The objects I chose in my 3D objects include boxes, cylinders, cones, and tori because they represent a variety of basic shapes that allow me to show different scales, rotations, and positions. By combining these shapes, I can create more complex objects, such as a pencil (using a cylinder and a cone) or rings of a notebook (using multiple torus meshes). Being able to scale and rotate these shapes also shows how transformations work in 3D, demonstrating skills such as scaling for size, rotating around different axes, and translating to move objects around.

To program the required functionality, I wrote code that uses SetTransformations, SetShaderColor, SetShaderTexture, and SetShaderMaterial functions. These functions help me apply transformations and appearances to the shapes before they are drawn. For example, SetTransformations are taken in the scaling (scaleXYZ), rotation (XrotationDegrees, YrotationDegrees, ZrotationDegrees), and position (positionXYZ). Then, I store or pass these values to the graphics pipeline so the final shape displays correctly. SetShaderColor changes the color for the shape, while SetShaderTexture and SetShaderMaterial pick which texture and material properties should be used.

A user can navigate my 3D scene using keyboard inputs and a mouse (or potentially a gamepad/joystick). The keyboard can be set up so that the W, A, S, and D keys let you move forward, left, backward, and right, while the Q and E keys can let you move up and down. The mouse can be used to control where the camera is pointed, such as looking around by moving the mouse left, right, up, or down. This way, the user has full freedom to explore the scene by walking around and looking in any direction.

To control the virtual camera, I capture input events from the keyboard and mouse. Then, in the functions ProcessKeyboardEvents, Mouse\_Position\_Callback and ProcessMouseMovement, I adjust the camera’s position or its rotation angles. For example, pressing the W key will move the camera forward, while moving the mouse left changes the yaw (the rotation around the vertical axis). This allows the camera to point in different directions, effectively letting the user aim in 3D space. If I wanted gamepad support, I could also read the OpenGL documentation and apply similar logic.

I also created custom functions to keep my code organized and modular. The SetTransformations function, for instance, takes in the scale, rotation, and position vectors. It then applies these transformations to the mesh before drawing it. This function can be reused for every shape I want to draw in the scene, making my code cleaner and easier to change. Meanwhile, SetShaderColor, SetShaderTexture, and SetShaderMaterial each provide a concise way to apply visual settings without having to repeat all the code for binding or enabling each texture or material. This keeps the drawing process consistent and modular.