**7-1 Final Project Reflection**

Jose Garza

Computer Science Department, Southern New Hampshire University

CS-330: Computational Graphics and Visualization

Professor Dr. Brian Holbert

December 16, 2024

**7-1 Final Project Reflection**

**Justify development choices for your 3D scene:**

The following is a photo of the scene I rendered in OpenGL:  
A black round table with a black object on it

Description automatically generated

I chose those objects as they all had shapes I found interesting. Moreover, I was especially curious about how to make the glasses case as it was a triangular prism and that shape was not readily available in our collection. In any case, I felt that many shapes here were covered, lighting was good, and the lamp was a great subject as a complex object.

From a requirements perspective, my scene would include the box, half sphere, cylinder, plane, and tapered cylinder from the basic mesh shapes included in the project. I also added a new shape in ShapeMeshes.h/cpp and defined the triangular prism by mirroring existing shape definitions from the Load methods and some examples online. This is then implement with the rest of the shapes in PrepareScene:  
  
void SceneManager::PrepareScene()

{

// load the texture image files for the textures applied

// to objects in the 3D scene

LoadSceneTextures();

// define the materials that will be used for the objects

// in the 3D scene

DefineObjectMaterials();

// add and defile the light sources for the 3D scene

SetupSceneLights();

// only one instance of a particular mesh needs to be

// loaded in memory no matter how many times it is drawn

// in the rendered 3D scene

m\_basicMeshes->LoadSphereMesh();

m\_basicMeshes->LoadTaperedCylinderMesh();

m\_basicMeshes->LoadCylinderMesh();

m\_basicMeshes->LoadPlaneMesh();

m\_basicMeshes->LoadBoxMesh();

// Load the mesh for the glasses case

m\_basicMeshes->LoadTriangularPrismMesh();

Textures were added to the floor and wall planes to look like the image, as well as the lamp shade. Materials were defined to ensure the appropriate qualities and lighting were applied to the objects. There were several light sources, and I applied feedback to diffuse the light further from the past assignment. All three lighting components were used: ambient, diffuse, and specular. Finally, the objects were placed relative to the scene from the photo.

**Explain how a user can navigate your 3D scene:**

Navigation is possible using both the mouse and keyboard. WASD keys enable the user to navigate forward, backward, left and right. The Q and E keys control moving up and down. Mouse cursor can look in any direction, and the scroll wheel controls the speed of movement. Finally, you can toggle perspective and orthographic displays of the scene using the P and O keys respectively. You can see, for example, how these views are implemented in the ProcessKeyboardEvents method in ViewManager:

// change between different projection views

if (glfwGetKey(m\_pWindow, GLFW\_KEY\_O) == GLFW\_PRESS)

{

// change to a multi-view orthographic projection

bOrthographicProjection = true;

// change the camera settings to show a front orthographic view

g\_pCamera->Position = glm::vec3(0.0f, 4.0f, 10.0f);

g\_pCamera->Up = glm::vec3(0.0f, 1.0f, 0.0f);

g\_pCamera->Front = glm::vec3(0.0f, 0.0f, -1.0f);

}

if (glfwGetKey(m\_pWindow, GLFW\_KEY\_P) == GLFW\_PRESS)

{

// change to perspective projection

bOrthographicProjection = false;

// change the camera settings to show a perspective view

g\_pCamera->Position = glm::vec3(0.0f, 5.5f, 8.0f);

g\_pCamera->Front = glm::vec3(0.0f, -0.5f, -2.0f);

g\_pCamera->Up = glm::vec3(0.0f, 1.0f, 0.0f);

g\_pCamera->Zoom = 80;

}

**Explain the custom functions in your program that you are using to make your code more modular and organized:**

Following best practices, I created custom functions to represent each object in the 3D scene, including the table, floor, backdrop, lamp, book, air freshener, and glasses case. Each of these methods are combined with the RenderScene() method to consolidate code and make things easy to read. Furthermore, comments are included in all methods and edits to ensure best practices and readability. Formatting, indentation, and spacing also align with best practices to keep each code block organized.

The following is an example of the simplification of the RenderScene() and modular use of object rendering methods:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* RenderScene()

\*

\* This method is used for rendering the 3D scene by

\* transforming and drawing the basic 3D shapes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void SceneManager::RenderScene()

{

RenderAirFreshner();

RenderBackdrop();

RenderBook();

RenderFloor();

RenderGlassesCase();

RenderLamp();

RenderTable();

}