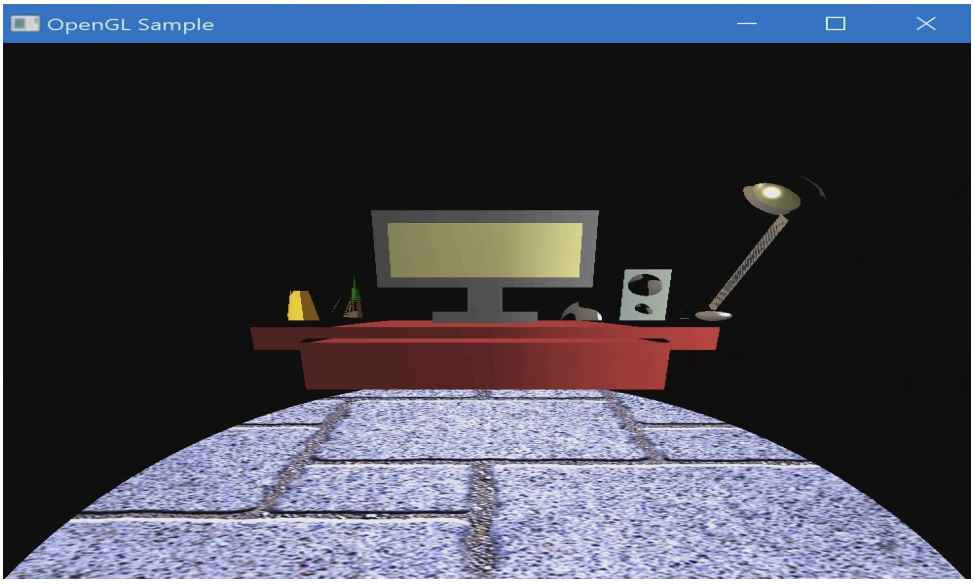
Ihab Elrayah

10/15/24

CS-330 7-1 Final Project Submission

3D Final Project: My Desk Setup

For my 3D final project, I choose to recreate an image of my desk setup at home. This project required the use of multiple shapes to accurately model the various elements of my workspace. By combining and overlapping different geometric forms, I created low-polygon 3D representations of real-world objects. In total, I utilized 15 shapes to construct 8 distinct items within my scene.



Object Breakdown

- **Floor**:
    - I used a **Plane** object for the floor and applied a texture that closely matches the appearance of my actual flooring.

- **Desk**:
    - The desk is divided into two components: the **Desk Frame** and **Desk Base**, both constructed from **Box** objects. I adjusted their sizes to closely resemble my desk's real-life proportions.
- **Computer**:
    - The computer is modeled using four separate objects: the **Computer Frame**, **Screen**, **Neck**, and **Base**. Each of these components was created using **Box** objects of various dimensions to form squares and rectangles. This design serves as the centerpiece of the scene, with careful attention to measurements. The screen overlaps the frame to achieve a realistic look.
- **Computer Mouse**:
    - I used a **Sphere** object for the mouse, carefuly scaling it to match the oval shape of an actual mouse. The sphere was rotated 180-degree to lay flat on the desk, ensuring accurate proportions.
- **Speakers**:
    - The speakers were modeled with three components: the **Speaker Base**, **Volume**, and **Button**. I utilized a **Box** object or the base, adjusting the size and rotation to a 90-degree angle to match the real structure. The dimensions were based on measurements from my actual speakers getting the correct separation length of the volume and the button both used with the **Sphere** object.
- **Plant**:
    - I divided the plant into two objects: a **Tapered Cylinder** for the pot and a **Cone** for the plant itself. I carefully calculated the measurements to ensure that the pot

sits correctly on the table and that the plant aligns properly on top of the pot, effectively representing their shapes and appearances.

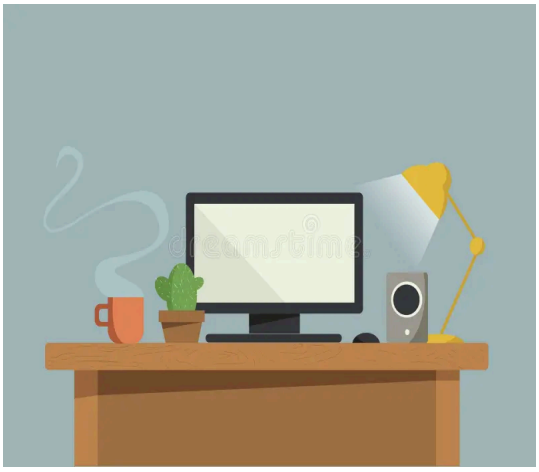
- **Coffee**:
    - The coffee setup consists of two parts: a **Cylinder** for the coffee mug and a **Torus** for the handle. I overlapped the mug over the handle to create the illusion of a cohesive coffee mug, effectively splitting the torus to accurately represent the handle's shape.
- **Lamp**:
    - The lamp is composed of four objects: the **Lamp Base**, **Neck**, **Head**, and a **Lightbulb**. I carefully adjusted the length, width, and height of each object to best showcase the structure of the lamp. A **Sphere** was used for the base, a **Box** for the neck, a **Cylinder** for the head, and another **Sphere** for the lightbulb, ensuring that each component harmoniously fits together.

