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

The reason that I chose the objects that I did for my model is simple really. I had them laying around the house, and they all seemed possible to produce for the scene. The two most complex objects in my 3d scene were the half gallon milk carton and the moka pot. The milk carton was deceptively difficult to model. It involved four separate 3d shapes: a triangular prism, a box, a thin box for the tab on the top, and the cap where you pour the milk from was a small cylinder. What made this object so difficult was the fact that it was rotated -20 degrees along the y-axis. This made positioning the 3d objects in relation to that shift extremely difficult. The cap on the milk carton was probably the most difficult shape in the entire 3d scene to position, entirely because it needed to match the rotation along the y-axis with the milk carton so that I could then rotate it towards the main camera along that new y-axis. This required a lot of trial and error. Not only this, but to adjust the protrusion of the cap alone required me to adjust its positioning on the x, y, and z planes, all because its parent object was adjusted along its own y-axis. The moka pot was simple by comparison, being just a few boxes for the handles, two mirrored tapered cylinders for the body, two regular cylinders for the middle of the body and the handle on the top, and two prisms stacked within each other to simulate a spout with coffee inside of it.

I wanted the 3D scene to be a challenge, but I also wanted it to be simple enough so that I could understand how lighting works. I believe that even the simplest of scenes can really shine with the right lighting conditions.

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

A user can easily navigate my 3D scene with the traditional WASD keys representing forward, backward, left, and right. The Q and E keys also represent up and down. Not only this, but I wanted a way to increase the sensitivity of the motion, because navigating the 3D scene can be a slow process sometimes, so the scroll wheel increases or decreases the sensitivity of the camera's motion in the 3D scene. I enabled the program to detect mouse inputs allowing for full control over where the camera is pointing within the 3D scene.

Along with the directional and intuitive mouse based controls, you also have control over the perspective of the 3D scene. This can be extremely useful when trying to align objects with each other without gaps and to see the scene from every perspective. As well as looking at the scale of objects in relation to one another. What may look normal and good in a 3D perspective, may look outlandish in a 2D, or orthographic perspective. To switch between these two view perspectives, I have mapped perspective, or the 3D perspective to the P key, and I have mapped the orthographic perspective, or the 2D perspective to the O key.

**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?**

The sheer number of objects that were being rendered in the scene was steadily increasing. Each 3D object consists of multiple simpler 3D objects. Because of this, I needed a way to organize each rendered complex object in its own helper function. This process was relatively simple. All I needed to do was add the method to the SceneManager header file, and that would enable us to hoist the function from anywhere within the SceneManager cpp file. This meant that every complex object had its own self explanatory title, and collapsable function. This enabled me to collapse all of the functions that I was not actively working on to quickly navigate the page. As a side effect, the RenderScene function became much cleaner and easier to maintain.

I took this same logic and applied it to the materials, lights, and textures, so that everything could live within its own function, increasing readability and modularity. Because of this, all of the lights live in one function. All of the materials live in their own separate function. And all of the textures can be imported, and live in their own function. Now, if I need to import another texture, I have a place to do that. If I want another material, I have a place to create it. And if i need to create another complex object, I have an area to make it in that won't crowd the RenderScene function.