**7-1 Final Project Submission**

Chris Bridges

Southern New Hampshire University

CS 330 Comp Graphic and Visualization

Professor Kurt Diesch

14 April 2025

There were a total of five objects I chose to render in three dimensions. The desk was the foundation for everything to rest on, so I chose to use a plane textured with a wood grain to simulate the wood. The second object was the oversized mouse pad on which I placed the objects. I used 2 different sized planes resting on top of each other to create the mat and the thin red border on the edge. I chose to use a leather texture for the top plane to give it a more aesthetically pleasing appearance and to convey a friction-like material. The Poké Ball was created using two half spheres for the upper and lower portions, a torus for the black ring, and a sphere for the button in the middle. I didn’t have to choose a texture for these components because they’re solid colors. The stand for the Poké Ball was constructed using a torus for the base plate, a tapered cylinder for the upright support, and a smaller torus on the top for the Poké Ball to rest on. Proper alignment was the most difficult part of the Poké Ball and base since there were many objects and interactions between them. The Rubik’s cube was constructed using a box and a texture I created in Microsoft Paint. Creating the soda can was one of the more difficult objects. I chose to use three cylinders, a long one for the can, and two very thin ones for the top and bottom of the can. I used a generic soda can texture for the can because creating one to match the one in my original image would have been far too artistic for me.

To enhance usability, I implemented an intuitive camera system that mimics movement patterns found in most first-person games. Users can navigate the 3D scene using a standard keyboard and mouse setup. The “W,” “A,” “S,” and “D” keys control forward, left, backward, and right movement, respectively. Mouse movement adjusts the camera’s rotation, allowing users to look around freely and focus on specific objects in the scene. This combination allows for fluid navigation and natural exploration, making it easy to appreciate the scene from any angle.

In addition to movement, I provided camera control features to improve precision. Users can adjust the camera speed using the mouse wheel, which is reflected at the top of the screen. This is helpful when needing either fast traversal or a slower, more detailed inspection. Users can also toggle between orthographic and perspective views by pressing the “O” and “P” keys. This perspective change provides different ways of viewing the objects. Orthographic for technical, distortion-free viewing, and perspective for a more natural, immersive 3D feel. This functionality was important to include because it offers both aesthetic and analytical views of the project. Overall, the control scheme enhances the user’s experience by offering flexibility and familiarity, ensuring that both casual and experienced users can explore the scene with ease.

One of my top priorities when designing the codebase was to keep it modular and organized. To do this, I developed custom functions that helped isolate specific responsibilities while promoting reusability. For example, I created a DefineObjectMaterials function that store material definitions for common surfaces such as leather, metal, plastic, and wood. This meant that when I wanted to add new objects, like a motorcycle in a future scene, I wouldn’t need to redefine materials. I could reuse existing ones to maintain consistency and save time. The RenderScene function was also designed for modularity. Each object, whether it was the Poké Ball, the Rubik’s Cube, or the soda can was rendered using individual blocks that could easily be copied or moved. This made positioning and reusing them in new scenes simple. I also used reusable transformation functions to manage scaling, rotation, and positioning for each mesh. This not only streamlined object placement but also made debugging much easier. If an object was misaligned, I could quickly isolate and tweak its transformation without affecting the rest of the code.