**7-1 Final Project**

David J Allen

CS330: Comp Graphic and Visualization

Southern New Hampshire University

Professor Rodriguez

April 16, 2023

**7-1 Final Project**

**Development Choices**

When comparatively looking over the elements that could be conveyed three-dimensionally, it made sense to not overcomplicate the scene by making it too busy. Also, per the instructions given, the inclusion of two primitive shapes combined to form an object was necessary. The first object I began developing immediately was the cup. This object is created using a tall hollow cylinder – as the simplest shape I chose to do this one first. In my picture this item was plain white, but I chose to texture so that it wouldn’t be drowned out in the scene. The next simple object I chose to build was the stack of paper. I was under the impression this was going to be an easy task but as it relied on a stretched-out cube it proved to be slightly more complicated. I decided to implement a darker color to simulate the stack underneath, so it was not just a small, stretched cube of white and gave more of an impression of a 3d object. Another simple object I chose to build was a standard cube. In the chosen picture the square was a charge block that I had but as it was only flat black, I decided to increase the size slightly and texture it like a miniature computer tower. The plane that was created was the second most difficult object due to its juxtaposition in accordance with the objects that needed to sit on top of it. The plane was set up and textured to represent the carpeting from the original picture. The complexity of the texture was hard to perfect and seemed to be tough to render so I appropriated it as a blue texture. The most difficult of the objects was the pen, because of the creation process necessary to model it in 3D. The pen was created using a combination of cylinders and a cone with texturing. The cylinders were used for the end of the pen, the body of the pen, and the clip. A cone was used for the end to simulate the tip of the pen. I chose to texture this object as well to match the Wal-Mart one from the original picture. I chose these shapes and textures to bring realism to the crafted scene to resemble as closely as possible the captured 2D image. In the design process another important aspect to adhere to requirements was switching the view from perspective to ortho to simulate 3D versus 2D, this is done with a key press of “V” for orthographic and “B” for perspective.

// Switches view from perspective to ortho "V" and "B"

if (glfwGetKey(window, GLFW\_KEY\_V) == GLFW\_PRESS)

perspective = false;

if (glfwGetKey(window, GLFW\_KEY\_B) == G

**User Navigation**

Camera navigation relied on programming the acceptance of input keys with corresponding motion on a keyboard. If the user holds “W” the camera will move forward – toward the scene. If the user used “S” instead, the camera would move backwards towards the user. The keys “A” and “D” correlate to movement left and right. The last two keys to manage camera movement are “E” and “Q” which move up and down.

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS)

gCamera.ProcessKeyboard(FORWARD, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS)

gCamera.ProcessKeyboard(BACKWARD, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS)

gCamera.ProcessKeyboard(LEFT, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS)

gCamera.ProcessKeyboard(RIGHT, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_E) == GLFW\_PRESS)

gCamera.ProcessKeyboard(UP, gDeltaTime);

if (glfwGetKey(window, GLFW\_KEY\_Q) == GLFW\_PRESS)

gCamera.ProcessKeyboard(DOWN, gDeltaTime);

The user can also navigate the scene using the mouse buttons and scroll. As can be seen in the instructions, the mouse cursor is programmed to change the orientation of the camera while the mouse scroll adjusts the speed of the camera. This is not the only navigational method in the scene, the light can also be moved. The registered key presses for ligjt movement are “J, L, I, K, U, and O”, all of which shift movement by 0.05 with each press.

if (glfwGetKey(window, GLFW\_KEY\_J) == GLFW\_PRESS)

gSpotLightPosition.x -= 0.05f;

if (glfwGetKey(window, GLFW\_KEY\_L) == GLFW\_PRESS)

gSpotLightPosition.x += 0.05f;

if (glfwGetKey(window, GLFW\_KEY\_I) == GLFW\_PRESS)

gSpotLightPosition.z -= 0.05f;

if (glfwGetKey(window, GLFW\_KEY\_K) == GLFW\_PRESS)

gSpotLightPosition.z += 0.05f;

if (glfwGetKey(window, GLFW\_KEY\_U) == GLFW\_PRESS)

gSpotLightPosition.y -= 0.05f;

if (glfwGetKey(window, GLFW\_KEY\_O) == GLFW\_PRESS)

gSpotLightPosition.y += 0.05f;

The light while beginning orbiting can also be stopped using a key press and reinitiated with another press. The left alt button causes the light to orbit while the right alt key stops the orbiting.

**Custom Functions**

For custom functions, I created if loops to be able to change the color of the lights and be able to turn them on and off. This was done using red, green, and blue in corroboration with key presses. This allows multiple possibilities of colored lights depending on the scene or perhaps the user may prefer to not have the light on to see how it changes the scene. There are three scene lightings based on if the user presses “1, 2, or 3” and then the left bracket and right bracket keys corresponding to if the light is on.