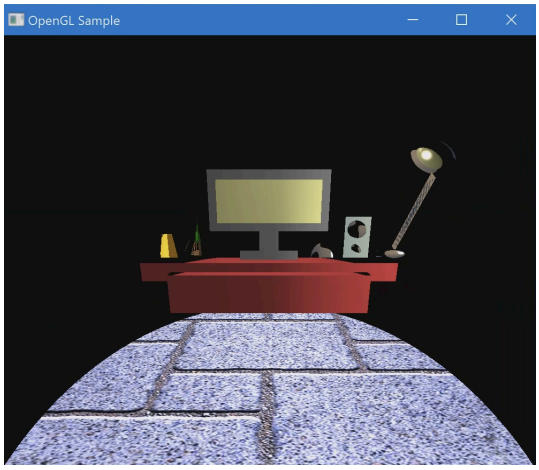
In developing my 3D scene, I engaged in continuous testing and integration of results to ensure that the dimensions of each object accurately matched their real-world counterparts. Each test provided valuable insights, allowing me to refine my approach and implement my code more effectively.

I carefully selected geometric shapes based on their ability to represent real-world objects. For example, I choose Cylinders for rounded items like the coffee mug and Boxes for more angular forms like the desk and computer components. This choice simplified the modeling process while maintaining visual accuracy.

In programming the functionality, I ensured that each object's scale, rotation, and position were precisely calculated. This attention to detail enabled the seamless interaction of objects within the scene, creating a realistic and cohesive environment. Through iterative testing and adjustments, I was able to optimize the placement and proportions, enhancing the overall quality of the 3D representation.

This project not only allowed me to practice my modeling skills but also helped me gain a deeper understandings of how to translate real-world objects into a 3D space. By carefuly selecting and manipulating shapes, I successfully created a visually appealing and accurate representation of my desk setup.



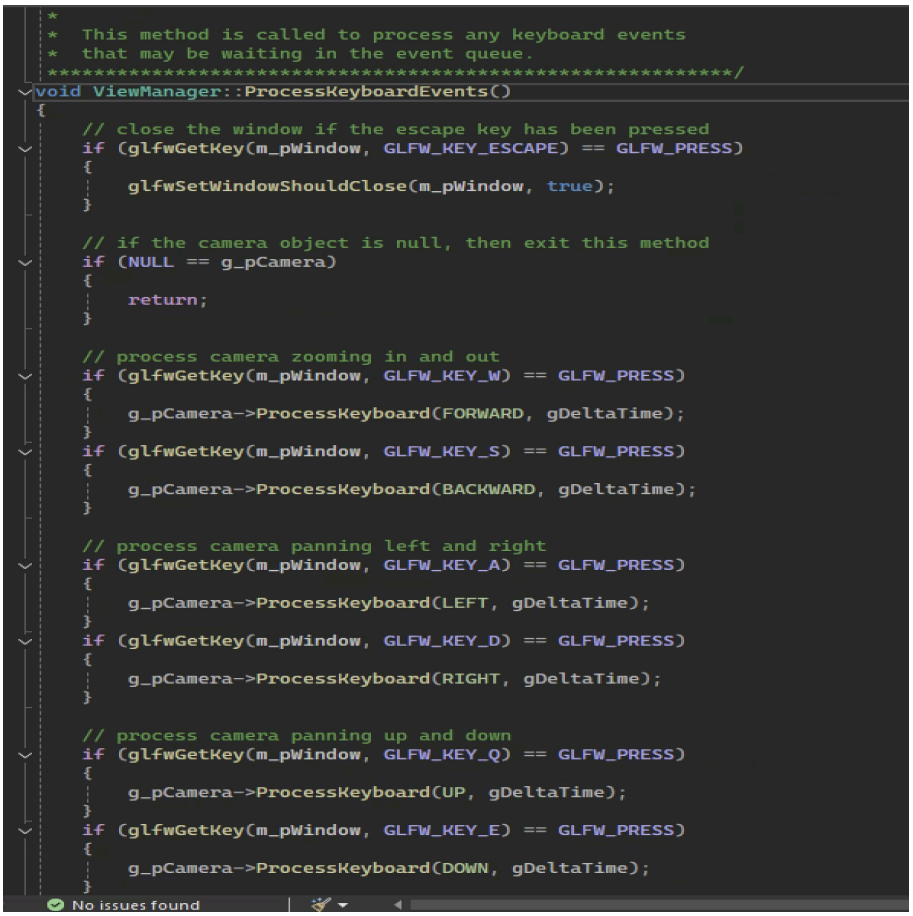


User Navigation in the 3D Scene

Users can navigate the 3D scene using a combination of keyboard and mouse controls, creating an intuitive and immersive experience. The navigation system is specifically designed to facilitate panning, zooming, and orbiting within the environment, allowing for flexible exploration of the virtual space.

