# 1. What is a game engine?

A game engine is not only used to make a game, but it can also be used to do a lot more things. It can create VR, scientific visualization, and so forth.

A game engine doesn’t create data, but loads existing files (data, or virtual file) from disk, we call it asset. They maybe make the texture in Photoshop, or 3d model in Maya, and then convert it to something that game engine can read, like jpeg for picture or obj for 3d model.

To build the game engine, we need many systems, like platform abstraction layer (if the game engine is cross platform), rendering, audio.

The aim of the game engine is that, for someone who don’t familiar with programming to create the data and visualize it. For example, Unity, you can use it to create a minimal game with minimal script.

# 2. Design the game engine

We are going to design a simple game engine with ImGUI, not a window application. We need:

1. Entry point: how to launch the main function

2. Application layer: deal with life cycle

3. Window layer: Input, event manager

4. Render: biggest system. Most of the game engine start with renderer immediately. But we will start with a debugging layer, and all those infrastructure, and that could make the renderer building more easily.

5. Render API abstraction. Our system should support multiple graphic API (OpenGL, Direct X, Vulcan), we need to add an additional layer on the top of the basic graphic API.

6. Debugging support.

7. Scripting language: support Lua, C#

8. Memory system: a necessary to manage memory

9. Entity component system (ECS): crate game object, and model it.

10. File I/O, Virtual file system (VFS)

11. Build system: connect to Photoshop to adjust the vertex of the model.

For now, this project only support Windows and OpenGL

# 3. Project setup

We will create a dll, and link all of our work into one dll, and make our game based on the dll. Instead of statically link all of our dependencies into the game.

First, remove the x86, it doesn’t support 32-bit system.

A screenshot of a computer

Description automatically generated

For all configurations option, we change the output to dll.

A screenshot of a computer

Description automatically generated with medium confidence

Change the output folder, and the intermediate folder

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font, line

Description automatically generated

Now we had our engine, but we need to add out application to use the game engine

A screenshot of a computer

Description automatically generated

Copy paste all the setting from the game engine to the sandbox, and the sandbox is an application

A screenshot of a computer

Description automatically generated with medium confidence

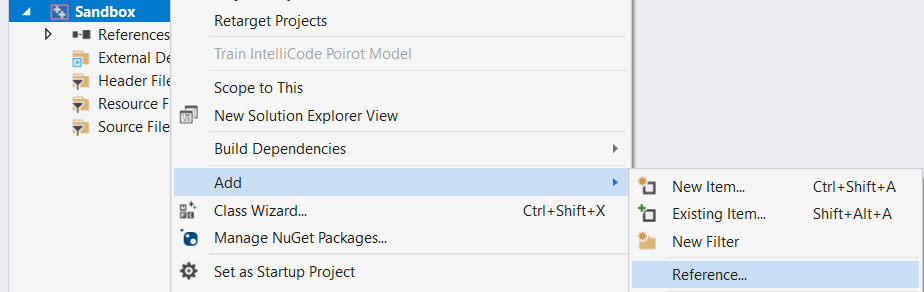
Also, we need to set the sandbox as the startup project.

Open GameEngin.sln and shift the order of the two projects.

A screenshot of a computer

Description automatically generated with medium confidence

Link this two projects

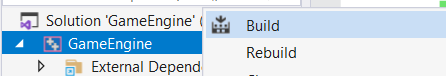


When the game engine generate a dll file, the sandbox will link to that file

Crate dll function, and build

A screenshot of a computer

Description automatically generated with medium confidence



Now we have dll and lib file

A screenshot of a computer

Description automatically generated

Then, go to the sandbox and crate the application, and import the dll and build

A screenshot of a computer program

Description automatically generated with medium confidence

We can copy paster the dll file manually from the game engine output folder, to the sandbox output folder, to run this project

A screenshot of a computer

Description automatically generated with medium confidence

# 4. Entry point

Each library must have an entry point, even dynamic library. Out application use dll code from game engine, so we need to attach or detach that code somewhere.

Here, the main function is inside the application, but we want to control it from the engine side

A screenshot of a computer program

Description automatically generated with medium confidence

Define the macro, and add the macro to the preprocessor.

A screenshot of a computer program

Description automatically generated with medium confidence

In Game Engine, add the macro of GE\_BUILD\_DLL and GE\_PLATFORM\_WiNDOWS. In application, add the macro of GE\_PLATFORM\_WiNDOWS.

