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

For this project I decided to create a scene that had a milk carton, a glass pyramid, and a crystal ball. I decided on these objects because I figured that they wouldn’t be overwhelmingly challenging to recreate. The course work did not come easy to me this semester, but I am proud of the progress I’ve made with OpenGL. Over the course if this term I ran into many roadblocks when developing this project. The first main roadblock was in the setup of all the project libraries and linkers. I hadn’t had too much experience with Microsoft Visual Studio which led to many simple errors. I watched the program evolved each day to meet the next weeks requirements. My biggest obstacle in this course was getting the lighting to interact with the objects set textures. At first I used the fragment and vertex shader files “6.light\_cube” and “6.multiple\_lights”. After spending a week trying to get the textures to render properly I switched to the “5.4.light\_casters” and “5.4.light\_cube.” After setting all the light properties and math for distance the textures appeared almost perfectly.

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

The controls for this project are commented in “Source.cpp” in lines 1-24. The camera along the X and Z-axes can be moved with the standard ‘WASD’ keys as well as ‘Q’ and ‘E’ for vertical movement along the Y-axis. I have also added controls so that the user can control and move around the cube light. The Y-axis is controlled with ‘Y’ and ‘H’, the X-axis is controlled with G and ‘J’, and the Z-axis is controlled with ‘M’ and ‘N’. The key ‘O’ is used to turn on “ortho-mode” which gives a more 2D appearance to the rendered scene. The key ‘L’ is used to activate “orbit-mode” where the camera begins to orbit around the 3D scene. Inside the render loop there’s a series of if and if-else statements that determine whether the user has activated these perspective changes. When they’re activated, the view and projection matrix are changed to the corresponding modes. I also edited the scroll\_callback function to increase or decrease the camera’s speed depending on which direction the scroll goes.

1. **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?

I didn’t have the amount of time I wanted to make this program completely modular however I was able to add a couple custom functions that helped with the changing of camera modes. The first function that I implemented was the loadTexture() function that takes in a texture location as a char pointer. The texture at that location is then loaded and set to wherever that method is called. This helped with making the code much easier to read. Instead of each of the 8 textures needing 15-20 lines each to be formatted and loaded properly, the loadTexture() function can be called and set to an unsigned int. The function resetCamera() is called when the user presses the ‘F’ key. When resetCamera() is called, all of the camera’s vectors are reset to the values that resetCamera holds. I chose to make the camera spawn about 5.0 float units out from the (0, 0, 0) orgin. The function resetCamera is called by another custom function called TransformCamera(). TransformCamera() is called after the glfwSwapBuffers() and essentially it checks to see if ‘O’ ‘L’ and ‘F’ were pressed. If ‘O’ or ‘L’ are pressed then the corresponding Boolean is changed, which then changes the view and projection matrices. If ‘F’ is pressed, resetCamera() is called and the mode Booleans are all set to false. I made small changes and tweaks in almost every single function of the program throughout the course. I replaced all of the code from the scroll\_callback() and allowed the mouse wheel control the cameras speed.

Calendar

Description automatically generated