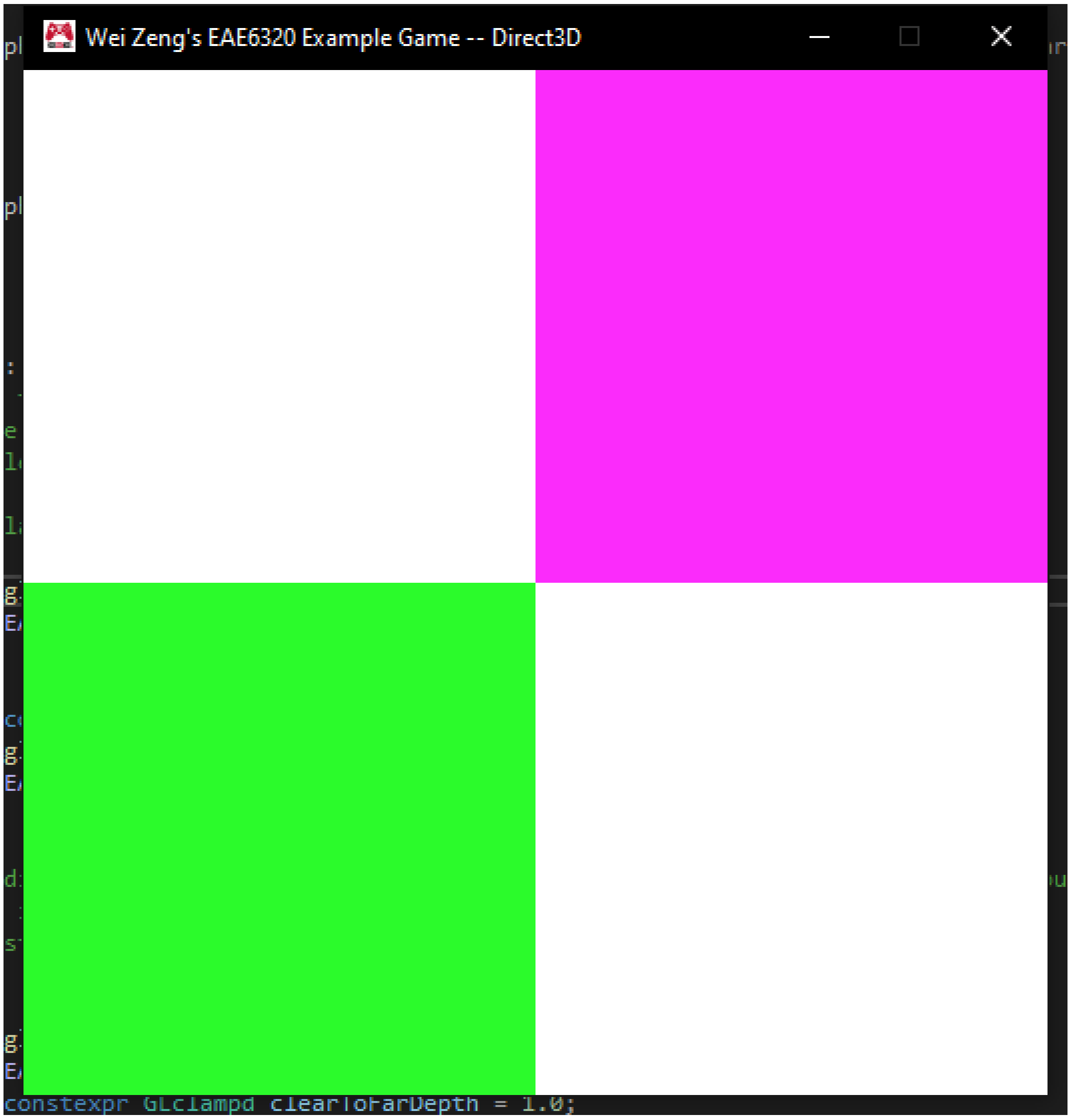
Assignment 3 Writeup

Game Executable:

https://github.com/WayGold/EAE6320_Assignments/blob/Assignment03/MyGame_.zip

Game running:



```
constexpr GLclampd clearToFarDepth = 1.0;
```

Explain how you made Graphics.cpp platform-independent:

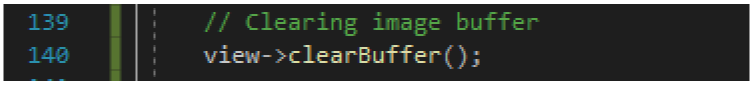
- **What new interfaces did you have to create?**

I ended up creating an extra interface for view related components and another interface for clearing and updating buffers.

