had a tutorial and code which I could use in my project to fix this problem. Specifically, I used their VBO indexer which reads indices for vertex coordinates, normals, and texture coordinates, then compares the normal or texture coordinate indices to what was already used for vertices. If these vertex coordinates are already used for other normals or texture coordinates, the VBO indexer recreates the vertices if already used so that normals and textures are rendered correctly. By adapting the site’s OBJ loader and VBO indexer to my project, I was able to remove coordinates from my file and instead import OBJ files, leading to a much cleaner and easier to follow code base. Additionally, these two classes can be used to import multiple Blender objects into future projects.