Christopher O’Dell

SNHU – CS330

Module 7

Final Project - Reflection

My 3D scene for this project involved both simple and complex objects being used to create different low-polygon, life-like objects. The initial object created for the project was the most complex- a Nintendo Entertainment System (NES) controller. The controller was chosen because it contains many smaller individual objects to create such as the controller box, the two red buttons, the two start and select buttons, the directional pad, and the coord. Along with these objects were also backplates for most objects that help them stand out on the controller. The controller box was made using a basic square, the two red buttons were made using cylinders, the directional pad was made using two boxes, and the coord was made using a cylinder. All of the backplates for the buttons on the controller were made using planes. These choices were made to create a complex object made from a multitude of shapes, creating a single complex object. The other objects in the scene were made in a much simpler method such as the compact discs and floppy disk which are made using a singular object, spheres for the compact discs and a box for the floppy disk. The difference between these objects is that a specialized texture was used for the objects to create more realistic-looking objects.

In my 3D scene, users are able to navigate the environment using both keyboard and mouse input, offering full control over camera movement and orientation. The W, A, S, and D keys allow for horizontal movement, enabling the camera to move forward, backward, left, and right within the scene. Vertical movement is handled by the Q and E keys, which move the camera down and up along the Y-axis, respectively. To control the orientation of the camera, the mouse is used—moving the mouse adjusts the pitch and yaw, allowing the user to look around the environment smoothly. Additionally, the mouse scroll wheel modifies the movement speed of the camera, offering fine-tuned control when navigating complex scenes or larger environments. A keyboard toggle key is implemented (O and P) to switch between perspective and orthographic projection modes. This feature allows the user to change how the 3D scene is visualized, without altering the camera’s orientation or position, making it easier to inspect geometric proportions or simulate a 2D layout view.

To maintain clean and organized code, I have modularized key functionality into several custom functions. The SetTransformations() function handles all geometric transformations—including scaling, rotation, and translation—by creating a model matrix that is passed to the shader. This standardizes how objects are positioned and ensures that the transformation logic is not repeated throughout the code. Similarly, SetShaderColor() is a utility function used to assign a solid color to an object when texturing is not required. It sets the appropriate color uniform in the shader and disables texture sampling, ensuring a clean visual result. For textured objects, I use the SetShaderTexture() function, which binds the correct texture based on a string tag. This allows textures to be applied consistently across different objects without managing raw OpenGL texture IDs directly.

During initialization, LoadSceneTextures() is called to load and bind all required textures into memory, organizing the texture setup into a single function. For lighting, SetupSceneLights() defines the position, color, and intensity of the lights used in the scene, including both ambient and specular components to support Phong shading. This function ensures that all lighting parameters are consistently applied and easy to modify. The DefineObjectMaterials() function stores material definitions for objects in the scene, such as ambient and specular color values, and shininess. This helps keep material properties organized and reusable across multiple objects. Finally, the main RenderScene() function coordinates the drawing of all objects by calling the transformation, texture, material, and draw commands in the correct order. By modularizing these operations, the code is easier to read, maintain, and extend—aligning with best practices in software development and real-time 3D rendering.