- **Where did you decide to declare these interfaces? Where did you define the platform-specific implementations for these interfaces?**

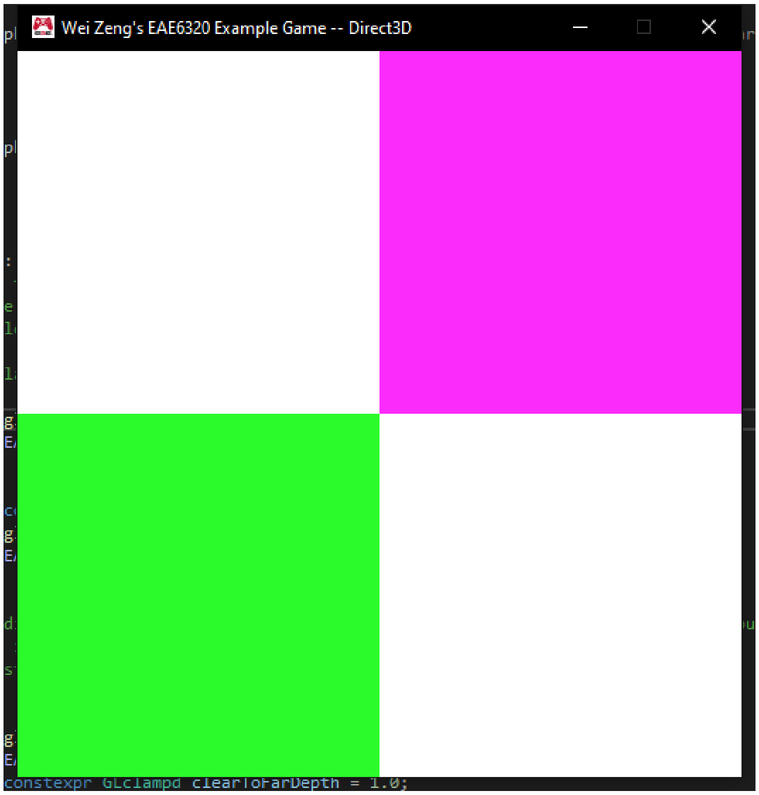
  Interfaces are located at the same level as Graphics.h and all the platform specific implementations go into corresponding directories.

- **Show us your code in the Graphics.cpp file that clears the back buffer color**



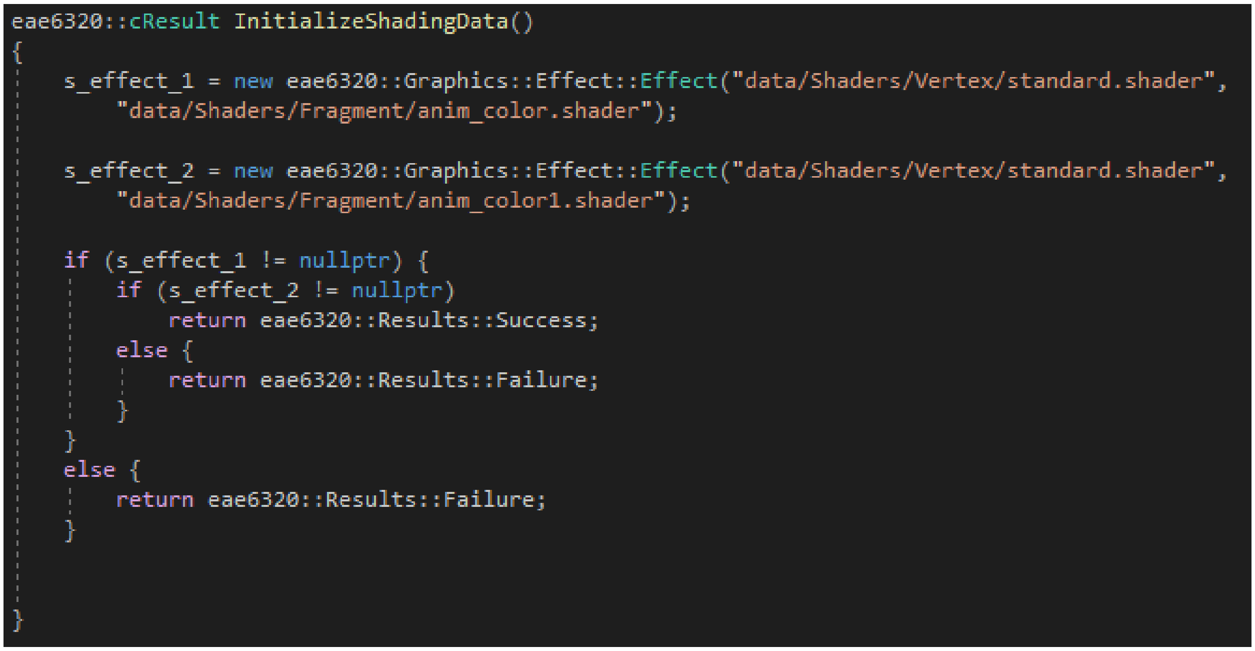
```
// Clearing image buffer
view->clearBuffer();
```

