**CS 330 Final Project – Design Decisions**

**Introduction**

The project's aim was to transform a two-dimensional tabletop model into a low-poly three-dimensional world using C++ and OpenGL. The focus was on meeting the requirements set out in the rubric, using a modular design approach, writing code that is both understandable and maintainable, and adding an interactive camera system that allows for extensive exploration of the world (Angel & Shreiner, 2018).

**3D Objects and Geometric Primitives**

The described scene features at least four unique objects, which in themselves are built up of differing geometric primitives. These objects include a plane representing the floor, a box representing the wooden table top, cylinders representing the legs of the table and the metal lamp's stand, a cone representing the textile lampshade, and a sphere representing an aesthetic marble object. It can be noted that the lamp itself forms a composite body crafted out of a union of a cylinder and a cone, hence qualifying for objects made of multiple primitives (Eberly, 2010). The procedurally generated meshes were constrained specifically to less than 1,000 triangles, specifically for all objects, to increase performance and conform to low-polygon standards.

**Texturing Methodology**

At least two objects used textures that were carefully obtained from royalty-free images, each with a resolution of 1024×1024 pixels. Wooden textures, for example, were used for both the flooring and the tabletop to maintain consistency in materials, while textile textures were used for the lampshade, metallic textures for the lamp stand, and a smooth marble texture for the sphere. Procedural methods were used to generate UV coordinates, using cylindrical unwrapping for cylindrical and conical shapes, equirectangular mapping for the sphere, and planar mapping for the plane and box. To reduce aliasing effects, especially during camera changes, techniques of mipmapping and linear filtering were used in the texture sampling (Shreiner et al., 2019).

**Lighting and Shading**

To enhance the visual realism of the scene, Phong shading was utilized, including ambient, diffuse, and specular terms (Phong, 1975). The scene is lit by two light sources: a white directional light that simulates global illumination and a warm-colored point light placed in a strategic location near the lamp to add realism to the lighting effect. Programmable control can be used to control the shininess of objects (default value: 16), allowing control over the range of specular highlights. In the process of mesh generation, vertex normals were computed using algorithms to support accurate lighting computation.

**Object Placement and Spatial Organization**

Affine transformations—including translation, rotation, and scaling—were employed to position objects in accordance with the 2D reference image. The floor plane anchors the scene spatially, the table is centered and offset along the Z-axis, and four legs are positioned relative to the tabletop corners. The lamp is positioned to the right of the table, while the marble sphere is located to the left to achieve visual equilibrium. All objects maintain a consistent scale, ensuring coherent spatial relationships and facilitating natural camera navigation.

**Camera Navigation and User Interaction**

Motion of the camera on the X, Y, and Z axes is achieved using the standard WASD + QE control scheme: W/S controls forward and backward movement of the camera, A/D controls side strafing, and Q/E controls vertical movement. The mouse controls yaw and pitch, while the scroll wheel dynamically controls the velocity of the camera to support detailed examination as well as quick scene navigation. Toggling between perspective and orthographic projections while maintaining the orientation of the camera is done by pressing the P key, hence supporting a comprehensive analysis of objects from both three-dimensional and two-dimensional views (Angel & Shreiner, 2018).

**Code Architecture and Modular Design**

The project follows a modular structure to promote code reuse, readability, and maintainability. The shader.h module encapsulates shader program creation and uniform variable management, while camera.h manages FPS-style camera movement and orientation. The main.cpp file orchestrates mesh generation, VAO/VBO setup, draw calls, and input processing. Dedicated builder functions for each primitive (plane, box, cylinder, cone, sphere) generate interleaved vertex data (position, normal, UV) and index buffers. Utility functions such as appendVertex minimize code duplication, enabling efficient addition of new objects or transformations. Comprehensive comments elucidate function purpose and critical mathematical computations, adhering to consistent naming conventions and formatting standards.

**Justification of Development Decisions**

A procedural approach was used for primitive construction to minimize dependence on external resources and to maintain minimum counts of triangles. The use of procedural UV mapping ensures texture preservation without relying on third-party UV manipulation. The implementation of a two-light setup (constituting a white directional light and a warm point light) improves form readability and highlights objects, thus conforming to rubric requirements concerning the usage of multiple lights and color variation. Object selection, comprising the table, lamp, and marble ball, directly corresponds to basic geometric primitives, enabling an accurate and scalable three-dimensional reconstruction of the initial two-dimensional reference (Eberly, 2010; Shreiner et al., 2019).

**Navigation Summary**

Keyboard controls: W/A/S/D (move), Q/E (vertical movement), P (toggle projection), ESC (exit). Mouse controls: orientation adjustment (yaw/pitch) and scroll wheel for dynamic speed control. The initial camera is positioned above the floor facing the table, allowing users to orbit freely and examine all objects within the scene.

**Reusable Functions and Abstractions**

Mesh builder functions (buildPlane, buildBox, buildCylinder, buildCone, buildSphere) are parameterized for flexibility (size, radius, height, segments) and facilitate reuse. The Mesh structure standardizes VAO/VBO/EBO creation and draw information. Together with auxiliary utility routines, these functions enable efficient object instantiation and transformation, fostering modularity and scalability.

**Project Outcomes**

The finished scene achieves or exceeds all rubric requirements: it includes four or more elements based on basic geometric shapes (with at least one composite shape), uses correctly implemented 1024×1024 textures, uses two lights that employ Phong shading methods, uses proper placement of objects, offers abundant three-dimensional movement facilitated by complex controls, and includes both perspective and orthographic projections. The code follows professional formatting, documentation, and module decomposition standards, evidencing a rigorous and professional approach to 3D graphics programming.

**References**

Angel, E., & Shreiner, D. (2018). *Interactive computer graphics: A top-down approach with WebGL* (7th ed.). Pearson.

Eberly, D. H. (2010). *3D game engine design: A practical approach to real-time computer graphics* (2nd ed.). Morgan Kaufmann.

Phong, B. T. (1975). Illumination for computer-generated pictures. *Communications of the ACM, 18*(6), 311–317. https://doi.org/10.1145/360825.360839

Shreiner, D., Sellers, G., Kessenich, J., & Licea-Kane, B. (2019). *OpenGL programming guide: The official guide to learning OpenGL, version 4.5 with SPIR-V* (9th ed.). Addison-Wesley.