Debynyhan Banks

7-1 Final Project

CS330 Computer Graphics and Visualizations

Southern New Hampshire University

Professor Bishop

December 10, 2023

My selection of objects in this 3D scene was influenced by practicality, simplicity, and my experience in 3D modeling with Blender. The way I constructed the scene was significantly influenced by my familiarity with Blender. Using basic geometric shapes as building blocks for more complex objects is one of Blender's principles. As a result of this experience, I was able to create complex objects using primitive shapes. Due to the pen's inherent cylindrical shape, a cylinder is used to model the pen. My cylindrical shape was created by using the CylinderParams struct. Due to its ring-like structure, the torus was selected for the pen holder. To represent the speaker, bookshelf, tablet, and room, I used a cube. Due to the inherent box-like structure of these objects, the cube was chosen. Cubes are fundamental shapes in 3D modeling, and I was able to modify them to represent different objects. The desk was modeled from a plane.

I used the Resource Acquistion Is Initialization (RAII) principle in my structs, such as GLMesh and TextureHandler to ensure efficient resource management. This approach is instrumental in handling the lifecycle of the various resources in my scene. For example, the GLMesh manages the vertex array, buffer, and element buffer objects for the 3D model. In the GLMesh constructor, OpenGL buffers are generated, and vertex data is uploaded. The destructor takes care of releasing these resources. This ensures that each mesh object autonomously manages its GPU resources. This helps reduce the risk of memory leaks or dangling pointers.

The TextureHandler struct manages the textures. Upon construction, a texture is loaded from the specified file, and OpenGL texture resources are allocated. The destructor handles the clean-up. This encapsulation simplifies resource management, and it allowed me to focus on the creative aspects without worrying about low-level details. The MouseHandler struct manages mouse input to control the camera’s view in the 3D scene. It plays a vital role in handling user input and making the scene interactive. The Vertex struct represents a single vertex position, texture coordinates, and normals.

Navigating the 3D scene is managed by the “processInput” function, which uses the GLFW’s API for reading input. Its key and mouse callback functions are used to detect keyboard and mouse inputs to control the camera’s position, orientation and speed of camera movements. The keys W, A, S, D, Q, E were assigned to forward, left, backward, right, up, and down movements, respectively. The speed of the movement is controlled by the mouse scroll wheel. By scrolling up or down on the mouse scroll wheel the speed of the camera’s movement will increase or decrease, respectively. The MouseHandler struct updates the camera’s yaw and pitch angles based on the movement of the user’s cursor. This allows the user to look around the scene. This is achieved by capturing the mouse’s position changes and translating into camera rotation.

In this OpenGL program, I have employed the functional programming paradigm and adhered to the principle of single responsibility and the RAII idiom to enhance the modularity and organization of the code. To embody these principles, I have created several custom functions and structs. I also created utility functions for window management, error checking, input handling, rendering, and scene setup. These functions enhance the modularity of my code by abstracting away complex operations.

Each of these elements is tasked with a distinct responsibility. This ensures that every component deals with a specific aspect of the program. This approach enhances the usability and extensibility of the code. By managing OpenGL mesh data in a struct and encapsulating the setup for VAOs, VBOs, and EBOs, this structure aligns with the single responsibility principle.

References:

Janssen, T. (2023, May 1). *SOLID design principles explained: the single responsibility principle*. Stackify. https://stackify.com/solid-design-principles/