- **Show a screenshot of your game with a clear color (i.e. background) other than black**



```
constexpr GLclampd clearToFarDepth = 1.0;
```

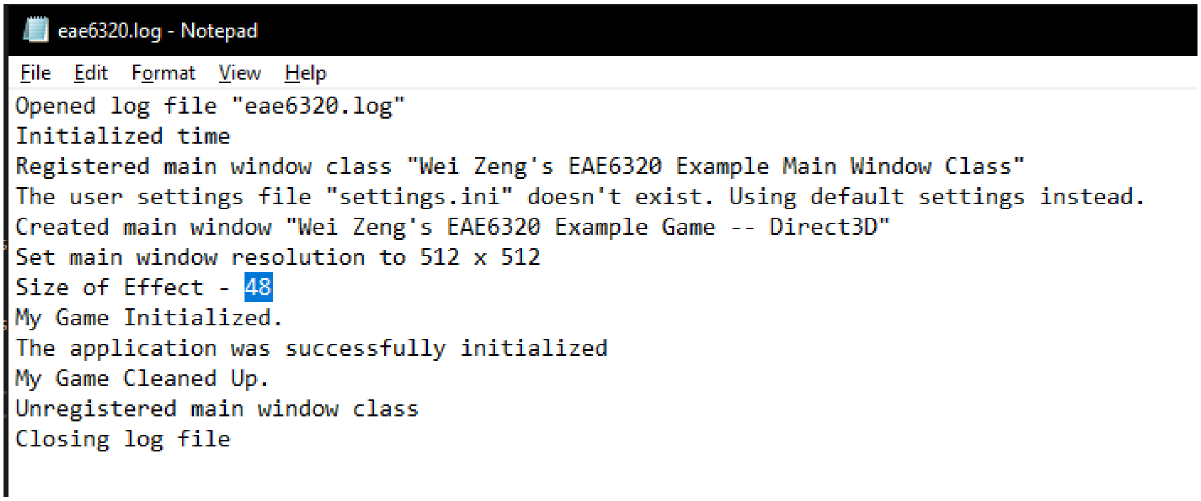
- **Show code from your Graphics.cpp file that initializes an effect. Explain what data the user is required to specify**



```
eae6320::cResult InitializeShadingData()
{
    s_effect_1 = new eae6320::Graphics::Effect::Effect("data/Shaders/Vertex/standard.shader",
        "data/Shaders/Fragment/anim_color.shader");

    s_effect_2 = new eae6320::Graphics::Effect::Effect("data/Shaders/Vertex/standard.shader",
        "data/Shaders/Fragment/anim_color1.shader");

    if (s_effect_1 != nullptr) {
        if (s_effect_2 != nullptr)
            return eae6320::Results::Success;
        else {
            return eae6320::Results::Failure;
        }
    }
    else {
        return eae6320::Results::Failure;
    }


}
```

