Write Up

Assignment 02 - GAMES6320

[Download Game]

In this second assignment, I learned a lot about game engine architecture and how to design effective and modular systems. As I said in my previous write-up, the solution given to us was fairly readable but also felt overwhelming, with each file containing many lines of code. In this assignment, I had the opportunity to dig into the Graphics.cpp files and separate out some sections of code to have a better interface.

Creating the Mesh Class

While creating the mesh class, I first had to read into how each platform handled geometry data and drawing on to the screen. During this analysis of the Graphics.d3d.cpp and Graphics.gl.cpp files, I realized that the two platforms had very different implementations of initialization, clean-up, and drawing. They also used different types of variables for the geometry data.

Due to these reasons, I decided to have just two platform-specific CPP files for my Mesh class (cMesh.d3d.cpp & cMesh.gl.cpp). Since there was no real implementation being shared by the two platforms, it did not feel effective to include a platform-independent cMesh.cpp file for this class.

Hence, I only had a simple cMesh.h file without defining any member variables, and let the platform-independent files handle the initialization of the geometry data, which was different for Direct3D and OpenGL. Since the implementation in both platforms did not share any code, I stuck with having only three methods for this class: InitializeGeometry(), CleanUpGeometry(), and DrawGeometry().

Although I believe this is a good starting point, I can definitely improve this design going forward, by including more functions to handle specific things, and making sure it's a cleaner interface to use for meshes in the engine.

Creating the Effect Class

Although I thought that creating the Effect class would be a similar approach to the Mesh, it was a completely different ball game where I had to make some interesting decisions. First of all, I found that the two platforms shared a lot of code in the implementations of Initializing and Cleaning Up shading data. Moreover, the shading data itself contains pretty much the same variables, with just the addition of s_programId for the OpenGL platform.

For this reason, I wanted to define the member variables in the cEffect.h file itself, so I used #Define in this case, as it seemed like the most effective way to run the correct code for each platform. Also, the Effect class definitely needed more than just the three functions, as a lot of code was shared between platforms.

So, I split the implementation of the shading data into five different functions, and whatever code I found to be ‘common’ between the two platforms, I put those in functions to be implemented by the platform-independent cEffect.cpp file. The rest of the code, which was only required by one platform, was then put into the platform-specific cEffect.d3d.cpp and cEffect.gl.cpp files.

Analyzing the code and splitting it into different parts made me realize the importance of having clean interfaces to avoid duplication, especially in games where we often develop for multiple platforms and configurations.

```
// Bind the shading data
{
    // Binding the shading data from Effect Class
    newEffect.BindShadingData();
}
// Draw the geometry
{
    // Drawing the geometry from Mesh Class
    newMesh.DrawGeometry();
}
```

```
eae6320::cResult InitializeGeometry()
{
    auto result = eae6320::Results::Success;

    // Initialize Geometry from Mesh Class
    result = newMesh.InitializeGeometry();
    return result;
}

eae6320::cResult InitializeShadingData()
{
    auto result = eae6320::Results::Success;

    // Initialize Shading Data from Effect Class
    result = newEffect.LoadShaderFiles();

    result = newEffect.InitializeShadingData(result);

    return result;
}
```

