#### Justification of Development Choices

Selected Objects:

* Cones: The cones in the scene represent natural elements like trees or hedges, adding a sense of life and structure to the environment. Their varying positions and scales provide visual interest and demonstrate the ability to create repeated but uniquely placed objects.
* Squares: The squares represent buildings or walls, contributing to the structural aspects of the scene. They showcase the capability to create and manipulate geometric shapes and provide a contrast to the natural elements.
* Torus: The torus adds a unique and recognizable shape to the scene, representing either an abstract sculpture or a functional element like a fountain. It helps in demonstrating the diversity of shapes that can be rendered.
* Ground and Background: The large plane for the ground and the vertical plane for the background help to create a sense of space and environment. They set the stage for the rest of the scene and provide a context for the placement of other objects.

Required Functionality:

* Transformation Functions: Using functions like SetTransformations, I was able to programmatically control the position, rotation, and scale of each object, demonstrating an understanding of basic 3D transformations.
* Shader Setup: Functions like SetShaderTexture and SetShaderMaterial were used to apply different textures and materials to objects, showcasing the ability to enhance the visual appearance of the 3D models.
* Drawing Functions: The DrawConeMesh, DrawBoxMesh, and DrawTorusMesh functions demonstrate the capability to render different types of objects in the scene, essential for creating a diverse and interesting environment.

#### Navigation in the 3D Scene

Camera Control:

* Mouse Input: The virtual camera can be controlled using mouse movements, allowing the user to look around the scene by clicking and dragging the mouse. This provides an intuitive way to explore the 3D environment.
* Keyboard Input: The arrow keys or WASD keys are used to move the camera forward, backward, left, and right. This familiar control scheme enables users to navigate through the scene easily.
* Zoom Functionality: Using the scroll wheel on the mouse, the user can zoom in and out, providing better control over the viewing distance and enhancing the navigation experience.

Setup:

* The camera control is set up by capturing input events and updating the camera's position and orientation accordingly. This involves setting up event listeners for mouse and keyboard inputs and writing functions to adjust the camera's properties based on the received input.

#### Custom Functions for Modularity and Organization

Function: SetTransformations

* Purpose: This function sets the scale, rotation, and position of an object. It takes parameters for scale (as a vector), rotation angles (in degrees for each axis), and position (as a vector).
* Reusability: By abstracting the transformation logic into a single function, it can be reused for any object in the scene, simplifying the code and making it more readable and maintainable.

Function: SetShaderTexture

* Purpose: This function applies a texture to the shader used for rendering an object. It takes the name of the texture as a parameter.
* Reusability: It allows any object to be textured with different materials, making the code modular and enabling easy changes to the visual appearance of objects without altering their fundamental properties.

Function: SetShaderMaterial

* Purpose: This function sets the material properties for the shader. It takes the name of the material as a parameter.
* Reusability: Like the texture function, this enhances modularity by allowing different materials to be applied to objects, facilitating easy experimentation with different visual effects.

Drawing Functions: DrawConeMesh, DrawBoxMesh, DrawTorusMesh

* Purpose: These functions encapsulate the drawing logic for different types of meshes. Each function is responsible for rendering a specific type of object (cone, box, torus).
* Reusability: By separating the drawing logic into individual functions, it becomes easy to add new types of objects to the scene. Each function can be called with different transformation and shader settings, allowing for a highly customizable and flexible scene setup.

### Conclusion

The choices made in developing this 3D scene were driven by the need to create a visually interesting and diverse environment while demonstrating fundamental 3D programming skills. The navigation setup ensures an intuitive user experience, and the use of custom functions enhances the modularity and maintainability of the code. This approach not only meets the project requirements but also lays a solid foundation for further development and enhancements.