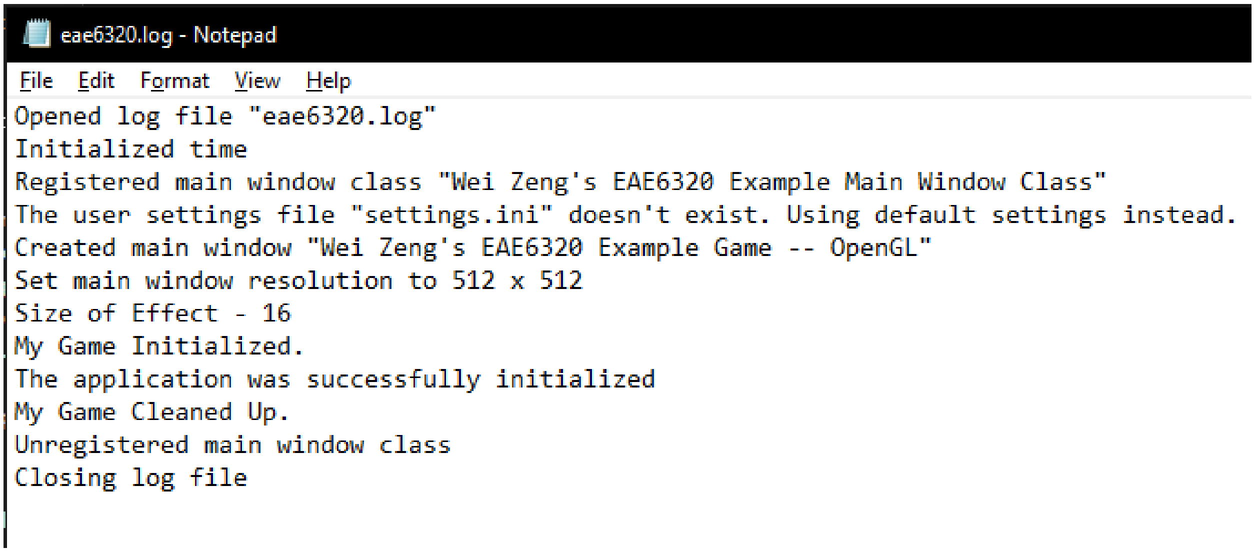
Users need to pass in the path to the vertex shader and the fragment shader for the effect.

- **Effect Memory Size**



```
Opened log file "eae6320.log"
Initialized time
Registered main window class "Wei Zeng's EAE6320 Example Main Window Class"
The user settings file "settings.ini" doesn't exist. Using default settings instead.
Created main window "Wei Zeng's EAE6320 Example Game -- Direct3D"
Set main window resolution to 512 x 512
Size of Effect - 48
My Game Initialized.
The application was successfully initialized
My Game Cleaned Up.
Unregistered main window class
Closing log file
```

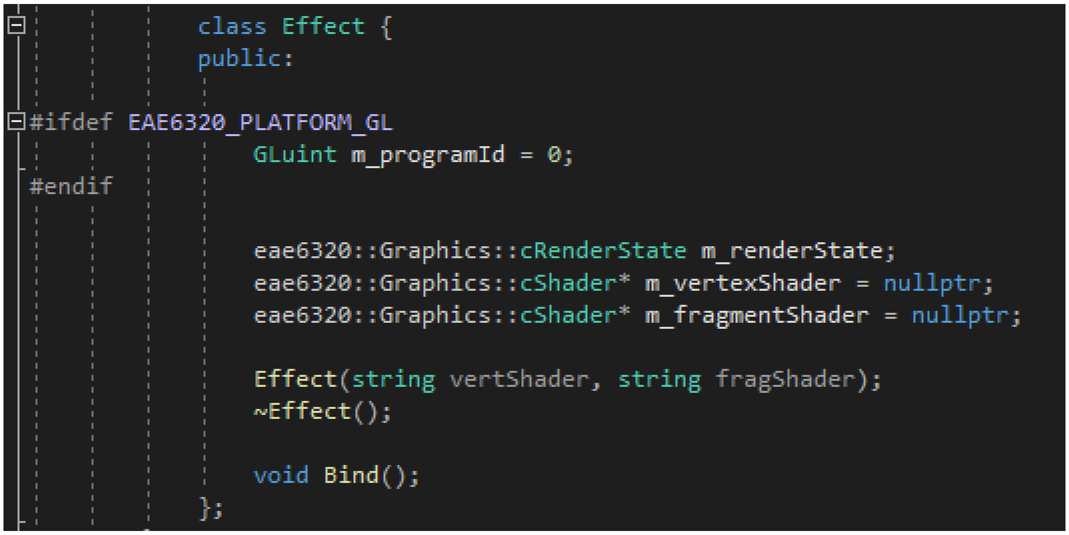
(D3D Effect Size Log)



```
Opened log file "eae6320.log"
Initialized time
Registered main window class "Wei Zeng's EAE6320 Example Main Window Class"
The user settings file "settings.ini" doesn't exist. Using default settings instead.
Created main window "Wei Zeng's EAE6320 Example Game -- OpenGL"
Set main window resolution to 512 x 512
Size of Effect - 16
My Game Initialized.
The application was successfully initialized
My Game Cleaned Up.
Unregistered main window class
Closing log file
```