And the application in Game Engine use that API

A picture containing text, screenshot, font, number

Description automatically generated

Add the include file

A screenshot of a computer

Description automatically generated with medium confidence

Also add the include directory in the sandbox

A screenshot of a computer

Description automatically generated

In the application in Game engine, define the application creator function, but implement this function is the sandbox.

A picture containing text, font, graphics, white

Description automatically generated

In the sandbox, there is no main function.

A screen shot of a computer code

Description automatically generated with low confidence

# 5. Logging

A system to log (print) event. We will use SPD log library. <https://github.com/gabime/spdlog>

The license show that we need to include this license to our project. We don’t need to have the whole code base, but add it as a submodule of our project.



Git will download it and create a .gitmodules file

A screenshot of a computer

Description automatically generated with medium confidence

Add the directory to the game engine and the sandbox project

A screenshot of a computer

Description automatically generated with medium confidence

We will create a middle layer class to wrap up the spd library.

Create the log file

A screen shot of a computer program

Description automatically generated with low confidence

Define macro

A screenshot of a computer program

Description automatically generated with low confidence

Init log

A screenshot of a computer code

Description automatically generated with medium confidence

# 6. Premake

Right now, our build toolset is VS on windows, but we want to create a cross platform system.

We will not use Cmake, because Cmake is extremely complicate. We will use Lua to create premake.

Download Premake from github: <https://github.com/premake/premake-core/releases/tag/v5.0.0-alpha13>

Copy the exe file and paste into the project, make a new folder

A screenshot of a computer

Description automatically generated with medium confidence

This vendor in the root directory of the project is different from the vendor in the Game engine folder, it covers the whole project.

Write the premake script as premake5.lua.

A picture containing text, screenshot, font, number

Description automatically generated

Write a batch file for this.

A screenshot of a computer

Description automatically generated with medium confidence

Then, you may delete GameEngine.sln, and .vcxproj, .vcxproj.filters, .vcxproj.user file, and click the batch file to generate those file by premake.

# 7. Event system

Handle with window input, for example, resize the window, mouse move, mouse click.

The application create the window class, and the windows class will finally draw triangle on the screen. When event happened, the window class will use the callback provided by the application.

Event type:

A screenshot of a computer program

Description automatically generated with low confidence

Event group:

A screenshot of a computer program

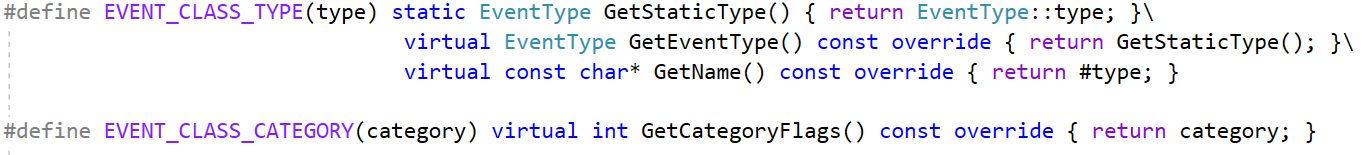
Description automatically generated with low confidence

Define event base class

A picture containing text, screenshot, font

Description automatically generated

Define macro to make the derived class easy



Define event dispatcher

A picture containing text, screenshot, font

Description automatically generated

Also the operator for output

A picture containing text, screenshot, font, line

Description automatically generated

For all of the other types of event, inherit this base class

A screenshot of a computer program

Description automatically generated with low confidence

# 8. Precompiled header

A screenshot of a computer

Description automatically generated

Include everything we need in one single file, and add it to the premake

A picture containing text, font, line

Description automatically generated

Now, you may delete all the relevant library:

A screenshot of a computer

Description automatically generated with medium confidence

For each .cpp file, you can see that it use the precompiled header

A screenshot of a computer

Description automatically generated with medium confidence

Mind that, you also need to include this precompiled header into your .cpp files:

1. make sure that all the .cpp files include this precompiled header “gepch.h”

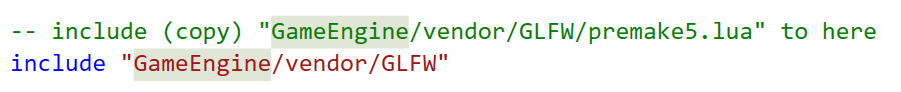
2. #include "gepch.h" must be on the top of all the included libraries.