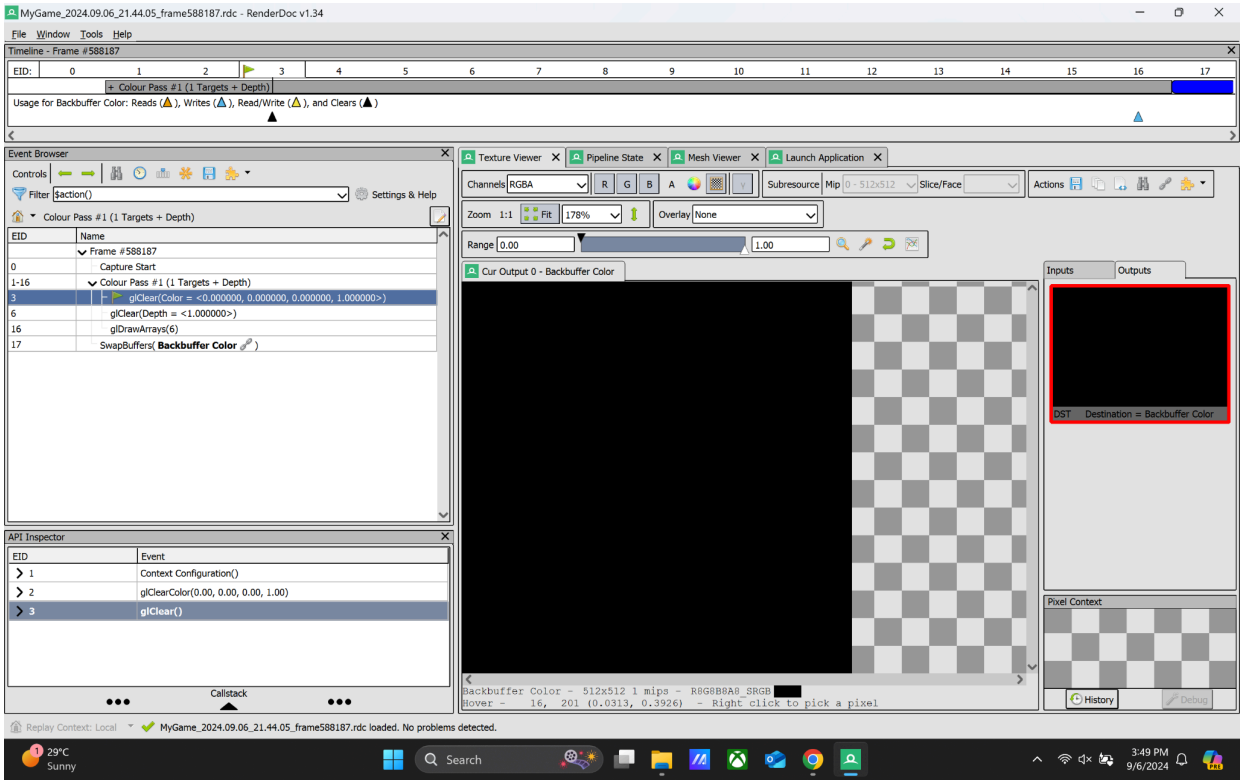
```
// Cleaning Up Geometry
newMesh.CleanUpGeometry();

// Cleaning Up Shading Data
newEffect.CleanUpShaderFiles();

newEffect.DeleteProgram(result);
```

