

# M<sub>inimal</sub>YST

CS-330-T4204 Comp Graphic and Visualization 21EW4

Ryan DeBaal



## Introduction

For my final project, I created a minimalist reproduction of a scene from the seminal point and click adventure game *Myst* which was originally released in 1993, released *again* in full 3D as *realMyst* in 2000, and released *again* as a VR game for the Oculus Quest in 2020.



Initially, I was going to attempt to recreate this scene as realistically as possible. I knew it was going to be time consuming, but I didn't expect it to be especially difficult since the 3D models which were captured for the original game are simple meshes from almost 30 years ago.



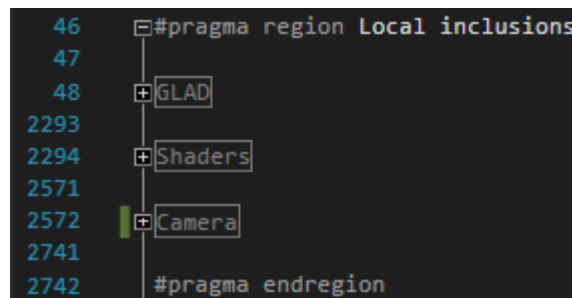
However, as the course proceeded I began to realize that my initial plan was overly ambitious and that I would have to scale back the complexity of my execution into order to focus on getting a semblance of the scene in place.

## Justification

The most time-consuming portion of the work came in Week 6, when our assignment was to submit our work using a shader which supports multiple light sources. Until that point, I had been using a shader that was defined inline and could only support for a single light source. This necessitated almost a complete rewrite of my project since the entire graphics pipeline was suddenly invalid. I chose to use the Open GL Sample as base and proceeded to migrate my hand-crafted VOAs over to the new paradigm.

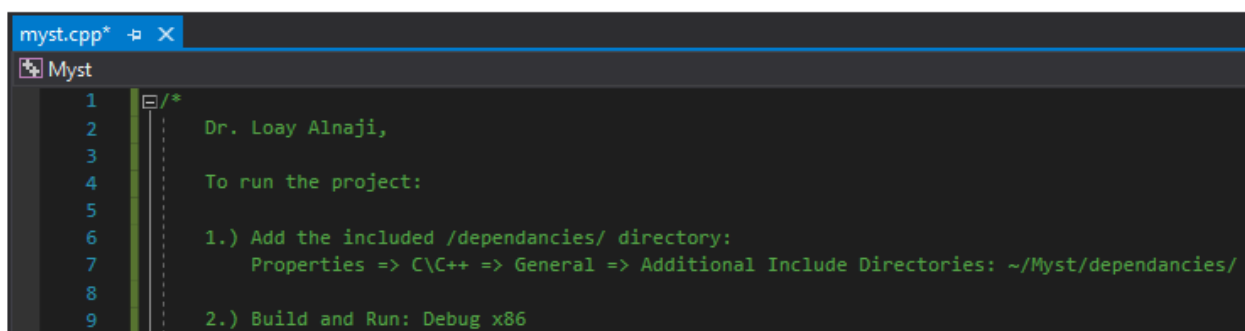
Once I was able to render a scene which supported multiple lights, I decided that I should focus on the primitives I had already working: Cube and Pyramid. This is when I decided that a minimalist approach harkening back to Minecraft would be the simplest approach which provided the greatest net positive.

I chose to merge the Shader class, Camera class, and contents of glad.c into my primary source code file because I was having reoccurring issues with the dependencies. I regret having to make this modification, as it creates a massive influx of almost 3000 lines of code, this made intellisense and navigation more difficult due to the sheer volume of code. I would have preferred to keep each of these functionalities isolated but I simply did not have the requisite familiarity with C++ to be able to manage a complex project *and* make meaningful progress on creating my scene.



```
46  #pragma region Local inclusions
47
48  #include <GLAD>
2293
2294  #include <Shaders>
2571
2572  #include <Camera>
2741
2742  #pragma endregion
```

I also chose to move all the external dependencies into a folder underneath the root directory. This allowed me to link to all the “Additional Include Directories” locally which I hope will make it much easier for another user to get the project up and running.



```
myst.cpp*  X
Myst
1  /*
2  Dr. Loay Alnaji,
3
4  To run the project:
5
6  1.) Add the included /dependancies/ directory:
7      Properties => C\C++ => General => Additional Include Directories: ~/Myst/dependancies/
8
9  2.) Build and Run: Debug x86
```

## Navigation

To navigate the scene, the “player” uses the mouse and keyboard. The WASD keys control your direction, and the mouse allows you to turn and look around the scene. While working on the movement controls I realized that “strafing” from side to side felt sluggish compared to the other first-person games I play. I added a StrafeModifier variable to tweak movement along the X-Axis to make the movement more in line with what gamers would be most accustomed with.

## Organization

I created a class called Object3D to store some of the transformation matrices information about primitives in my scene. This allowed me to create arrays of objects which could then be iterated over to draw the scene based on the currently loaded mesh.

```
//Stores the data relative to an object in 3D space
const struct Object3D {
    glm::vec3 position;
    glm::vec3 scale;
    string texture;
};
```

I also implemented a map collection to associate texture `GLuint` identifiers to a string so that I could easily retrieve any loaded `GL_TEXTURE_2D` by string and apply different textures to different primitives.

```
//Texture variables
map<string, GLuint> textureMap;

void LoadTextures() {
    //Load textures into map for easy reference
    textureMap.insert({ "container", LoadTextureFile("images/container2.png") });
    textureMap.insert({ "granite", LoadTextureFile("images/granite.jpg") });
    textureMap.insert({ "concrete", LoadTextureFile("images/concrete-2.jpg") });
    textureMap.insert({ "column", LoadTextureFile("images/column.jpg") });
    textureMap.insert({ "marble", LoadTextureFile("images/marble.jpg") });
    textureMap.insert({ "grass", LoadTextureFile("images/grass.jpg") });
    textureMap.insert({ "water", LoadTextureFile("images/water.jpg") });
    textureMap.insert({ "travertine", LoadTextureFile("images/travertine.jpg") });
    textureMap.insert({ "moon", LoadTextureFile("images/moon.jpg") });
    textureMap.insert({ "white", LoadTextureFile("images/white.jpg") });
    textureMap.insert({ "clouds", LoadTextureFile("images/skybox-2.jpg") });
}

glBindTexture(GL_TEXTURE_2D, textureMap.find(skybox.texture)->second);
```

I used standard C++ naming conventions and grammar and tried to organize my code as logically as possible so that I could locate different functionalities throughout the development of the project.

## Conclusion

This was a challenging project which required a significant dedication to testing and time management. I am happy with the results I was able to create. Over the duration of this course, I have garnered greater respect for developers who write 3D applications. I often take these functionalities for granted because the games industry has been iterating for decades. I can see now why more and more games are created with Unreal, Unity, CryTek, and other popular commercial 3D engines instead of relying on homebrewed solutions.

I found this to be an incredible rewarding experience and plan to continue working on my C++ and Open GL skillset after this course is over.