**Brooklen Ashleigh**

**CS330**

**Professor Bishop**

**June 23rd, 2024**

**Project Reflection**

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

3D scenes are often represented by approximate, simple objects. High detail and fidelity in 3D modeling mean a larger number of shape vertices and faces that must be calculated and rendered, therefore increasing the overall resource cost to render. Approximating objects with simple geometry with apt texturing and lighting can create a decent representation of objects for many purposes. This is especially pertinent in video games, where rendering complex objects on top of logic can become very resource intensive very quickly, necessitating the creation of techniques that increase performance.

For my scene, I chose to replicate a photo I took that already had a somewhat geometric appearance; I thought my rendering would look more accurate and fitting if I chose something that already fit that aesthetic.

Though I knew it would be challenging, I was able to produce the odd aquarium roof, one of the dominant details of the photo, using prisms. My texturing isn’t perfect, but without UV unwrapping, I think it fits the grid texture and shape well. Aligning the texture on the non-symmetrical shape was tricky and required some compensation, especially because the roof is composed of multiple differently scaled pieces. I believe I have a better understanding of texture mapping because of the difficulties I faced.

Because the yellow bar accent on the top middle of the building used the same shapes multiple times, I was able to reduce the amount of extra code and parameter changes by only calling the methods that changed the parameters I needed to. The downside to this would be if the code needed to be rearranged later. The rest of the building was easy to approximate with squares.

One of the code features that helped me the most was using relative positioning for the duplicate posts in the walk-up to the building. I used an array to coordinate the positioning of and group the component objects together, which made it very easy to pattern them across the entrance with an offset.

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

The program has been setup to respond to both keyboard and mouse input, which allows for fluid movement through the scene. The WASD keys control the X and Y positioning of the camera, while Q and E control its Z-height. The mouse input controls the angles of the camera, allowing the user to look fully around the scene. The mouse wheel increases and decreases the camera’s movement speed, allowing the user to make small or large adjustments. Lastly, the P and O keys change the type of camera lens used in the scene. They also reset the camera to its original position, just in case one gets lost exploring the scene. With every program loop, input from these devices are checked and callback functions make minute changes to the positioning of the camera to allow smooth control. These inputs, when used together, allow nuanced travel across the scene, and were instrumental in building it, as being able to see 3D objects from various angles is important to being able to tell how the objects are interacting in 3D space.

**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?

This program uses many methods to simplify repetitive operations as shapes are rendered and configured within the program. SetTransformations(), as an example, does a large amount of repetitive work every time an object’s parameters need to be changed or set before it is drawn. Passing the transform, rotation, etc. vectors to it allows the reuse of integral code in a modular way without having to type the same property declarations over and over.

I didn’t write any custom additional functions for this project, as I didn’t think they were necessary. If it needed to become more complex, however, I would have created an abstracted method for creating multiple of the same objects at once. I can imagine a helper function that would take the placement of multiple objects and draw them together, which could help organize the code for adding more objects. A function call like drawPattern(object, stepX, stepY) would have been helpful for creating lines of lamp posts as I did for my scene. Overall, the code could be improved or modularized to make the RenderScene function shorter or not as busy, especially if there are multiple types of the same object.