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 objects that I selected to use in this project were the various component pieces that make up the greater ziggurat structure that is and was the Great Ziggurat of Ur located in modern day Iraq. This was a structure that was originally built in the early Bronze-Age by the Neo-Sumerian empire, also known as the third dynasty of Ur.

The structure was composed of three tiered layers, each a smaller version of the previous to form somewhat of a pyramidal shape. The smaller component shapes were created using square frustum. This square frustum is what is used to create each of the stepped layers of the ziggurat. The implementation started out as a copy of the box shape that we were given, and I then adjusted each of the four edges that make up top plane of the box to be approximately 80% of the length of the four edges that make up their corresponding bottom edges / base plane. The result was a box that slopes inwards similar to a four sided pyramid, but that has a flat top instead of a point.

I also needed to use custom texture coordinates primarily on the sides of the square frustum to account for the difference in size from the base to the top. I started out using multiples of the smaller sized value of 0.4 (80% of .5) with 3.2 and 1.6 seeming to fix the skewing problems. I then reduced this value to fit within the standard sizes of 0 to 1.0, and ended up with working texture coordinate values between 0.4 to 0.16 for the sloped sides.

I also needed to create the shape that was used for my ramps that are used to ascend to the ziggurat's peak. In my first iteration I used the provided prism shape, but noticed that it was not in the desired format for my structure. What I needed was a single prism shape, whose triangular sides were in the shape of a right triangle. The provided prism contained a second prism shape, and they both contained obtuse triangles.

I extracted a single triangular prism from the provided dual prism shape mesh, and used it to construct a single right triangle prism that then worked perfectly for my ramps. I primarily only needed to adjust the values for the hypotenuse's face/plane, and to then adjust the texture coordinates to remove the diagonal skewing.

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.

For standard navigation of my 3D scene, I used W, A, S, and D keys as the directional controls for camera movement. W causes the camera to move forward, S to move backwards, A to move left, and D to move right. I also added binding for Q to move the camera up, and E to move the camera downwards.

I also added keyboard bindings for the camera to switch the perspective from the standard to either Orthographic by pressing the O button on the keyboard, or projection by pressing the P button on the keyboard.

All of these keyboard mappings are assigned within the ProcessKeyboardEvents function of the ViewManager class.

For panning related camera functionality, I implemented controls for mouse movement, with movement into each of the directions corresponding with a directional pan of the camera. Moving the mouse left pans the camera left, right pans right, up pans the camera up, and down pans the camera down. I adjusted these values to be less sensitive, so as to make the camera panning easier to manage. This was done in the ViewManager, MousePositionCallback function, by multiplying the x and y offset values by the camera sensitivity float variable that was added, this is done each time movement is

registered, and before the updated x and y offset values are used to call the ProcessMouseMovement function of the Camera object.

I also implemented a mouse scroll wheel binding that controls the speed at which the camera will move around the 3D environment, with scroll up increasing the movement speed, and scroll down decreasing the movement speed. This was done by modifying the camera header file, in the ProcessMouseScroll function. I added an adjust speed modifier float variable which is multiplied by the yoffset parameter before being incremented to the movementSpeed variable. This is called from the ViewManager MouseScrollCallback function whenever a scroll input is registered.

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 implemented several custom functions to reduce the amount of duplicate code needed within the RenderScene function of the SceneManager class. The following functions are reusable, and are used to configure and render each of the shapes that are used in my 3D scene.

The first of these custom functions is my ConfigShape function. It takes four arguments, a three value vector for scaleXYZ, a three value vector for positionXYZ, a SHAPE enum value, and a three value vector for the rotationXYZ. The rotation is passed last, as I primarily only ended up needing to use it for the configuration of my ramp right triangle prism shapes. Rotation is assigned a default value of 0,0,0 within the header file declaration of the configShape method to remove the need to pass it in any other function calls. The vector values for the scaleXYZ, positionXYZ, and rotationXYZ are initially set to 0,0,0 at the start of the renderScene method.

A case statement determines what shape is being configured using the passed in enum Shape typeShape, this case statement assigns both the shaderTexture, and the shaderMaterial depending on the shapeType being configured. A series of conditional branches to add additional texture scaling functionality is executed following the case statement. The branches set a custom UVScaling value for the ziggurat shape, and the right triangle prism shape, for any other shapes, a default UVScale width and height of 1.0 is used. The SetUVTextureScale function is then called for the shape, followed by the SetTransformations function to finalize the shape configuration. The return value from this shape is the positionXYZ of the shape, with added values for the next shape to position slightly elevate it on the y axis so the next shape is not buried. As the updated positional value for the next shape is passed back as a new positionalXYZ vector, the positionXYZ vector set up at the beginning of the RenderScene method is not used, only the positional data that is passed back as a vector return from the configShape. The positionXYZ can be used as a 0,0,0 reference when needed in later calls. This aids reusability in that the only values that must be set before calling the configShape function is the scaling values for the shape that will be configured. The positional vector values used for the next shape are modified based on one of the prior named positional vectors returned from the configShape function.

When the configShape function returns to the renderScene function, the DrawShape is then called using the shapeType enum value that was used to call the configShape function. This DrawShape function is simple custom function that contains a case statement which calls the correct DrawShape method using the shape mesh that corresponds to the enum shapeType value passed as an argument, and that was just configured.

The last custom function that I added was the AddRamp method. The AddRamp function is an additional step to the shape configuration process, used exclusively for the right triangle prism shape configuration process. The same vector arguments that are used to call the configShape method are

used to call the AddRamp method, with the exception of enum Shape shapeType argument being replaced by an enum Ramp rampType value. This method begins by setting up the positioning of the new ramp using the passed in x, y, and z scale values. It then assigns the rotational value for the new ramp based on the rampType enum parameter. The options for the rampTypes are standard, reversed, and perpendicular. Standard receives a -90 y rotational value, reversed receives a 90 degree y rotational value, and perpendicular receives a 180 degree y rotational value. The adjusted arguments are then used to call the configShape function, to finish the process of configuring the ramp. The drawShape function is then called at the end of the AddRamp function call, and no value is returned to the SceneRender method.