// Modifies color of light

if (glfwGetKey(window, GLFW\_KEY\_1) == GLFW\_PRESS)

{

gKeyLightColor.r += 0.001f;

if (gKeyLightColor.r > 1.0f)

{

gKeyLightColor.r = 0.0f;

}

}

if (glfwGetKey(window, GLFW\_KEY\_2) == GLFW\_PRESS)

{

gKeyLightColor.g += 0.001f;

if (gKeyLightColor.g > 1.0f)

{

gKeyLightColor.g = 0.0f;

}

}

if (glfwGetKey(window, GLFW\_KEY\_3) == GLFW\_PRESS)

{

gKeyLightColor.b += 0.001f;

if (gKeyLightColor.b > 1.0f)

{

gKeyLightColor.b = 0.0f;

}

}

// Turns off light "["

if (glfwGetKey(window, GLFW\_KEY\_LEFT\_BRACKET) == GLFW\_PRESS)

{

gKeyLightColor.r = 0.0f;

gKeyLightColor.g = 0.0f;

gKeyLightColor.b = 0.0f;

}

// Turns off key light "]"

if (glfwGetKey(window, GLFW\_KEY\_RIGHT\_BRACKET) == GLFW\_PRESS)

{

gKeyLightColor.r = 1.0f;

gKeyLightColor.g = 0.0f;

gKeyLightColor.b = 1.0f;

}

Another modular aspect that has to do with the light that was referenced before was the orbiting light controls. Utilizing key presses can change if the light is or is not orbiting the scene which can change how it affects the lighting.

static bool isLKeyDown = false;

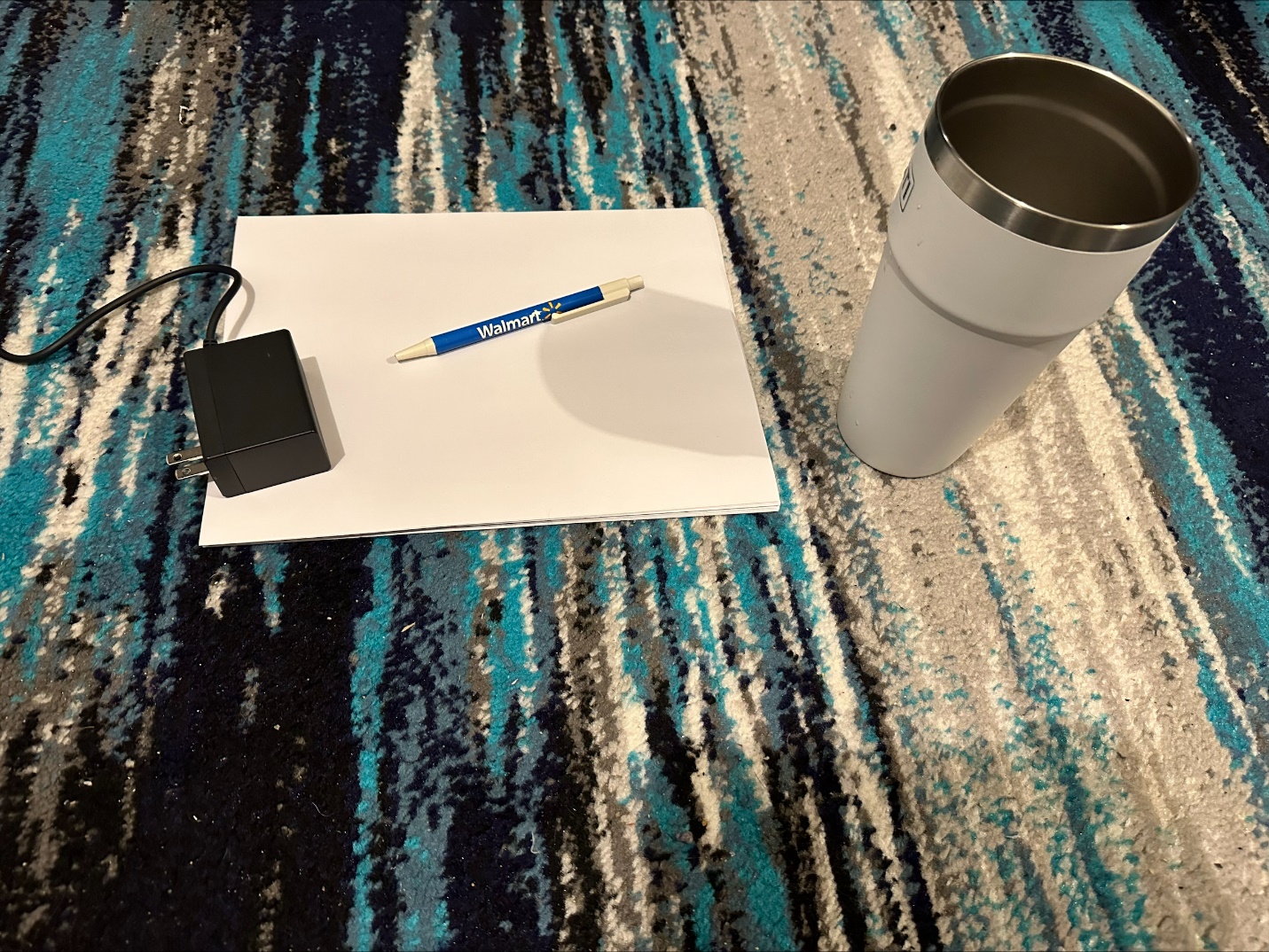
if (glfwGetKey(window, GLFW\_KEY\_LEFT\_ALT) == GLFW\_PRESS && !gSpotLightOrbit)

gSpotLightOrbit = true;

else if (glfwGetKey(window, GLFW\_KEY\_RIGHT\_ALT) == GLFW\_PRESS && gSpotLightOrbit)

gSpotLightOrbit = false;

**Original**

****

**Mockup Drawing**

**Diagram

Description automatically generated**

**3D Scene**

