In designing my 3D scene for CS-330 Computational Graphics and Visualization, I made a series of deliberate choices regarding the objects, materials, lighting, and overall scene layout. My objective was to create a visually interesting and technically functional environment that would demonstrate the key principles of 3D graphics, including transformations, camera movement, shader control, and texture mapping.

The objects I selected were basic geometric primitives such as cubes, spheres, and planes, as defined in the ShapeMeshes module. These objects were chosen because they provide simple yet effective surfaces for applying textures and lighting. They also serve as a solid foundation for showcasing shader functionality. The layout of the scene includes a ground plane, several 3D shapes positioned at various locations and scales, and dynamic lighting to highlight the materials applied to each object.

From a rendering perspective, I implemented ambient, diffuse, and specular lighting components using custom object materials. These materials are defined in the SceneManager class using the OBJECT\_MATERIAL struct, which includes values such as ambient strength, shininess, and RGB color components. Each material is tagged and reused across multiple objects, which allows for clean and modular control of visual styles throughout the scene.

Textures were loaded and managed using the CreateGLTexture and BindGLTextures functions. These encapsulated the OpenGL texture setup and memory binding operations, ensuring reusability and easier debugging. Each texture is assigned a unique tag and ID, and textures are bound in a fixed array to prevent memory overflows. This structured approach supports modularity and efficient memory management.

The navigation and interactivity within the 3D scene were handled primarily through the ViewManager class. This class manages user input, the camera system, and projection setup. Users can navigate the scene using familiar controls:

* **W/S** to move forward/backward
* **A/D** to pan left/right
* **Q/E** to move down/up
* **Mouse movement** to look around
* **Scroll wheel** to increase/decrease movement speed
* **O/P keys** to toggle between orthographic and perspective projection views

To make the program more modular, I developed and used several reusable functions. For example, SetTransformations() in SceneManager abstracts all transformation logic (scale, rotation, position) and sends transformation matrices to the shader. Similarly, SetShaderMaterial() centralizes material settings so that lighting calculations can be applied consistently to all objects using tagged materials. The ProcessKeyboardEvents() function abstracts keyboard input logic, checking for specific keys and updating the camera accordingly.

This modularity allows for easier maintenance and expansion. If a new object, material, or camera behavior needs to be added, the codebase is organized enough to support such updates without significant refactoring. The division between ViewManager (view and input) and SceneManager (setup and rendering) adheres to the principle of separation of concerns, which makes the program easier to understand and extend.

*Welcome to OpenGL*. Learn OpenGL, extensive tutorial resource for learning Modern OpenGL. (n.d.). https://learnopengl.com/