The keyboard controls include the **WASD** keys, which enable straightforward movement. Pressing **W** moves the camera forward (zooming in), while **S** moves it backward (zooming out). The **A** and **D** keys allow for panning left and right, respectively. Additionally, the **Q** and **E** keys control vertical movement, with **Q** moving the camera upward and **E** moving it downward. This setup provides users with a comprehensive range of motion within the scene.

Mouse controls further enhance navigation capabilities. Users can change the orientation of the camera by moving the mouse, enabling them to look up, down, left, or right with ease. The mouse scroll wheel is also utilized to adjust the speed of camera movement, alowing users to travel around the scene at their preferred pace. Together, these controls create a seamless and engaging navigation experience in the 3D environment.



```
*
*  This method is called to process any keyboard events
*  that may be waiting in the event queue.
***********************************************************/
void ViewManager::ProcessKeyboardEvents()
{
    // close the window if the escape key has been pressed
    if (glfwGetKey(m_pWindow, GLFW_KEY_ESCAPE) == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(m_pWindow, true);
    }

    // if the camera object is null, then exit this method
    if (NULL == g_pCamera)
    {
        return;
    }

    // process camera zooming in and out
    if (glfwGetKey(m_pWindow, GLFW_KEY_W) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(FORWARD, gDeltaTime);
    }
    if (glfwGetKey(m_pWindow, GLFW_KEY_S) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(BACKWARD, gDeltaTime);
    }

    // process camera panning left and right
    if (glfwGetKey(m_pWindow, GLFW_KEY_A) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(LEFT, gDeltaTime);
    }
    if (glfwGetKey(m_pWindow, GLFW_KEY_D) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(RIGHT, gDeltaTime);
    }

    // process camera panning up and down
    if (glfwGetKey(m_pWindow, GLFW_KEY_Q) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(UP, gDeltaTime);
    }
    if (glfwGetKey(m_pWindow, GLFW_KEY_E) == GLFW_PRESS)
    {
        g_pCamera->ProcessKeyboard(DOWN, gDeltaTime);
    }
```

Enhancing Modularity and Organization through Custom Functions

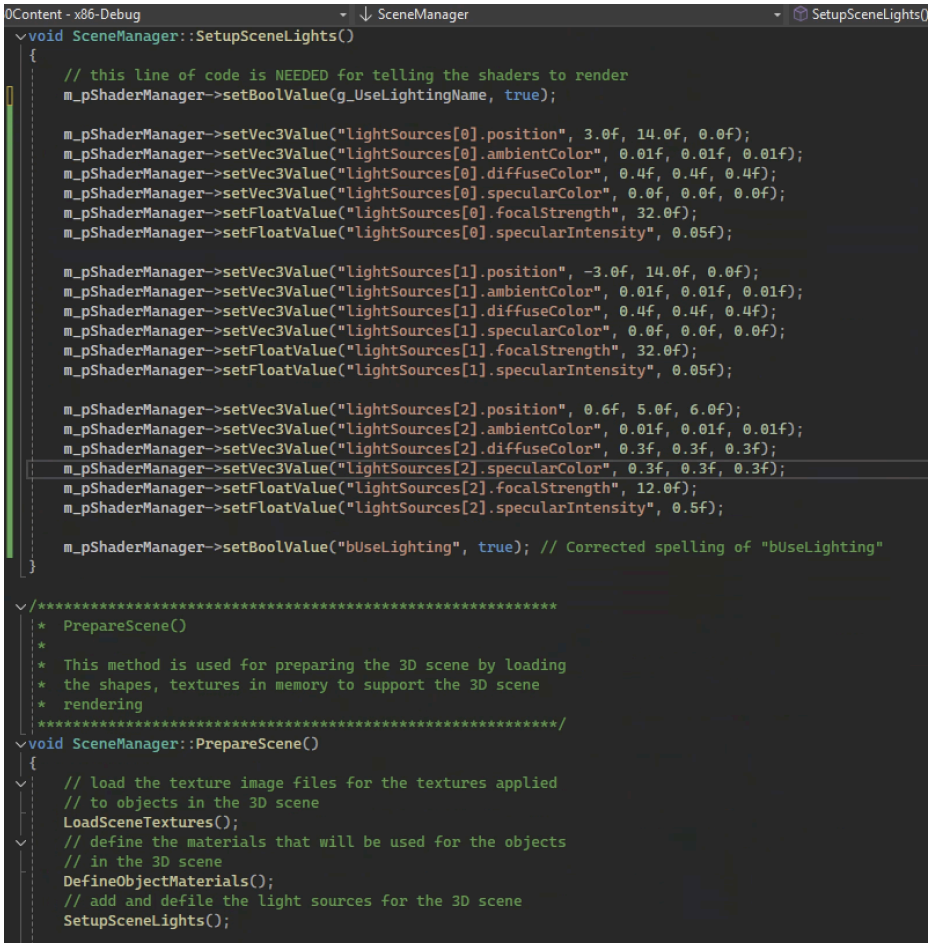
In my program, I implemented several custom functions to enhance modularity and organization, adhering to clean code principles and best practices. This approach not only makes the code easier to maintain but also ensures that it is readable and understandable for anyone who might follow along.