GPU Captures

Visual Studio Graphics Analyzer

I could not get the Graphics Analyzer working, it just freezes on me :(

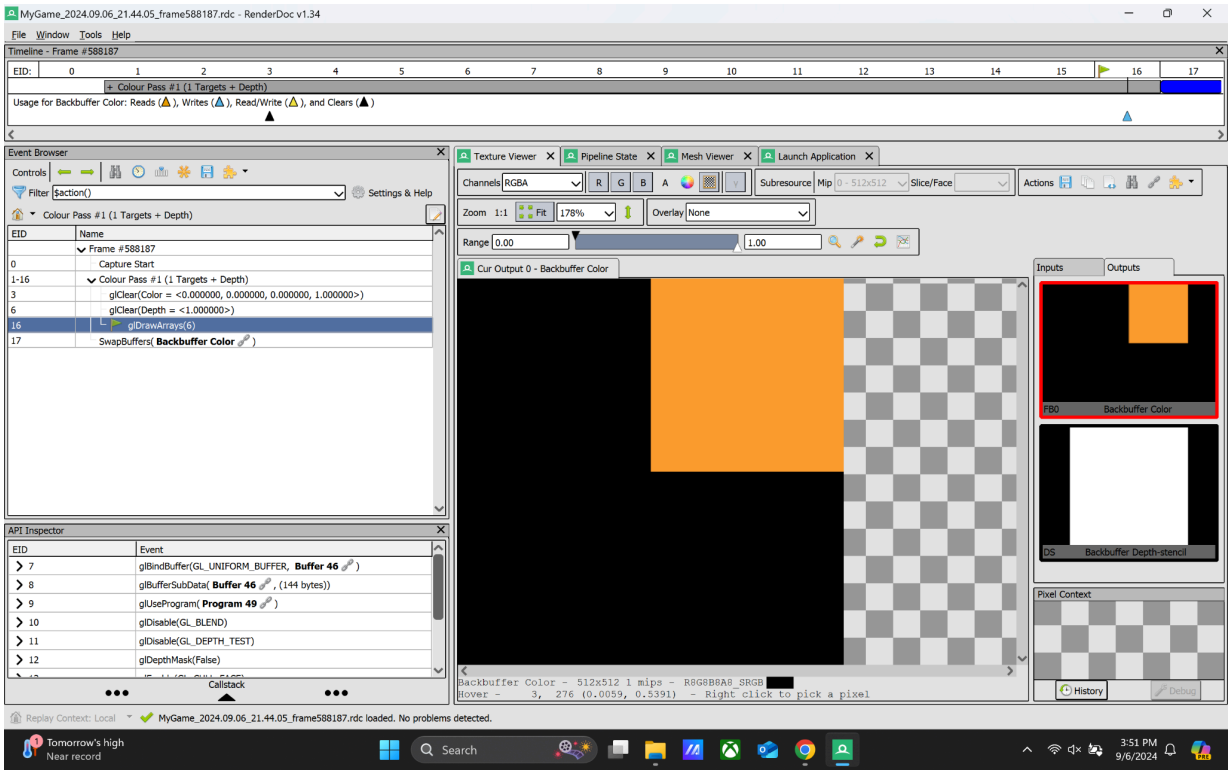
Render Doc

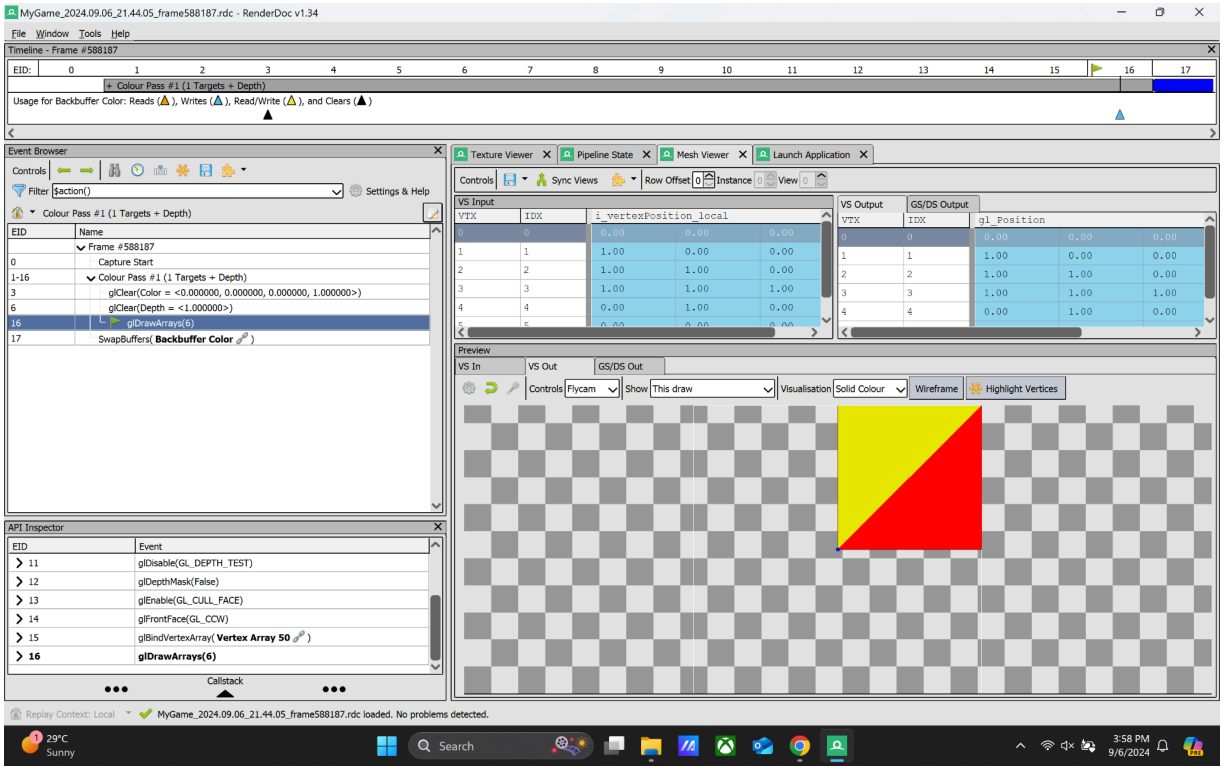


(More Screenshots can be found on Next Page)

Thinking Forward

Apart from the representation of Mesh and Effect, the Graphics.d3d.cpp and Graphics.gl.cpp files do share code but also have their own differences. For example, the initialization of platform-independent graphics objects, events, and views can definitely be converted into a cleaner interface with just the platform-independent Graphics.cpp file. Additionally, the RenderFrame() function also contains implementation that can be refactored going forward.





[Download Game]