**Final Project Reflections:**

Course: CS-330

Professor: Ronald Bishops

Student: Brahim Benouari

1. **3D Scene Development and User Interaction**

* Object Selection Rationale

In this first computer graphics course, I designed a 3D scene that includes a diverse array of objects placed on a wooden platform to address various rendering challenges using OpenGL. This setup features items like a vase, hard drive, Alexa device, a book, and a candle stand, each chosen to demonstrate different aspects of 3D rendering:

* **Complexity in Shapes**: The candle stand, composed of multiple forms, showcases advanced OpenGL modeling techniques, illustrating how complex shapes can be constructed and rendered.
* **Surface Properties**: The objects vary in material properties—metallic, glossy, and textured—highlighting the effects that can be achieved with OpenGL shaders. This variety helps in exploring how different surfaces interact with light, enhancing the realism of the scene.

This project was designed to gradually introduce me to the fundamental and advanced concepts of computer graphics. By including both simple and intricate items, it allowed me to develop a thorough understanding of 3D rendering—from basic shape construction to complex shader programming—providing a comprehensive learning experience.

1. **User Navigation and Interaction**

* **Camera Control**: Users navigate the scene using keyboard and mouse inputs. Camera movements are managed by functions like *ProcessKeyboardEvents()* for keyboard inputs and *Mouse\_Position\_Callback()* for mouse movements, allowing users to explore the scene from different angles.
* W key: Moves the camera forward.
* S key: Moves the camera backward.
* A key: Moves the camera to the left (panning).
* D key: Moves the camera to the right (panning).
* Q key: Moves the camera upward.
* E key: Moves the camera downward.
* **Scene Interactivity**: By adjusting the camera view, users can focus on details like the textures of the vase or the shine on the hard drive and the candle , enhancing their interactive experience within the 3D environment.

1. **Custom Functions for Modularity and Reusability**

My 3D scene management code is structured to be modular and reusable, enhancing maintainability and scalability. This is achieved by encapsulating functionality into specific functions, each handling distinct aspects like lighting setup, texture management, and rendering configurations. For instance:

1. *CreateGLTexture(const char\* filename, std::string tag)*

* **Purpose**: Loads an image from a file and creates a texture object in OpenGL, which can be used for rendering surfaces with detailed imagery.
* **Functionality**:

Loads an image using *stbi\_load*, checking for successful loading.

Generates an OpenGL texture, setting parameters like wrapping and filtering.

Supports different image formats (RGB, RGBA) and configures the texture storage accordingly.

Stores the texture ID and a user-defined tag in a map for later retrieval.

* **Reusability:**

This function is highly reusable for any task that requires texture loading.

The ability to tag textures with strings allows for easy identification and use in different parts of the program, promoting code organization and preventing errors such as texture misapplication.

1. *BindGLTextures()*

* **Purpose**: Binds all loaded textures to respective texture units in OpenGL, preparing them for use in rendering.
* **Functionality**:

Iterates over all stored textures and binds each one to a texture unit starting from GL\_TEXTURE0.

This setup is crucial for shaders that sample multiple textures simultaneously.

* **Reusability:**

This function can be used in any rendering context where multiple textures need to be bound, making it generic and flexible for different rendering scenarios.

1. *SetShaderMaterial(std::string materialTag)*

* **Purpose**: Configures the shader's material properties based on predefined settings associated with a tag, facilitating dynamic visual effects.
* **Functionality**:

Looks up material properties (ambient, diffuse, specular colors, and shininess) by tag.

Sets these properties in the shader, influencing how the object interacts with light (reflectivity, glossiness, etc.).

* **Reusability**:

This function allows for easy changes to material properties of objects throughout the scene from a single point of control.

It can be reused in any rendering context that involves variable material properties, making it highly flexible and powerful for thematic adjustments in the scene.

1. **Summary**

Each function in the program is designed to handle a specific task, which helps keep the code organized and easy to manage. This approach breaks down complex operations into smaller, straightforward tasks, making the overall rendering process smoother and improving various parts of the program. As a result, the code becomes clearer and easier to follow, which reduces the chance of making mistakes. It also makes it easier to update and maintain the software over time.

# References

Group, K. (2024). *Avoiding 16 Common OpenGL Pitfalls*. Retrieved from OpenGL: https://www.opengl.org/archives/resources/features/KilgardTechniques/oglpitfall/

w3shools.com. (2024). *https://www.w3schools.com/*. Retrieved from https://www.w3schools.com/: https://www.w3schools.com/mongodb/mongodb\_mongosh\_update.php