3. you don’t need to include it in the header file, because it is already been precompiled.

# 9. Window abstraction

Currently, we will use GLFW to create the window. But GLFW doesn’t support DirectX. Later, we may use win32 because it supports DirectX. We also use git submodule to add GLFW to our project. <https://github.com/thecherno/glfw>





A picture containing text, font, screenshot

Description automatically generated

Define a window structure

A picture containing text, screenshot, font, number

Description automatically generated

Define the window class

A screen shot of a computer program

Description automatically generated with low confidence

You may have two problems when following this episode:

1. lots of unresolved external symbols

2. unresolved external symbols DwmExtendFrameIntoClientArea

1. For the first problem, you may notice that in GLFW project setting, go to C/C++ -> Code Generation, on the right hand side, the runtime library is Multi-threaded DLL (/MD), that means when compile this project, GLFW will use C++ dynamic library. And in the Hazel project setting, the runtime library however is Multi-threaded (/MT), that means when compile this project, Hazel will use static links of C++ runtime. Now we can have 4 combinations, and all of them are valid except GLFW use (/MD) and Hazel use (/MT).

Therefore, you can have GLFW use (/MD) and Hazel use (/MD), or GLFW use (/MT) and Hazel use (/MD), or GLFW use (/MT) and Hazel use (/MT)

Here, it is better to set this options in premake5.lua file, however I didn't find any method to change that setting. You can set it in Visual studio directly. And don't forget to set it each time after you regenerate the solution .sln file.

2. For the second problem, I guess that the new version of GLFW use this new function, and this function is in dwmapi.lib. Therefore, add this dependency into premake5.lua.

links

{

"GLFW",

"opengl32.lib",

"dwmapi.lib"

}

And regenerate the solution. Don't forget to change the C++ runtime setting.

# 10. Window event

Set event callback for GLFW

A picture containing text, font, line, screenshot

Description automatically generated

Each time when event happened, this function will be called. It will get the pointer of the window, and create a event class

Set event callback in application.

A picture containing text, font, line, screenshot

Description automatically generated

In C++, I have such a function in C++: void Func(std::function<void(int&)> f) { f(); } The input of this function is std::function<void(int&)>

Now I want to call this function, and I have such a function as input void fun(int& t) { printf("this is the input function"); } I find that Func(fun); is wrong. The compiler said that no suitable convert from "void (int&)" to "std::function<void(int&)>"

But, if the input is a labmda function, then it works: Func([](int& t) {printf("this is the input function");});

ChatGPT:

The reason you're encountering an error when calling Func(fun) is because the type of fun is a regular function pointer (void (\*)(int&)), not an std::function<void(int&)>. Although a function pointer can be implicitly converted to an std::function type when the signature matches, it doesn't happen automatically in this case.

When you use a lambda function, it works because a lambda function creates a closure object that can be implicitly converted to std::function. The lambda function you provided has the correct signature (void(int&)), so it can be converted to std::function<void(int&)> without any issues.

If you want to use a regular function like fun without modifying its type, you can either explicitly cast it to std::function<void(int&)> or use std::bind to create a callable object with the desired signature.

Using std::bind would look like this: Func(std::bind(fun, std::placeholders::\_1)). std::bind creates a callable object that "binds" the function fun with the placeholder argument \_1. The placeholder indicates where the argument int& t will be passed when the bound function is called.

Alternatively, you can cast the function pointer to the desired type: Func(static\_cast<std::function<void(int&)>>(fun)). Here, the static\_cast explicitly converts the function pointer fun to an std::function<void(int&)> before passing it to Func.

Both of these approaches allow you to use the regular function fun as the argument to Func, but the lambda function and std::bind provide more flexibility and convenience when working with different function signatures.

OnEvent function

A picture containing text, screenshot, font, line

Description automatically generated

Use dispatcher to execute the event.

# 11. Layers

Like Photoshop, you can choose to turn on or turn off a layer to draw. Each layer will be update separately, like a layer stack. The layer should be rendered from bottom to top, while the events should be handled from top to bottom.

