**Jeanelyn Roman  
CS-499 Computer Science Capstone  
Dr. Troy Hawk  
Milestone Two: Enhancement One – Software Design and Engineering**

**Software Engineering and Design**

**CS-330 OpenGL 3D: Enhancing the Main Application File with Lighting, Texture, and Shader Integration**

The original main.cpp file served as the entry point for a 3D graphics application using OpenGL. Its core responsibilities included initializing the rendering context via GLFW and GLEW, setting up the rendering window, and calling the scene manager to display a basic 3D scene. However, the original version lacked realism, modular lighting, and support for advanced graphical features like textures and shaders.

**Enhancement Goals and Summary**

For Milestone Two, I revised main.cpp to enhance the visual fidelity and modularity of the 3D scene by implementing the following improvements:

* Added Phong lighting model uniforms
* Integrated a ShaderManager class for modular shader handling
* Loaded and applied a 2D texture using stb\_image
* Refactored code for improved readability and maintainability

**Details of the Enhancements**

**1. Phong Lighting Model**  
I introduced uniform variables to control ambient, diffuse, and specular lighting using the Phong reflection model. This provides more realistic lighting by simulating how light interacts with surfaces. Key elements now included are:

* lightSources[0] properties: position, color, specular intensity
* material properties: shininess, ambient strength, and color
* viewPosition for accurate specular reflection calculations

This brings more depth and realism to the scene, improving the overall user experience.

**2. Texture Integration**  
Using the stb\_image.h library, I created a loadTexture() function to read image data from disk and bind it as an OpenGL texture. I applied a rustic wood texture (rusticwood.png) to scene objects, which enhances the visual appeal by simulating real-world surfaces.

3. Shader Management System  
To make shader updates more modular, I utilized the ShaderManager class to load GLSL shader files externally:

cpp

CopyEdit

g\_ShaderManager->LoadShaders(

"../../Utilities/shaders/vertexShader.glsl",

"../../Utilities/shaders/fragmentShader.glsl"

);

This approach separates graphics logic from core application logic and aligns with software engineering best practices like separation of concerns and maintainability.

4. Code Structure and Documentation  
I organized the revised main.cpp into logical blocks: initialization, texture loading, shader setup, rendering loop, and cleanup. Sectioned comments were added for clarity and future maintainability.

**Challenges Encountered**

One major challenge was resolving missing file references for the shader and texture assets. I had to verify that all asset paths were related to the project root and that files like fragmentShader.glsl and rusticwood.png were stored correctly under the Utilities folder.

Another technical hurdle was ensuring that GLEW and GLFW were correctly linked within the building environment. This required reviewing my include directories and linker settings to avoid compilation errors.

This enhancement deepened my understanding of how OpenGL handles the graphics pipeline, particularly in the areas of lighting, texturing, and modular architecture. I gained valuable experience in balancing both visual design and software engineering practices within a C++ project.

The updated main.cpp now provides enhanced realism, modularity, and clarity, making it a stronger and more portfolio-ready artifact. It follows software engineering principles such as clean architecture, modular design, and visual realism through lighting and textures. This enhancement reflects my growth in both technical and architectural aspects of software development.