Christopher Merced

CS 330 Final Project

Reflection Paper

Professor Philip Enkema

12/7/24

Overall, throughout the project I utilized a wide array of different types of objects to give my best rendition of the scene I’ve provided within the project. The scene itself is an office desk scene which is comprised of a wood top desk, a cup, a monitor, a mouse, a keyboard, a lamp, a phone and a multitude of books. This allowed me to be creative with the diversity of objects that were at my disposal. At the end I utilized the box shape in the base structure of the table as well as the wood top. The box shape was also utilized to provide a base structure to the monitor and was creatively utilized multiple times to make a book object for the scene. The box mesh was also utilized for the keyboard and all the keys that exist on it as well. The Plane mesh was utilized to set the scene with a wooden floor and a plaster wall to give the effect of a personal office room. The tapered cylinder shape was also utilized to replicate a lamp shade as well as the base of a cup. The half Torus shape was then utilized to replicate the handle of the coffee mug and then further reused to create the mouse wheel. A cylinder mesh was employed to create the base of the lamp as well as its legs and then further used to round out the edges of the monitor and gives the monitor’s base a more realistic look to it by granting curvature to the box objects that were used for the stand. The Cylinder also curved out the edges of the phone to better match a modern aesthetic.

You can navigate the 3D scene through both mouse and keyboard inputs, which control the virtual camera's orientation and position. Mouse movements adjust the camera's view by detecting the relative changes in cursor position within the window. The Mouse\_Position\_Callback() method calculates the horizontal (xOffset) and vertical (yOffset) changes between the current and last recorded mouse positions. These offsets are used to call the ProcessMouseMovement function, updating the camera's orientation. Keyboard inputs enable movement within the scene, allowing the camera to translate in various directions. The ProcessKeyboardEvents() method listens for key presses and responds based on the input. Pressing W or S moves the camera forward or backward, while A and D shift it left or right. Additional keys, such as Q and E, allow upward and downward movement. You can also toggle between orthographic and perspective projections by pressing O or P.

By the end of this project we’ve added and utilized SetupSceneLights(), DefineObjectMaterials(), LoadSceneTextures, and more. Each one of these custom functions helped to better organize the project so that the flow of the programs functionality remained clear and easy to follow through proper naming conventions and the utilization of comments to better inform the reader of the program tasks. Using two separate functions like PrepareScene() and RenderScene() allow us to directly distinguish between different phases of the program’s functionality which uplifts the ease of understanding from the reader’s perspective. We gain an intuitive notion of what these functions do just based on their naming conventions which significantly decreases the amount of visual clutter for operations that we may not yet need the full code base for. Another major bonus from these functions is their reusability, working on the keyboard object I created a LoadKeyboard function to invoke when we want to add the keyboard to the scene. This function also takes in three parameters for x, y and z positioning within the environment, allowing the user to spawn multiple keyboards wherever they desire. Within this keyboard function we utilize another custom function called GenerateKeyCap() which makes use of a double for loop and the previously passed parameters from LoadKeyboard() to generate sixty keys onto the keyboard without having to manually type in every single key generation. This style of functionality is also seen within the environment from the LoadBooks() function where a for loop is used to generate nine books onto the desk neatly next to each other with ease. Throughout the project you can see the reusability of functions like Draw<shape>Mesh() through the use of PrepareScene() where certain shapes are loaded into the program in order to be used later by calling our Draw<shape><Mesh() function cutting down greatly on repeat code and expediting development process. This exact same style of preparation with reusability is also seen in the use of DefineObjectMaterials() where textures are located and imported into an array to be mapped onto objects with simple tags later on via SetShaderTexture(“tag”) and even further modifications to the objects and their interactions with lighting can be made as well with the DefineObjectMaterials() function that can be utilized to instantiate different materials and cause objects to behave differently with light sources when assigned different materials via the SetShaderMaterial(“tag”) function as well. Overall this project is a great example of just how invaluable and helpful organization, clean code, and OOP practices are for all users and most importantly, future users.