Add layer and layer stack

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

It use emplace to append elements:

A picture containing text, screenshot, font, line

Description automatically generated

emplace function have a better performance

MyClass obj(arg1, arg2);

vec.insert(vec.begin(), obj);

In this case, it will first create this object, and then copy it into the vector. However, if you already know the obj class, you can use emplace to avoid copy constructor:

vec.emplace(vec.begin(), arg1, arg2);

And emplace function is available from C++ 11, and must have the argument first.

Then add this layer function into the application.

A picture containing text, font, screenshot, line

Description automatically generated

Also, update the run function

A picture containing text, screenshot, font, line

Description automatically generated

Handle the event, the event should be handle from top to bottom.

A screen shot of a computer program

Description automatically generated with low confidence

Test layer in the sandbox

A screenshot of a computer program

Description automatically generated with medium confidence

# 12. Modern OpenGL (Glad)

We need to load OpenGL functions from GPU to C++ code. Instead of glew, we will use glad. Glad is more modern

Go to the website: <https://glad.dav1d.de/>, and generate a loader, and download the zip file

A screenshot of a computer

Description automatically generated with medium confidence

Write premake file

A picture containing text, screenshot, font

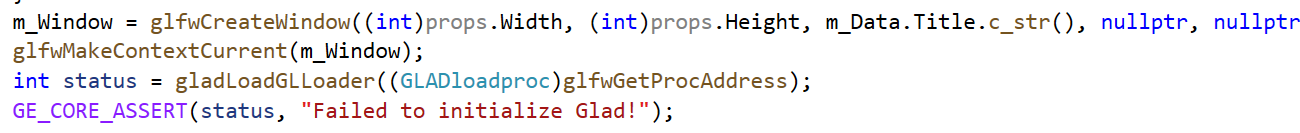
Description automatically generated

And include this premake file into the project, and regenerate the solution

A screenshot of a computer program

Description automatically generated with low confidence

Load Glad in WindowsWindow



You should include glfw after glad, or it will cause error: OpenGL header already included, remove this include, glad already provides it

To solve this error, you can find that in OpenGL:

A picture containing text, font, screenshot, white

Description automatically generated

A picture containing text, font, white

Description automatically generated

You should define GLFW\_INCLUDE\_NONE to make sure that glfw doesn’t include gl

Then, add this predefine in the premake5.lua, and regenerate the project

A black text on a white background

Description automatically generated with low confidence

Now, because glfw doesn’t include gl anymore, in application.cpp, glClearColor doesn’t exist. You need to change the #include <GLFW/glfw3.h> to #include <glad/glad.h>

A screenshot of a computer program

Description automatically generated with low confidence

# 13. ImGUI

Download and add it as a submodule of our project. <https://github.com/TheCherno/imgui>, use this premake file, but download ImGui newest version (or docking branch) from its own website.

git submodule add https://github.com/TheCherno/imgui GameEngine/vendor/imgui

Add it to the premake5.lua file, and regenerate the solution, don’t forget to include the backends of imgui to the include directory.

Create a ImGui layer and inherit Layer class

A picture containing text, screenshot, font

Description automatically generated

Check the example. For attach function:

A screen shot of a computer code

Description automatically generated with low confidence

For update function:

A picture containing text, screenshot, font

Description automatically generated

In Application.cpp, add OnAttach function call when push layer

A picture containing text, font, screenshot, line

Description automatically generated

And in sandbox, add this new ImGui layer.

A screen shot of a computer code

Description automatically generated with low confidence

ImGUI event system will be skiped since imgui\_impl\_glfw has a complete event system.

# 14. Polling Input, and Key and Mouse code

Instead of event handling system, we need to build another system, that can allow user to ask whether some event happened. For example, the user wants to build a program, so that when the alt key is pressed, he can rotate the model. Here, we need the polling input.

Define these functions in input.h

A picture containing text, screenshot, font

Description automatically generated

Implement these functions in WindowsInput.cpp

A picture containing text, screenshot, font, number

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

Test it in Application Run function

A picture containing text, screenshot, font, number

Description automatically generated

1. For key and mouse code, we don’t want to use glfw in our sandbox. So, we need to establish our own key code

2. glfw and win32 and other libraries have it own key code. So, we need to separate the glfw code from our code.

A screenshot of a computer code

Description automatically generated with low confidence

# 15. Math

It is easy to write a math library by yourself, but it is very hard to fully utilize CPU resources. For example, how to use AVX instruction set? How to write in a Single instruction multiple data (SIMD) in a 128 bit instruction? You can either write in assembly code, or compiler intrinsics.

