# 3D Scene Reflection

To ensure the program runs as expected, I employed functional coding logic best practices by organizing my code into well-defined, modular functions that each serve a specific purpose. For instance, the SetTransformations() function handles all 3D object transformations, including scaling, rotation, and translation, making it easy to reuse for every object rendered in the scene. Similarly, SetShaderTexture() and SetShaderMaterial() abstract the logic of applying textures and materials, reducing duplication, and improving consistency. Each function was written with a single responsibility in mind, which not only aligns with clean coding principles but also makes the code easier to debug, test, and expand. I also utilized functions like CreateGLTexture() to efficiently load and configure image files, and FindTextureSlot() to retrieve texture bindings by unique tag names, both of which support future scalability if new textures are introduced.  
Following functional best practices, I also placed strong emphasis on clarity and documentation. I made sure each method is preceded by a clear header comment that summarizes its purpose, and I added in-line comments throughout complex logic sections. This is especially visible in the PrepareScene() function, where I define object materials using diffuse, specular, and shininess properties. Each block is labeled to clarify which material is being configured and what visual effect it contributes to the scene. These comments are helpful not only for other developers reviewing the code, but also as a reference point for myself during future updates. This approach ensures that my work is both readable and maintainable.  
  
For the visual design of my 3D scene, I chose to create a landscaped path bordered by decorative bushes and illuminated by a realistic lamp post. I selected these objects because they offer a balanced combination of organic and structural elements referenced within my 2D image, enabling me to highlight a wide range of techniques in object creation, texture mapping, and lighting. The path and grassy ground plane provide a foundational base using tiled textures to enhance realism without stretching. The bushes are constructed from spheres for the foliage and cylinders for stems, demonstrating how multiple primitives can be combined to create a cohesive natural object. Meanwhile, the lamp post assembled from a cylinder for the pole, a box for the lamp casing, and a sphere to simulate a glowing bulb, allowed me to integrate point and directional lighting and simulate reflections using specular material properties.  
One of the more difficult challenges in this project was correcting the lighting to ensure realistic results that enhanced the visual quality of the scene. A major issue I encountered was finding the right balance between ambient, diffuse, and specular light values. When ambient lighting was too strong, it flattened the appearance of the objects, making them look dull and removing the sense of depth. When it was too low, shadows became overly dark, and some shapes became hard to distinguish. Tuning the point light for the lamp post was particularly tricky; the attenuation values for constant, linear, and quadratic falloff had to be carefully adjusted to make sure the light glowed naturally without overpowering nearby objects or disappearing entirely. Specular highlights on reflective materials such as the glass lamp casing or metallic surfaces also required careful tuning, too much shininess made them look artificial, while too little made them appear matte and lifeless. I also had to ensure that all objects had proper normals and lighting toggles set in the shaders; a single incorrect uniform variable or lighting flag would cause parts of the scene to render without proper illumination. Through iterative testing and refinement, I was able to resolve these issues and produce lighting effects that added realism without sacrificing performance or clarity.  
Programming the required functionality was supported by the modular design of my helper functions. Instead of writing the transformation and material logic from scratch for each object, I relied on reusable methods that accepted parameters for scaling, position, and texture. This allowed me to maintain consistency in how shapes were drawn and reduced the risk of introducing small logic errors. It also enabled me to experiment freely with object placement and size, knowing I could do so within a structured and reliable framework.  
The user can interact with the 3D scene through standard input devices such as a keyboard and mouse, which are handled through the ViewManager class. This class enables camera navigation via mouse movement and keypresses, allowing users to rotate their view, move through the scene, and examine objects from different perspectives. Scroll callbacks can also be configured to control zooming, making the experience feel more immersive and interactive. I implemented a perspective projection to create depth and realism, giving the illusion of being able to walk through the landscaped area as if navigating in first-person.  
  
Finally, to maintain code clarity and flexibility, I continued reinforcing modular design through custom utility functions. For example, SetTransformations() became the backbone for adjusting object placement in the scene and could easily be reused for any new mesh introduced. Likewise, the material and texture setter functions allow new visual styles to be introduced without altering the underlying rendering logic. These choices made the codebase not only robust but also adaptable to future improvements, expansions, or entirely new scenes. By following functional and commenting best practices throughout this project, I have ensured that both the scene and the source code are clean, interactive, and professionally structured.