CS-330 Project Reflection

Benjamin White

02/19/2023

My scene is a recreation of an image of a covered bridge that passes over a stream. To recreate this bridge, I utilized tall thin cubes for the pillars, a long rectangular cube for the body, and a triangular prism for the roof of the bridge. The stream and grassy ground are both represented by planes. Additionally, the sun was recreated with a large sphere placed far in the distance.

The vertices and element order of the shapes were created using two structs that are defined in the include/types.h header file. I defined a struct Vertex{} that is made up of glm::vec3 and glm::vec2 arrays. This struct contains a vec3 Position, vec3 Color, vec3 Normal, and vec2 Uv. The shapes struct consists of several inline statements that instantiate a vector of struct Vertex and a corresponding vector of uint32\_t element order, per object. Using the Vertex struct allowed me to easily identify and manipulate the xyz coordinates of a shape’s vertices as well as color data. If I decided to add an object to the scene, I would simply create a new inline statement within struct Shapes{} and specify the Vertex data. Adding the vertex data manually worked well for creating polygons. However, when I needed to create a sphere, I added inline functions sphereVertices() and sphereElements(). Again, the sphereVertices() function returns a vector of type Vertex while sphereElements() returns a vector of type uint32\_t.

In order to manage the multitude of shapes that were positioned together to create the bridge, I abstracted out the shape objects into a class called Mesh. A mesh object consists of a string “name”, vector of type Vertex “vertices”, and vector of type uint32\_t “elements”. So, if you wanted to create a new shape, you could define the vector data in include/types.h, then append the vertices and elements to a vector of type Mesh from the setupScene() function. Last but not least, we loop through each mesh in our vector of meshes, assign the texture we need by accessing the mesh’s name, and draw the mesh.

This abstraction made it easy to implement new features as I developed the project. After playing around with textures, I knew it was important to implement some lighting that would behave as though it was being reflected off the water. To do so, I decided to create a sphere to represent the sun. The sun’s position was then passed to the object’s shader so that reflectivity could be calculated. The sun’s specular strength needed to be high, which meant that the objects in shadow needed a secondary ambient light to fill in the dark side of the objects. Much like the Mesh class, a Light class was created so that additional lights can easily be added to an array and looped through to draw individually.

User input is captured using the glfw functions glfwGetKey() and glfwGetCurserPosition(). A user can navigate through the scene using standard WASD keyboard movement. Additionally, the ‘Q’ key allows the user to move up and the ‘E’ key moves the camera down. The orthographic camera is switched on and off by pressing the ‘P’ key. Mouse input is also captured. The user may inspect the in all directions. If the user wishes to increase the speed of the camera movement, they can use their mouse’s scroll wheel. To exit the program, press the ‘Escape’ key.