Assembly code: Writing your math library functions in assembly code allows you to have fine-grained control over the instructions and registers used by the CPU. You can directly write instructions using assembly language mnemonics specific to the target architecture, such as x86 or x86-64. This approach provides the highest level of control and optimization but requires expertise in assembly programming.

Compiler intrinsics: Compiler intrinsics are functions provided by the compiler that allow you to write code that directly maps to specific CPU instructions. They provide a higher-level abstraction compared to assembly code and are typically written in C/C++. Compiler intrinsics are architecture-specific, so you need to use the appropriate intrinsics for the target CPU. For example, to use AVX instructions, you would use AVX intrinsics. Intrinsics make it easier to write SIMD code without diving into low-level assembly programming.

Using compiler intrinsics, you can express SIMD operations in a more readable and maintainable manner compared to assembly code. The intrinsics are typically defined as inline functions or macros, and the compiler translates them to the corresponding SIMD instructions during the compilation process.

Both approaches have their advantages and disadvantages. Writing in assembly code provides the utmost control and optimization possibilities, but it requires expertise and can be more challenging to maintain. Compiler intrinsics offer a higher-level abstraction and are easier to work with, but they may not provide the same level of fine-grained control as assembly programming.

A picture containing text, screenshot, font

Description automatically generated

Download glm: <https://github.com/g-truc/glm/releases> , and add it into the include directory of premake5.lua

# 16. ImGUI docking and Viewport

Docking: like browser, we can drag the one window and dock it to anywhere we want. And for viewport, we can drag the window and put it outside out application. In the previous episode, Cherno implemented all the event function for ImGUI, but it is butter to use it own functions since ImGUI will update over time.

A screen shot of a computer

Description automatically generated with medium confidence

Include this file to tell the compiler to build the ImGui OpenGL and GLFW implementation.

Add a new ImGui layer

A picture containing text, screenshot, font

Description automatically generated

Append this layer into the Application layer stack

A screen shot of a computer program

Description automatically generated with low confidence

And in the run function

A picture containing text, screenshot, font, number

Description automatically generated

Also remove the ImGui layer in sandbox. Now, we add include directory in sandbox premake file, and draw something in the ImGui rendering.

A screenshot of a computer code

Description automatically generated with low confidence

We can find that those external symbol is not found. Because ImGui is build as a static library, and GameEngine is build as a dll file. Now, if you use ImGui::Begin() in sandbox, sandbox will call GameEngine.dll, but this dll doesn’t use ImGui::Begin(), so this function is not compiled into the dll.

1. we can use those functions in GameEngine to make the compiler compile the function into dll, and also use \_\_declspec(export) in ImGui to compile the ImGui function into dll.

2. A better choice is that, use Linker->Input->Module Definition File, so that everything in ImGui static library will be linked to the dll. But def file is not easy to make. You need to create such a .def file, and manually write all the functions you want to link to the dll from the static library ImGui, include the function signature.

A screenshot of a computer

Description automatically generated with medium confidence

3. Just change the GameEngine link to static link. It is also good for the compiler to do optimization since it knows how the function come from. We don’t need those \_\_declspec and can also get rid of those warning.

We will use solution 3 to change everything to static link.

# 17. Introduction to Rendering

If you want a very high performance engine, you will not choose OpenGL, you should choose Vulcan or DirectX. Vulcan is more complicated than OpenGL, and it may take 1500 lines of code to draw a simple triangle. For a example, OpenGL is at a 10/10 level of abstraction, while Vulkan is in 1/10 level, and Direct12 is in 3/10 level, and Direct11 and Metal are around 5/10. OpenGL is the simplest API, while Vulcan is the most difficult API.

You can’t have only one graphics API in the engine. OpenGL is like Java, it can work everywhere (windows, Linux, Mac, Android, IOS), however, it is not the best choice for each platform. For windows and Xbox, the best choice is DirectX 11 or 12, for Mac and IOS it’s Metal, for Linux and Android and it’s Vulkan. That’s why you need to design additional abstraction layer to hide all the Graphics layer.

But however, all of those API works completely differently. It is really hard to extract out the features and build the abstraction layer.