(GL Effect Size Log)

The size is different for the two platforms because there are lots of platform dependent data.

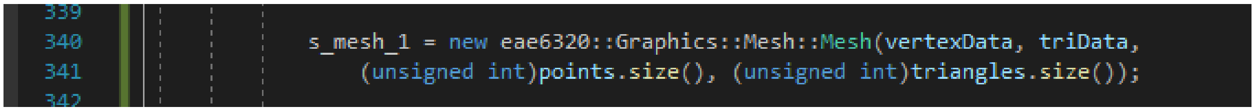


```
            class Effect {
            public:

#ifdef EAE6320_PLATFORM_GL
                GLuint m_programId = 0;
#endif

                eae6320::Graphics::cRenderState m_renderState;
                eae6320::Graphics::cShader* m_vertexShader = nullptr;
                eae6320::Graphics::cShader* m_fragmentShader = nullptr;

                Effect(string vertShader, string fragShader);
                ~Effect();

                void Bind();
            };
```

Assuming that VS C++ is using 8 byte alignment by default. In x86 cRenderState is 4 byte and in x64 cRenderState is 32 byte. Therefore no matter how you arrange these orders you could only get the same best usage.

X86 = 4 + 4 + 4 + 4; x64 = 32 + 8 + 8

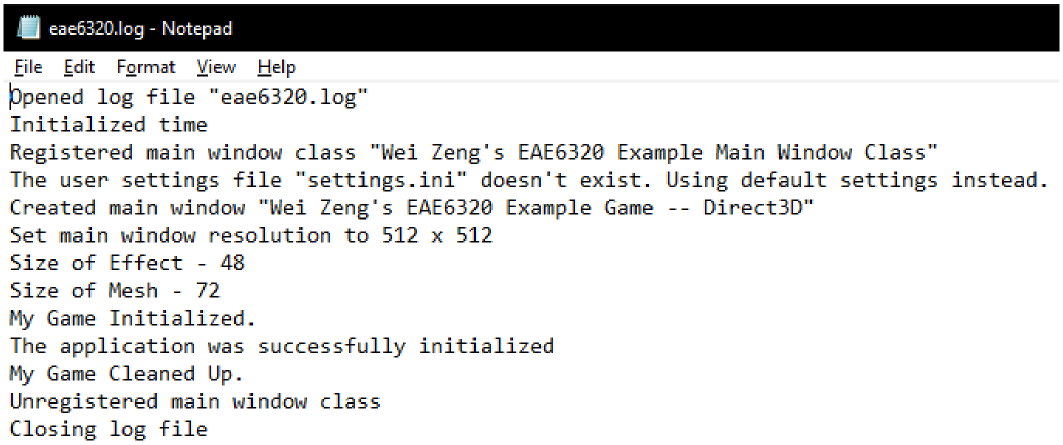
- **Show code from your Graphics.cpp file that initializes a mesh. Explain what data the user is required to specify**



```
			s_mesh_1 = new eae6320::Graphics::Mesh::Mesh(vertexData, triData,
				(unsigned int)points.size(), (unsigned int)triangles.size());
```