One of the key functions is “ProcessKeyboard()”, which manages all keyboard input for camera control. This function takes parameters like direction and delta time to calculate camera position updates based on user input. By centralizing keyboard handling in this function, I can easily modify or extend camera controls without changing the main event loop. This modularity makes it reusable in different contexts, facilitating future enhancements and ensuring that the code remains clean and efficient.

Another important function is “SetTransformations()”, which applies transformations, such as scaling, rotation, and translation, to 3D objects. This function takes parameters for each transformation type and updates the object's transformation matrix accordingly. Utilizing this function across various objects promotes consistency and allows for easy adjustments to how objects are rendered, all while keeping the code organized and avoiding redundancy.

I also implemented “SetShaderColor()”, which simplifies the process of setting color values in shaders. By calling this function with RGBA values, I can quickly change the appearance of multiple objects with minimal code, enhancing maintainabillity and reducing errors.

Lighting and Scene Preparation In my 3D project, I used custom functions to enhance organization and modularity. The PrepareScene() function loads shapes and textures, while DefineObjectMaterials() sets properties like ambient and diffuse colors. The SetupSceneLights() configures light sources to create realistic illumination. Lastly, RenderScene() handle object rendering with transformations. By maintaining clean code with clear comments, I ensured that the project is easy to read and update.



In addition to functionality, I focused on adding comments throughout the code to explain each function's purpose and its parameters. This practice helps with understanding the code's structure

but also makes it easier for others to follow along. Adhering to clean code principles, such as meaningful naming conventions and consistent formatting, ensures that the code remains organized and accessible.

Overall, these custom functions and practices contribute to a clean, organized codebase that is easy to read and maintain, benefiting both current and future development efforts.

```
/***************************************************************/

// Speaker 11

// set the XYZ scale for the mesh
scaleXYZ = glm::vec3(0.0f, 3.0f, 1.5f);

// set the XYZ rotation for the mesh
XrotationDegrees = 0.0f;
YrotationDegrees = 90.0f;
ZrotationDegrees = 0.0f;

// set the XYZ position for the mesh
positionXYZ = glm::vec3(5.2f, 4.0f, 0.0f);

// set the transformations into memory to be used on the drawn meshes
SetTransformations(
    scaleXYZ,
    XrotationDegrees,
    YrotationDegrees,
    ZrotationDegrees,
    positionXYZ);

// set the color values into the shader
SetShaderColor(0.68f, 0.85f, 0.9f, 1.0f); // Light sky blue

// draw the mesh with transformation values
m_basicMeshes->DrawBoxMesh();

//Speaker volume circle

scaleXYZ = glm::vec3(1.0f, 0.5f, 0.5f); // Different shape
positionXYZ = glm::vec3(5.1f, 4.8f, 0.0f); // Adjusted position for overlap
SetTransformations(scaleXYZ, XrotationDegrees, YrotationDegrees, ZrotationDegrees, positionXYZ);
SetShaderColor(0.5, 0.5, 0.5, 1.0);
m_basicMeshes->DrawSphereMesh();

//Speaker volume button
scaleXYZ = glm::vec3(1.0f, 0.2f, 0.2f); // Different shape
positionXYZ = glm::vec3(5.1f, 3.7f, 0.0f); // Adjusted position for overlap
SetTransformations(scaleXYZ, XrotationDegrees, YrotationDegrees, ZrotationDegrees, positionXYZ);
SetShaderColor(0.5, 0.5, 0.5, 1.0);
m_basicMeshes->DrawSphereMesh();
/***************************************************************/
```