**Final Project Reflection**

**Navigation**

**Controls**

**System**:

ESC - Exit App  
1 - Wireframe mode  
2 - Fill mode

**Lights**:  
F - Toggles Flashlight (spotlight)  
J - Toggles Directional Light

**Movement**:  
W - Forwards  
A - Left  
S - Backwards  
D - Right

Space or E - Up (Y-axis)  
Left Control or Q - Down (Y-axis)

**Camera**:  
P - Toggle between Perspective and Ortho Cameras  
  
Mouse - Look around  
Scroll Wheel Up - Increase Speed (0.1)  
Scroll Wheel Down - Decrease Speed (-0.1)  
Shift + Scroll Wheel - Same as above, but increases +- 0.5 (Modifies the new movement speed formula)  
Click Mouse Wheel - Reset Movement Speed back to 3.0f

Currently the scene is only controllable via Mouse and Keyboard, but the Process Input function can make calls to the Camera class that have abstracted the methodology of movement to allow for code to easily be added to the ProcessInput() to handle input from gamepads and other input devices easily without having to re-write the core movement logic.

All logical operations for movement are handled via the Camera class. All input reading is currently handled in the source class, which then makes the calls to the Camera class’s functions.

In later iterations, the ProcessInput method could be further abstracted via a class that can provide an API for any input device to easily call based on input.

**Development Choices**

There was a bug late in development that I kind of noticed when I was writing the logic to handle overlay textures (the label on the candle), where a texture with text was noticeably flipped on the X-axis (open GL does flip on Y, but you can change that). I didn’t think anything of it, and just flipped the texture’s UV generation logic in the class file for the object.

When I was finalizing the design and working on the last objects, I noticed that occasionally lights would not affect objects unless I rotated the scene (via glm::rotate) by 180 degrees. I determined that this is probably due to the normals being inverted (they are dynamically calculated for Cylinders and Spheres based on an algorithm found online). When you flip the model matrix, this resolves it, so this probably means that the back-faces are actually the front-faces, and then when you flip the model view, the faces correct themselves.

I later found that OpenGL expects Counter-Clock-wise building by default (which is only a problem on Cylinders, Spheres, and pyramids for me, since I hand-coded the other shapes and those are dynamic via loops). Since I am out of time, I bandaged the system with a ResetModelView method that sets the model view back to 1.0, then rotates by 180 for Cylinder and Sphere objects in the scene. Given more time, I would like to revisit this and correct the logic.

Another design decision caused by the necessity of time, was there are no 3D walls for container meshes like the candle jar. This reduces the number of polys, and provides an accurate scene depiction, but does cause the issue of no reflections on the inside of the containers due to back-faces not having normals.  
We also lose the thickness of the side walls, but this was a necessary corner to cut.

Last corner I purposefully had to cut was I wanted to animate a flame texture for the candle wicks. This would have involved a texture atlas, and on each frame incrementing the UV value on the plane that I rendered the texture to. (0.0 to 0.5 on frame 1, then 0.5 to 1.0 on frame say 30) This would have allowed for texture swapping on a single object (plane) and allowed for simple animated textures. But due to time constraints, it was placed on the back burner for now.

I also would have liked to get my glass to work, but I ran out of time when trying to make the texture and shader work.

**Modularity**

I took all primitive meshes used and gave them a custom class, this class dynamically generates the UV, Normals, and Positions of the mesh, as well as the VBO and VAO for the object. The class also provides the Draw method for drawing the object and a Deallocate call for deallocating the resources when the scene is closed.

This dramatically cut the code down and provides a simple way to add a primitive, and draw it with just 2 lines of code in the source (excluding the import).

I did the same with the Textures via a Texture2D class, which provides a means for importing textures with a single line, and handles both textures with Alpha and full-opaque textures.

The shaders also have dynamic functionality to handle the use of overlay textures. This allows a single shader to be used to handle a variety of objects, and in theory allows for all objects to have a transparent overlay, such as the label on the candle, to be displayed over its primary texture. (The overlay texture does support its own diffuse as well).

If I had more time, I would have looked to implement consolidation of the Diffuse and Specular textures to a single Texture2D class, since normally you would use the matching specular and diffuse at the same time. Then I would provide a call to automatically bind the specified textures to given Active Texture IDs.

Once that was done, I could have provided a means to specify what texture was assigned to a mesh via the actual object and consolidated all texture assigns to the Mesh Object’s Draw method rather than having to remember what texture(s) go with what mesh.

TL;DR: Given more time, I would have liked to consolidate matching Diffuse and Speculars to a single Texture Object, that can then be assigned to a Mesh Object (such as Cylinder, Sphere, etc) and have the Mesh Object handle its own textures.