**CS330: Final Project**

**Justify development choices for your 3D scene. As you write, think about why you chose your selected objects. Also consider how you were able to program for the required functionality.**

I decided to create a 3D scene showcasing five of my favorite video games and consoles. The scene includes a PlayStation 1 console, a Nintendo Gameboy, a Spiderman PlayStation 2 disc, a Red Alert PlayStation game case, a Donkey Kong Land Gameboy cartridge, and a floor. I chose these objects for a couple of reasons. First, I personally find them cool, so I thought it would be more enjoyable to work on a project where I'm recreating objects I love. Second, I picked these items because they provided just the right level of complexity and challenge for the scope of my project. I designed the scene with a mix of complexity, featuring the intricate PlayStation 1 console, and simplicity, using basic shapes for the other objects. To make all this happen, I used OpenGL, which is an open-source graphics API (application programming interface) that provides a set of functions and commands for rendering 2D and 3D graphics. With OpenGL, I could create 3D objects using its built-in draw command along with a custom function I created for the different primitive shapes. Plus, I could move these objects around by telling OpenGL to transform the objects to change their positions and orientations. To make things look more life-like, I added textures and shading to my objects, giving the whole scene a lifelike appearance.

**Explain how a user can navigate your 3D scene. As you compose your thoughts, discuss how you set up to control the virtual camera for your 3D scene using different input devices.**

Users can seamlessly navigate within my scene using a combination of mouse and keyboard inputs. Facilitating movement, the virtual camera possesses fundamental controls, enabling users to effortlessly glide forward, backward, left, right, upwards, and downwards. These maneuvers are effortlessly executed by pressing specific keys on the keyboard. The experience is further enhanced by the camera's ability to dynamically scan the surroundings as users swivel the mouse. An example of this would be the user shifting the mouse to the left which would then prompt the virtual camera to smoothly pivot in the same direction. To expand the range of motion, I incorporated a zoom function through the scroll wheel. The foundation of this camera control setup lies in the "camera.h" file sourced from "LearnOpenGL.com", which inherently supplies a significant chunk of the required camera functionalities. In combination with the gCamera class, updates were necessary within my "Source.cpp" file to perpetually monitor key bind presses. I utilized the UProcessInput, key\_callback functions, UMousePositionCallback, and UMouseScrollCallback for this purpose. The UMousePositionCallback function steered camera orientation, relayed seamlessly to the virtual camera as it was called every frame to check for new updates. In addition, the UMouseScrollCallback function gauges zooming requisites based on scroll wheel activity. With the combination of these functions, I seamlessly passed them into the gCamera class as new inputs have been made. Refer to the accompanying table for an overview of key binds and their corresponding actions within the program.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **W:** Forward | **S:** Backward | **A:** Left | **D:** Right | **Scroll Fwd:** Zoom In |
| **Q:** Down | **E:** Up | **Esc:** Quit | **P:** Toggle View | **Scroll Back:** Zoom Out |

**Explain the custom functions in your program that you are using to make your code more modular and organized. Ask yourself, what does the function you developed do and how is it reusable?**

I implemented a function named UCreateCube that can create a primitive cube object with all the necessary vertices, normal and texture coordinates. In my scene I had multiple cubes meshes that I needed to create, so having this function allowed me to keep my code more modular and organized by not having to keep rewriting redundant code. I also created custom functions for the other primitive shape types that I used within my scene. These functions include the UCreatePlaneMesh and UCreateCylinderMesh. Although these shape types were only used once in my program, having the ability to call these functions, when necessary, in a separate portion of the code kept things cleaner and simpler to read. The alternative would have been to create all the objects within the Main() function, and this would ultimately make the Main() function very cluttered and hard to read. With my method of making my code more modular, I believe that it makes it significantly easier to go back and make changes or add additional code if needed.