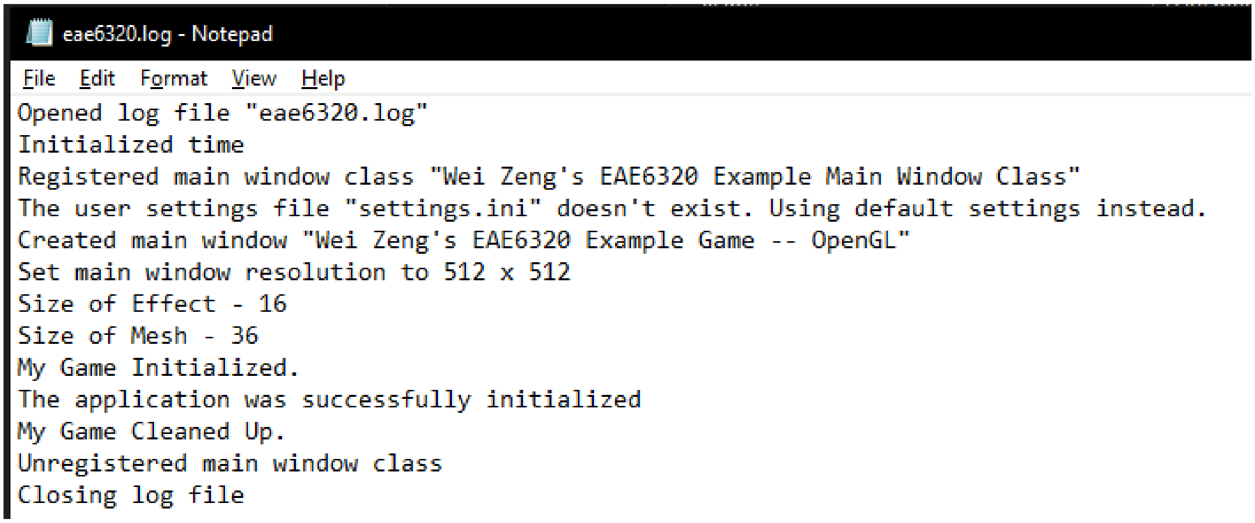
Users need to pass in an array of all the vertex points needed, an array of vertex indexes that represents the triangles, the size of the vertex array and lastly the size of the triangle index array.

- **Mesh Memory Size**



```
Opened log file "eae6320.log"
Initialized time
Registered main window class "Wei Zeng's EAE6320 Example Main Window Class"
The user settings file "settings.ini" doesn't exist. Using default settings instead.
Created main window "Wei Zeng's EAE6320 Example Game -- Direct3D"
Set main window resolution to 512 x 512
Size of Effect - 48
Size of Mesh - 72
My Game Initialized.
The application was successfully initialized
My Game Cleaned Up.
Unregistered main window class
Closing log file
```

