Michael Lorenz

CS-330 Comp Graphics

Professor Montalvo-Ruiz

18 August 2024

7-1 Final Project

To begin the development of my 3D scene, I carefully selected objects that would not only provide visual interest but also present an appropriate level of complexity for implementation. When selecting objects, I considered their shapes, textures, and variety to avoid overly complex shapes and maintain an appealing balance between complexity, lighting, and color. Bringing these 3D objects into the scene meant implementing several essential components to create the required functionality. These requirements include the ability to load primitive shape meshes and add them to the scene, adjust each object’s scale, position, and rotation, load and apply various textures and materials to objects, and fine-tune the attributes and UV scaling of these textures and materials. Creating the 3D objects using this functionality was relatively straightforward. For simple objects such as the cutting board, counter, background, and coaster, I could use a single primitive object for each (box, plane, and cylinder). For more complex objects, I “glued together” multiple primitive objects to make them appear as one continuous object. This was especially difficult for the glass jar and knife, but I could make it look just right with careful positioning and scaling. To add realism to my scene, I textured nearly every object to make it look similar to my reference objects. Then, I created custom materials to apply to each object and added lights around the scene to accurately recreate the lighting in my reference scene. I used two overhead lights with directional lighting and one additional light to adjust the level of ambient lighting.

To allow users to explore the scene, I implemented controls within the ‘ViewManager’ class that utilizes the user's keyboard and mouse input devices to control the camera. Using keyboard press input as events, users can pan the camera forward, backward, left, right, up, and down using the ‘W,’ ‘S,’ ‘A,’ ‘D,’ ‘Q,’ and ‘E,’ keys, respectively. I added zoom functionality through the up and down arrow keys to adjust view depth. Using the O' and P keys, the user can switch between orthographic and perspective views. Mouse input automatically calls callback functions used for navigation. Moving the cursor calculates the cursor’s offset and rotates the view left, right, up, or down accordingly. The scroll wheel also triggers a callback, allowing for granular adjustment of the camera’s movement speed.

With the extensive functionality of this project, it was important to me to maintain modular and organized code. I focused on refactoring my code to eliminate redundancy and encapsulate functionality with reusable functions throughout development. The first key function I developed was ‘DrawMeshTransformation(),’ which I used to add 3D objects to the scene and accepts five parameters: an XYZ scale vector, XYZ rotation vector, XYZ position vector, ‘ShapeMesh’ object pointer’ and the function used to draw the specific shape. To draw any primitive shape with this function, I had to create a simple wrapper class to wrap all functions of the ShapeMeshes class in wrapper functions because they have varying numbers of parameters, and I needed to make them compatible with DrawMeshTransformation. I can then pass functions such as ‘DrawCylinderMeshWrapper’ to ‘DrawMeshTransformation().’ I added the ‘SetShaderAttributes()’ function to modify all visual attributes of objects in the scene. It accepts a color RGBA vector, texture name, texture UV scale vector, and material name. With this, I can set all attributes for an object in a single call. To improve the readability of the ‘RenderScene()’ method, I reused my modular functions extensively and created additional specialized functions for drawing complex objects, including ‘DrawJar(),’ ‘DrawCup(),’ ‘DrawCucumber(),’ and ‘DrawKnife().’ Each of these takes an XYZ position parameter, allowing for easy adjustment within the scene. Using this approach to create modular and reusable functions not only makes my code more organized and readable–it also allows me to efficiently adapt these functions in other projects to create different 3D scenes and effects and reuse the same complex objects from this scene.