**Design Decisions for 3D Scene Implementation**

This document outlines the design decisions made during the development of the 3D scene that replicates a real-world environment of a reading nook using OpenGL and C++. The implementation follows computational graphics best practices while meeting all project requirements the various milestones throughout the course. The scene features a dining table setting with accompanying furniture and lighting elements, created through careful composition of primitive shapes, texture mapping, and lighting techniques.

**Object Creation and Composition**

The scene contains five primary objects constructed from basic geometric primitives: a wooden table, two fabric stools, a ceramic cup, and a glass chandelier. Each object was designed with low-polygon counts while maintaining recognizable forms. The table represents the most complex object, composed of five box primitives (four legs and one tabletop) with a total polygon count of approximately 120 triangles. This design decision balanced visual fidelity with performance considerations.

The ceramic cup demonstrates composition using multiple primitive types, incorporating a torus for the base (≈320 triangles), a cylinder for the body (≈96 triangles), and a smaller torus for the handle (≈320 triangles). This combination created a recognizable drinking vessel while keeping the total polygon count under 1,000 triangles. The stools each use two box primitives (≈48 triangles total per stool) to form the base and padded seat.

**Texture Implementation**

Texture mapping was applied to enhance realism while maintaining performance. The wooden table uses a high-resolution (1024×1024) oak texture sampled from royalty-free sources on AmbientCg.com, with UV scaling set to 3.0 to prevent visible tiling artifacts. The ceramic cup features a cream-colored ceramic texture at 1024×1024 resolution, providing sufficient detail for close inspection. Both textures were loaded using the stb\_image library and bound to texture units 0 and 1 respectively.

The implementation includes texture parameter optimization:

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

These settings ensure proper texture repetition and filtering while minimizing visual artifacts. Texture memory is managed efficiently through the DestroyGLTextures() method, which properly releases resources when the scene closes.

**Lighting Configuration**

The lighting system employs four distinct light sources to create a balanced illumination environment:

1. **Primary directional light** (index 0) simulates sunlight with warm white coloration (RGB 1.0, 0.95, 0.9) positioned at (0,15,0) to cast downward shadows. This light uses high specular intensity (0.08) to create pronounced highlights on glossy surfaces.
2. **Secondary key light** (index 1) provides complementary illumination from the front (0,15,4) with cooler white tones (RGB 0.6,0.6,0.6) and stronger specular contribution (0.4) to enhance surface details.
3. **Ambient fill light** (index 2) creates base illumination (RGB 0.5,0.5,0.5) from a central position (0,7,0) to soften shadows and ensure no areas become completely dark.
4. **Rim light** (index 3) positioned behind the scene (0,6,-6) adds depth perception with subtle highlights (RGB 0.4,0.4,0.4) along object edges.

The Phong shading model is fully implemented with separate controls for ambient, diffuse, and specular components for each material. For example, the ceramic material uses:

material.ambientColor = glm::vec3(0.95f, 0.92f, 0.85f);

material.ambientStrength = 0.4f;

material.diffuseColor = glm::vec3(0.96f, 0.93f, 0.86f);

material.specularColor = glm::vec3(0.9f, 0.9f, 0.88f);

material.shininess = 96.0f;

This configuration produces realistic ceramic reflectance properties with warm ambient tones and bright specular highlights.

**Camera Navigation System**

The camera control system implements six degrees of freedom for comprehensive scene exploration:

* **WASD keys** provide horizontal movement along the X and Z axes
* **QE keys** control vertical movement along the Y axis
* **Mouse movement** adjusts view orientation through pitch and yaw
* **Mouse scroll** modifies movement speed for precision navigation

The camera orbits around the scene center point at a dynamically adjustable radius, ensuring all objects remain visible. The implementation includes smooth interpolation between positions to prevent jarring transitions. Perspective and orthographic projection modes can be toggled via keyboard input, with the camera maintaining its current orientation during the switch.

**Code Organization and Best Practices**

The implementation follows several key software engineering principles:

1. **Modular Design**: The SceneManager class encapsulates all scene-related functionality, separating concerns from rendering and shader management handled by ShaderManager.
2. **Resource Management**: Texture and material resources are tracked through arrays with maximum limits (16 textures) to prevent memory leaks. The destructor properly cleans up all allocated resources.
3. **Reusable Functions**: The SetTransformations() method centralizes model matrix calculations, reducing code duplication when positioning objects:

void SceneManager::SetTransformations(

glm::vec3 scaleXYZ,

float XrotationDegrees,

float YrotationDegrees,

float ZrotationDegrees,

glm::vec3 positionXYZ)

{

glm::mat4 modelView;

glm::mat4 scale = glm::scale(scaleXYZ);

glm::mat4 rotationX = glm::rotate(glm::radians(XrotationDegrees),

glm::vec3(1.0f, 0.0f, 0.0f));

*// ... additional transformations*

modelView = translation \* rotationX \* rotationY \* rotationZ \* scale;

m\_pShaderManager->setMat4Value(g\_ModelName, modelView);

}

1. **Error Handling**: Texture loading includes comprehensive status checking:

if (image) {

*// Process successful load*

} else {

std::cout << "Could not load image:" << filename << std::endl;

return false;

}

Conclusion

This 3D scene implementation demonstrates effective use of computational graphics principles while meeting all specified requirements. Through careful composition of primitive shapes, strategic texture application, and balanced lighting, the scene achieves visual fidelity without excessive computational overhead. The modular code organization and comprehensive input handling create an interactive environment suitable for both demonstration and extension. Future enhancements could include additional objects, advanced shading techniques, or physics-based interactions to further increase realism.