To achieve multithread, the game engine need to have a work queue, which you can submit the work into the queue, and multiple workers will take out the work and finish. Vulkan has it own implementation, and you just need to put all the rendering task into Vulkan, while OpenGL hasn’t. So, you will need to implement it by yourself.

A picture containing text, screenshot, menu, font

Description automatically generated

Also, you should notice that what we want to do is to implement a general game engine, and it is definitely perform worse compared to a specific game engine designed for a specific game.

A screenshot of a computer

Description automatically generated with low confidence

# 18. Rendering Context

In GLFW make context current function:

A screenshot of a computer program

Description automatically generated with medium confidence

There is a context class to contain the context. It will set the window as a render target. GLFW has already abstract this layer for us, and what we want to do is to implement this ourselves. Because GLFW supports OpenGL and Vulkan, but we also want it to supports DirectX and Metal.

In the Graphics Context base class

A picture containing text, screenshot, font, line

Description automatically generated

In OpenGL Context inherit class

A screen shot of a computer code

Description automatically generated with low confidence

# 18. First triangle

This course, we will draw our first triangle regardless of any structure design.

In application.h

A black text on a white background

Description automatically generated with low confidence

In the constructor of application class:

A screenshot of a computer code

Description automatically generated with low confidence

In the Run function

A screenshot of a computer code

Description automatically generated with low confidence

You will see a triangle

A black and white triangle

Description automatically generated with medium confidence

In the following course, we will substitute those code step by step by our module.

# 19. OpenGL Shaders

<https://www.khronos.org/opengl/wiki/Shader_Compilation>

Create the shader class

A picture containing text, screenshot, font

Description automatically generated

Copy the code from Wiki example, and paste it into the constructor of Shader.

A picture containing text, screenshot, font, design

Description automatically generated

A screenshot of a computer code

Description automatically generated with low confidence

Add it into the application

A black text on a white background

Description automatically generated with low confidence

Write the shader program

A screenshot of a computer program

Description automatically generated with medium confidence

Bind shader before draw

A screenshot of a computer code

Description automatically generated with medium confidence

A picture containing screenshot, colorfulness, creativity, design

Description automatically generated

# 20. Rendering API abstraction

You can choose:

1. Choose your graphics API when you compile. You may choose OpenGL or DirectX to compile your program. The disadvantage of this is that you need a separate binary build of the game engine to select the API you choose. And it is not good for a developer. For a developer, he need to know the difference between OpenGL version and DirectX version, and he needs to switch between this two renderers when the program is running. However, if the API is selected when compile the program, he needs to compile the program over and over to check the different. The advantage is that the performance is good. The engine don’t need to decide which engine you use while the program is running.

2. Choose your graphics API in the run time. You can use if else statements to switch between API, but the performance is bad. You can also build a pure virtual class as renderer, and branch them by different inherit class like OpenGL renderer or DirectX renderer. In the runtime, when you create the shader, if current shader is OpenGL, then you create an OpenGL shader.

Even if you can choose your graphics API while in runtime, you still need to define macro to disable some part of your API. For example, in Mac or IOS, you need to disable all the code relate to DirectX, since it can not be compiled.

Here we will first create buffer (vertex buffer and index buffer), and we will use create buffer instead of gen buffer. Mind that glCreateBuffers is a newer function introduced in OpenGL 4.5 that combines name generation and memory allocation for multiple buffer objects, while glGenBuffers is an older function that only generates names for buffer objects without allocating memory.

Define Render API

A screen shot of a computer

Description automatically generated with low confidence

First create buffer base class

A picture containing text, screenshot, font

Description automatically generated

Define create function

A screen shot of a computer code

Description automatically generated with low confidence

Use OpenGL buffer to inherit it

A screen shot of a computer code

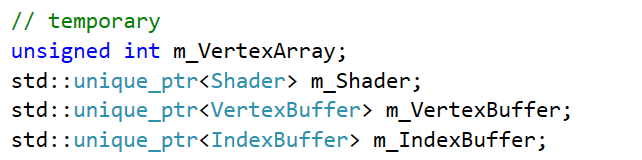
Description automatically generated with low confidence

Implement OpenGL function

A picture containing text, screenshot, font

Description automatically generated

Finally in application.h, define those class



A picture containing text, screenshot, font, line

Description automatically generated

A picture containing text, font, screenshot, line

Description automatically generated