For this 3D scene, I selected guitar amplifiers as my primary objects because they consist of simple geometric shapes while still being recognizable real-world objects. By using box primitives, I was able to efficiently create multiple amps with different dimensions and textures while keeping the polygon count low to meet the requirements.

The texture choices (black and tweed) are my best guess at trying to imitate two very common guitar amp covers. The tweed was a little more straightforward than the black. I ensured the textures met the resolution requirement (1024x1024 or higher) and that they were accurately projected onto the objects.

For lighting, I implemented three light sources, including a point light and a colored light, to enhance realism and create a visually engaging environment. The Phong shading model was used to improve material realism by incorporating ambient, diffuse, and specular components.

To meet the required functionalities:

* I created 3D models were created using box primitives
* Texturing was applied using the CreateGLTexture() function, which loads and binds textures for reuse.
* Lighting was implemented in SetupSceneLights(), ensuring multiple light sources interact dynamically with objects.
* Navigation was programmed using WASD for movement, QE for vertical motion, and the mouse for looking around.
* Camera projection toggling was implemented, allowing users to switch between perspective and orthographic views dynamically.

Users can freely navigate the 3D scene using keyboard and mouse controls:

* WASD keys: Move forward, backward, left, and right.
* Q / E keys: Move up and down for full vertical control.
* Mouse movement: Adjusts the camera's yaw and pitch to look around.
* Mouse scroll wheel: Dynamically adjusts movement speed, allowing for finer control in exploration.
* P key: Switches to perspective view (default 3D view).
* O key: Switches to orthographic view (2D-style projection).

These controls allow the user to explore the scene with ease.

The camera system is handled using a Camera object that maintains its position, front direction, and movement speed. The camera is updated in real-time based on user input. The following techniques were used:

1. Keyboard Controls (ProcessKeyboardEvents())
   * Adjusts the camera position based on movement inputs (WASD, QE).
   * Uses glm::normalize(glm::cross(Front, Up)) for accurate lateral movement.
2. Mouse Movement (Mouse\_Position\_Callback())
   * Tracks x and y offset values to rotate the camera smoothly.
   * Implements sensitivity adjustments to fine-tune rotation speed.
3. Mouse Scroll (Mouse\_Scroll\_Callback())
   * Adjusts the camera movement speed dynamically.
   * Ensures speed remains within reasonable limits (0.5f minimum).
4. Perspective & Orthographic Projection (PrepareSceneView())
   * Toggles between perspective (3D) and orthographic (2D) views using P and O keys.
   * Uses glm::perspective() for a realistic view.
   * Uses glm::ortho() for a flat view.
5. PrepareScene() (SceneManager)
   * Loads models, materials, and textures into memory.
   * Ensures everything is pre-loaded before rendering begins.
   * Makes the scene setup modular so additional objects can be added without modifying the rendering loop.
6. RenderScene() (SceneManager)
   * Calls SetTransformations() and DrawBoxMesh() for each object.
   * Allows new objects to be added with minimal extra code.
   * Uses SetShaderTexture() and SetShaderMaterial() to apply textures/materials efficiently.
7. SetTransformations()
   * Takes in scale, rotation, and position parameters to apply transformations dynamically.
   * Ensures object positioning and orientation can be modified easily.
8. CreateGLTexture()
   * Loads images, generates OpenGL textures, and binds them for reuse.
   * Prevents duplicate texture loading by associating each texture with a unique tag.
9. SetupSceneLights()
   * Defines multiple light sources with individual positions, colors, and intensities.
   * Ensures consistent lighting effects without modifying shaders manually.
10. ProcessKeyboardEvents() (ViewManager)
    * Handles all user inputs efficiently.
11. Mouse\_Position\_Callback()
    * Converts mouse movement into camera rotation.
    * Provides smooth first-person style controls.
12. Mouse\_Scroll\_Callback()
    * Adjusts movement speed based on user preference.
    * Allows control of exploration speed with the mouse wheel.