For my 3D Project, I chose to recreate a simple, visually compelling layout that includes 4 objects staged in a clear background scene. The scene includes four distinct low- polygon objects using an Earl Grey Tin, simple coffee mug, a tall wine glass and fragrance spray bottle. But in composite primitive shapes express, I had a granite plane, torus, cube, sphere, cylinder with a combination of tapered cylinder. The variety allowed me to fulfill the projects’ requirements to include at least four primitives, while also balancing depth and visual clarity.

Although, the selection of the objects was easy, I did run into a controllable factor because at first the items were overlapping with each other. After fixing the barrier, I was able to continue to use the easy, simple objects in an OpenGL primitive shape, and they offered enough complexity to natural lighting areas, interactivity, and spatial structure without exceeding a triangle limit, that I have learned. Two of the objects are composed of modeling techniques, such as stacking and transforming base meshes such as wine glass and coffee mug.

For textures, I applied them to all 4 of the objects, as close as possible, using the textures provided in file. After flopping the first time with implementing the textures for the objects, I had to do it down and look at it again in order to complete that section of the project. Each of the objects, including the plane, had a couple similar textures. The texture of the fragrance bottle had similarities to stained glass, wine glass (two objects making it one) color coordinate was camouflaged but the texture is close to a backdrop, Earl grey tin color texture is close to tile, and the plane was closer to cheese wheel.

In terms of lighting, this stage was the hardest because of the specific staging of the Phong shading model. White directional light and colored point light (soft purple glow). Adjusting the ambient, diffuse, and specular strengths allowed me to somewhat achieve a balance of illumination without overexposing or really deep shadows. In this area of lighting, I can see that the skill behind adjusting the lighting so that all objects have a clear definition to themselves and are not overpowered by an object such as the coffee mug itself will require a deeper understanding on what section of the code can I adjust to change the placement of the light.

To meet the navigation criteria, I programmed camera movement using:  
WASD & QE keys to enable full freedom of motion along the X, Y, & Z axes.

In the section with Mouse\_Position\_Callback, I attempted to run a code that will let the mouse input pinch, and yaw based on orientation. What I was able to do was create the code, add to the project and focus on the errors and calling the specific movement. The final implementation of the V key that switches the camera’s projection matrix between perspective and orthographic views, preserving the camera’s orientation and ensuring continuity during the switch. In the end, I was not successful in adding V key action. Overall, my development choices reflect a lot of reflection, focus on simplicity, great number of clarities, and demonstrate the full rubric coverage while attempting to deliver a clean, interactive experience.

Users can freely navigate the 3D environment using both the keyboard and mouse input:

* W/A/S/D keys can move the camera forward, left, backward, and right along the horizontal plane
* Q/E allow movement on the vertical plane, giving up and down enabling full 3D transversal
* Mouse Movement controls the cameras eye level movement.

Pending implementation-

* V key toggles between perspective (3D Depth) and orthographic (2D flat) projections, without resetting the camera’s orientation.
* Additional Mouse Movement that will allow controls the camera’s orientation with pitch (up/down) and yaw (left/ right) simulating first- person or fly through camera perspective.

The controls make the experience interactive and allow the used to examine the models from certain angles or distances. Here are some custom functions developed to maintain modularity:

* SetShaderColor(r, g, b, a)- which applies a solid RGBA color to the objects by passing a **vec4** into the shader. But also, it can be called before any draw call to change object colors dynamically.
* SetPointLightColor(r, g, b)- sets the RGB color of the point light in the scene. But used during initialization light mood needs adjustments (toggling light themes).

By compartmentalizing behaviors into focused, reusable methods, the codebase stays maintainable, readable, and ready for upgrades- whether its new objects, lighting effects, or input devices. In the end, I realized it was essential to fully understand the code for calling shapes and configuring lighting before implementation. Each section required a clear, simple comment to keep everything organized and cohesive.