Ashlyn Godwin

CS 330 Computer Graphics and Visualization

Battersby

2/19/23

Final Project Reflection

In order to create a comprehensive 3D scene, objects had to be recreated with proper models, realistic lighting, and discernable shaders. Because of the difficulty that exists with modeling complex 3D objects using OpenGL, the items selected in the scene had to be easily recreated with simple 3D shapes. The image selected for this project was taken with simplicity in mind: all objects were composed of a combination of cones, cylinders, tori, planes, and boxes. The image was taken at my place of work due to ease of measurements for accurate sizing. Objects in the scene are a string trimmer box, an electric chainsaw box, a gas can, and a spool of string trimmer line. The trimmer box is composed of two cubes resized to fit. The chainsaw box is one cube. The gas can was made by two cylinders, a cone, and a torus. Finally, the trimmer line is made from a torus and a cylinder. All of these models were made using basic 3D shapes provided in the Meshes.cpp file and applying matrix transformations to change position, rotation, and scale of the object. In several cases, faces were not drawn in the rendering phase because they overlap with other faces and would therefore not render. Lighting was performed by two overhead lights in order to recreate the overhead lighting of the scene. Textures were applied to all objects due to the way the shader was written, though the string trimmer box only has a solid red texture applied due to difficulty getting the textures to apply evenly across both cubes.

Navigation of the environment can be done through a combination of keypresses, mouse movements, and scroll wheel utilization. The bulk camera movements were controlled by the W, A, S, D, Q, and E keys, which allowed the camera to move forward, left, backward, right, down, and up respectively. Additionally, the P key was utilized in order to swap between perspective and orthographic representations of the scene. The mouse controlled camera directions. This was done by implementing a mouse position callback function. Moving the mouse left would cause the camera to look left, right for right, etc. This was due to checking for differences in mouse position from frame to frame. Once a difference was registered, a camera transformation was applied to change the camera angle. Finally, mouse scroll allowed for changes in the speed of the camera. This was done by changing the value of the velocity variable. All of these functions were performed by the camera object utilizing callback functions. It should be noted that while alternative control methods were not implemented for this project, they can be implemented with the same methodology. Gamepad controls could be created by using GLFW’s gamepad callback functions and drivers. Analog controls would operate similarly to the mouse callback function (using a distance from center instead of change in mouse position) and button presses would work exactly like keypresses.

In order to make all of these disparate pieces function as a whole, several modular functions were created to provide functionality. For example, the function UProcessInput contains calls to all of the input functions (the aforementioned mouse, scroll, and keyboard callback functions). The CreateTexture function provides texture creation for the program. This allows objects to be rendered with textures and was vital for rendering the scene properly. It also contains a call to flipImageVertically, which was made because OpenGL reads image files upside down as opposed to most image types. CreateTexture also makes calls to the open source file stb\_image.h, which allows for texture creation in many popular image types. For this project, all of the file types were .jpg, but others are available. Likewise, the DeleteTexture function exists to remove textures from memory once they are done being used. Shader programs were created using the CreateShader function and released from memory with the DeleteShader function. These provided the important step of drawing the shapes to screen with information taken from the renderer. Finally, the URender function provides all of the main rendering work utilized in the program. This includes applying matrix transforms to the shapes provided in Meshes.cpp, using the correct shader program, and sending uniforms to the shader program. All of these functions can be changed and implemented in other systems in order to render different scenes.