Author: Keyvan Gharehtapeh

Date: 08/17/2025

**3D Scene Development Reflections**

The primary goal in developing this 3D scene was to create a visually appealing and realistic representation of a backyard oasis, centered around a swimming pool and hot tub. The choice of objects was driven by a desire to build a cohesive and inviting environment. The pool and hot tub serve as the central focus, immediately establishing a theme of relaxation and leisure. Elements like the stone patio, retaining walls, and outdoor lighting were selected to complement this theme and add a sense of completeness and realism to the scene.

The development process began with the foundational elements. The ground plane was established first, textured with paving stones to create a realistic patio effect. The pool and hot tub were then constructed using basic mesh shapes—primarily boxes and cylinders. The decision to use these primitive shapes was a practical one; they are computationally efficient and can be scaled, rotated, and combined to create more complex forms. For instance, the hot tub was created by layering cylinders of slightly different sizes to form the main basin, the water's surface, and the surrounding ledge.

A significant challenge was achieving a sense of realism. This was addressed through the careful application of textures and materials. I sourced high-resolution textures for stone, water, and pavers to give the surfaces a tangible quality. The CreateGLTexture function was instrumental here, allowing for the loading and binding of these textures. Furthermore, the DefineObjectMaterials function was used to specify how these surfaces interact with light. For example, the water material was given a high shininess value and a blueish diffuse color to simulate its reflective and translucent properties, while the stone was given a rougher, less reflective material to appear more natural.

Lighting played a crucial role in setting the mood of the scene. I implemented multiple light sources to create a dynamic and visually interesting environment. A directional light was used to simulate ambient sunlight, providing overall illumination. Additionally, two point lights were strategically placed. One was positioned inside the light fixture on the back wall to create a warm, localized glow, making the scene feel more alive and functional. Another was placed near the hot tub to highlight it as a focal point. The SetupSceneLights function was used to configure the position, color, and intensity of these lights, allowing for precise control over the scene's ambiance.

**Scene Navigation and Camera Control**

A key aspect of any 3D environment is the ability for the user to explore it. To facilitate this, the logic to control the camera's movement through the keyboard and mouse was implemented as follows: 🖱️

* **Rotating the Camera:** By moving the mouse, the user can rotate the camera horizontally and vertically around the scene. This is achieved by capturing the mouse's movement and using it to update two angles: yaw (for horizontal rotation) and pitch (for vertical rotation).
* **Pan and Zoom:** The user can zoom in and out using the W and S and pan the camera left, right, up or down. This action modifies the position of the camera.

**Modular and Reusable Code**

To keep the codebase organized and efficient, there are several custom functions that encapsulate specific, repeatable tasks. This modular approach makes the code easier to read, debug, and extend.

* **SetTransformations(scale, rotX, rotY, rotZ, position)**: This is perhaps the most frequently used custom function in the RenderScene method. Its purpose is to take simple scale, rotation, and position values and construct a complete model matrix. It handles the complexities of creating individual scale, rotation, and translation matrices and multiplying them in the correct order. This function is highly reusable; it is called before rendering every single object in the scene, from the ground plane to the smallest light fixture. By abstracting this process, I avoided writing repetitive and error-prone matrix calculations for each object.
* **SetShaderColor(r, g, b, a)**: This function simplifies the process of setting an object's color. It takes red, green, blue and alpha values, creates a glm::vec4, and passes it to the shader. It also tells the shader to disable texture mapping for that object. This is useful for rendering simple, untextured objects like the light bulb or for applying a base color to an object that will also have a material applied. Its reusability is evident when rendering multiple objects with solid colors.
* **SetShaderTexture(textureTag)**: This function handles the setup for textured objects. It takes a string tag (e.g., "water" or "stone"), finds the corresponding texture slot, and binds it for the next draw call. It also tells the shader to enable texture mapping. This function is essential for applying the various textures throughout the scene and makes the RenderScene code much more readable, as I can refer to textures by a descriptive name rather than an integer ID.
* **SetShaderMaterial(materialTag)**: Similar to the texture function, this one sets the material properties (diffuse color, specular color, and shininess) for an object. It uses a string tag to find the desired material from the pre-defined list and sends its properties to the shader. This allows for the easy application of complex lighting characteristics to objects, enhancing their realism. This function is reused for every object that needs to interact with the scene's lighting in a specific way.

By creating these helper functions, the main RenderScene function is more declarative. Instead of being a long list of low-level OpenGL and GLM calls, it reads more like a descriptive list of instructions: set the transformations for an object, give it a texture and material, and then draw it. This modular design was key to managing the complexity of the scene and building it in an efficient and organized manner.