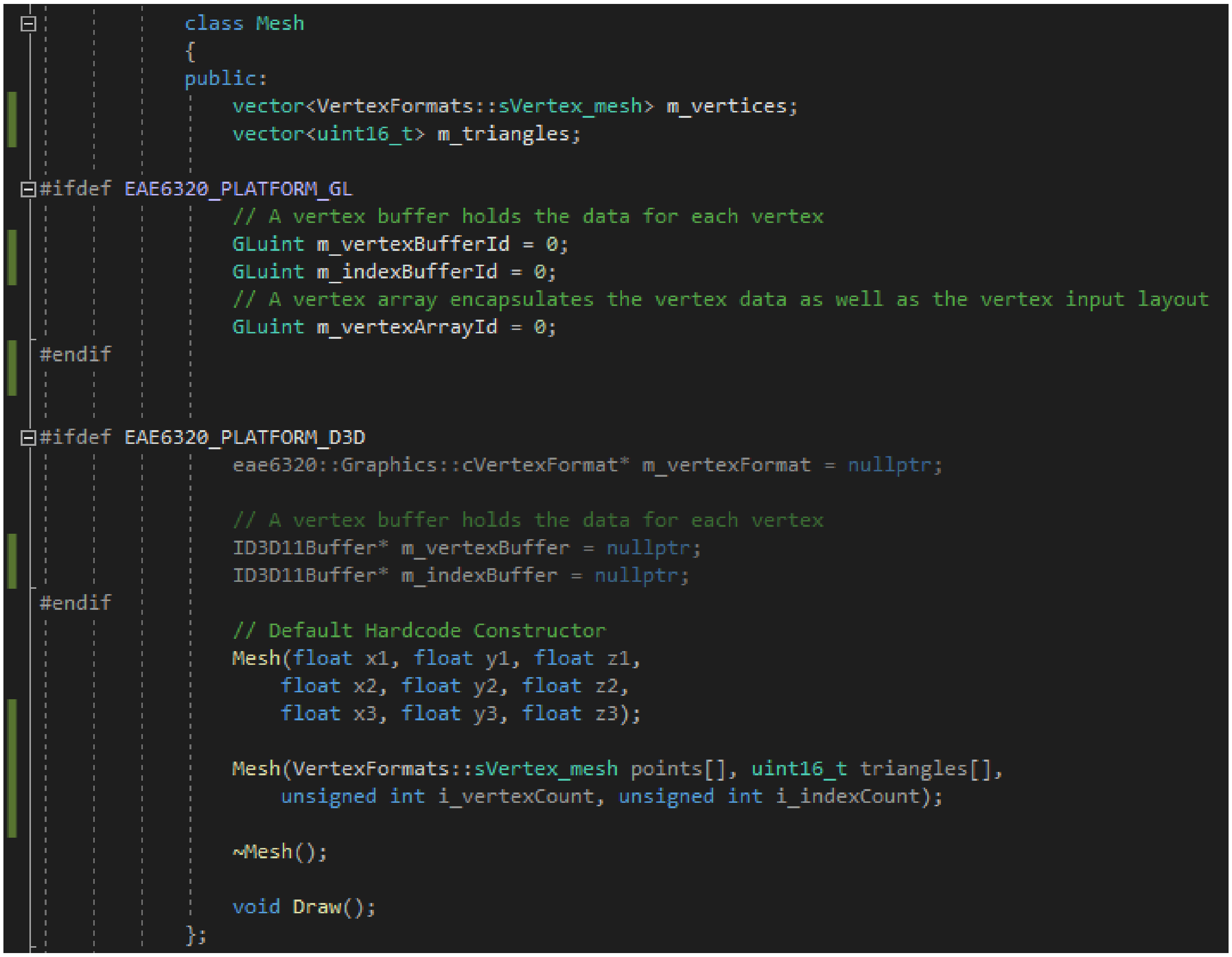
(D3D Mesh Size Log)



```
Opened log file "eae6320.log"
Initialized time
Registered main window class "Wei Zeng's EAE6320 Example Main Window Class"
The user settings file "settings.ini" doesn't exist. Using default settings instead.
Created main window "Wei Zeng's EAE6320 Example Game -- OpenGL"
Set main window resolution to 512 x 512
Size of Effect - 16
Size of Mesh - 36
My Game Initialized.
The application was successfully initialized
My Game Cleaned Up.
Unregistered main window class
Closing log file
```

(GL Mesh Size Log)

The size is different for the two platforms because there are lots of platform dependent data.



```
            class Mesh
            {
            public:
                vector<VertexFormats::sVertex_mesh> m_vertices;
                vector<uint16_t> m_triangles;

#ifdef EAE6320_PLATFORM_GL
                // A vertex buffer holds the data for each vertex
                GLuint m_vertexBufferId = 0;
                GLuint m_indexBufferId = 0;
                // A vertex array encapsulates the vertex data as well as the vertex input layout
                GLuint m_vertexArrayId = 0;
#endif


#ifdef EAE6320_PLATFORM_D3D
                eae6320::Graphics::cVertexFormat* m_vertexFormat = nullptr;

                // A vertex buffer holds the data for each vertex
                ID3D11Buffer* m_vertexBuffer = nullptr;
                ID3D11Buffer* m_indexBuffer = nullptr;
#endif

                // Default Hardcode Constructor
                Mesh(float x1, float y1, float z1,
                    float x2, float y2, float z2,
                    float x3, float y3, float z3);

                Mesh(VertexFormats::sVertex_mesh points[], uint16_t triangles[],
                    unsigned int i_vertexCount, unsigned int i_indexCount);

                ~Mesh();

                void Draw();
            };
```

There is potential in making this smaller since I have two vectors to keep track of the input arrays and their sizes. Each vector would consume 12/24 bytes depending on the platform. For the current code I just need to get the size of the vectors to calculate buffer sizes. If not considering future extension, I could have had two int to store the sizes which could reduce lots of space.