7-1 Final Project Submission

Natalia A. Santiago

Southern New Hampshire University

CS-330: Comp Graphic and Visualization

Gholam Shaykhian

October 20, 2024

7-1 Final Project Submission

I have chosen a minimalist nine-drawer dresser, two table lamps, framed artwork, a decorative bowl, and a rectangular box. All these will create a serene symmetrical atmosphere when combined. Since it is a minimalist design, the visuals are neat and uncluttered, with much attention given to the interplay between light and texture.

In the implementation of class creation for each object using the principles of OOP, instantiation and manipulation of objects have become easier. Each class should have methods defining the properties such as position, size, and texture, behaviors such as interactions and animations for a class. The rendering engine was then responsible for drawing and updating these to make them appear and behave as intended.

Keyboard input-tracking can control it with WASD keys for movement and arrow keys to look around, and by mouse input in the form of dragging to look around. This setup offers intuitive and flexible navigation; hence users can explore the scene fully.

It does this using event listeners for keyboard and mouse inputs to move the virtual camera. For instance, it can move forward with key W pressed or by dragging the mouse to rotate the camera view. This is done through real-time updating of position and rotation matrices for the camera in order to make smooth and responsive navigation.

So, keeping in mind the modularity of the code and organizing it properly, I developed several functions to deal with custom tasks. It has functions to initialize objects, update states, and render. Each of the functions does a certain job and makes the whole code more maintainable and reusable.

Examples:

def initialize\_object(name, position, size, texture):

obj = Object(name, position, size, texture)

scene.add(obj)

return obj

This function initializes an object and adds it into the scene and is reusable for any object type.

def update\_object(obj, delta\_time):

obj.position += obj.velocity \* delta\_time

obj.update()

This function updated the object’s position according to its velocity and the time elapsed to make animations smooth.

def render\_scene(scene, camera):

for obj in scene.objects:

obj.render(camera)

This function would render all the objects of the scene from the camera's point of view, encapsulating the rendering logic.

Thus, these functions put together little self-contained pieces of work, making code modular, easy to handle, and reusable in several projects.

In short, my approach to the 3D scene was based on a balanced and eye-catching environment, intuitive user navigation, and modular and organized code. This combination ensures that there would be a qualitative and maintainable 3D experience.