

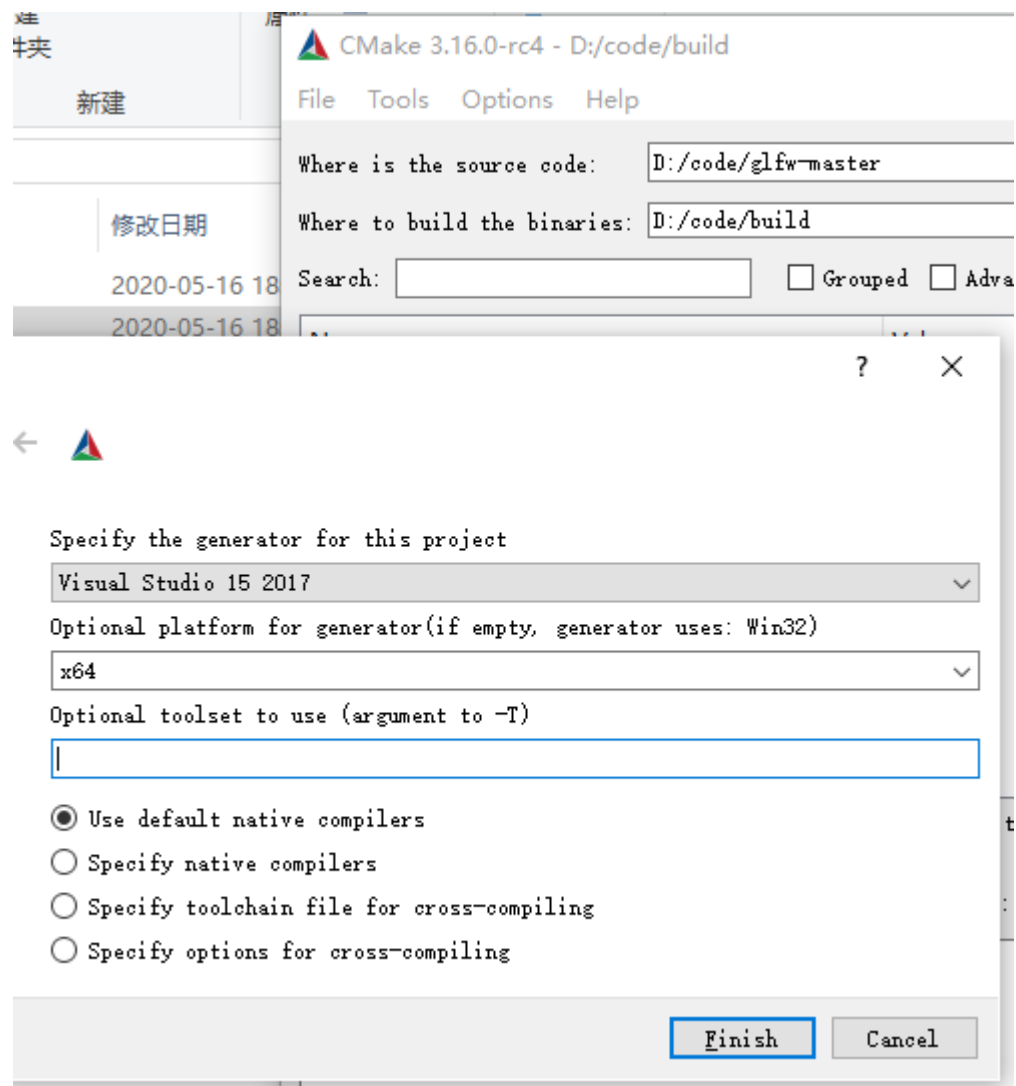
图形学大作业

(数媒 1701 陈汉轩 201726010211)

目录

图形学大作业.....	1
(数媒 1701 陈汉轩 201726010211)	1
步骤 1: 在 Github 下载 glfw 源码并编译.....	2
步骤 2: 在 Github 下载 glew 源码并编译.....	7
步骤 3: VS 新项目测试环境.....	13
步骤 4: 编写各部分基础类.....	14
步骤 5: Github 下载导入 glm 库.....	17
步骤 6: Github 添加 imgui 制作 UI 界面.....	19
步骤 7: 安装 GLAD.....	23
步骤 8: 编写相机类.....	23
步骤 9: 绑定二维码生成截屏到 GUI.....	28
步骤 10: 动态纹理更新	29
附录	31
参考资料.....	31

步骤 1: 在 [Github](#) 下载 glfw 源码并编译

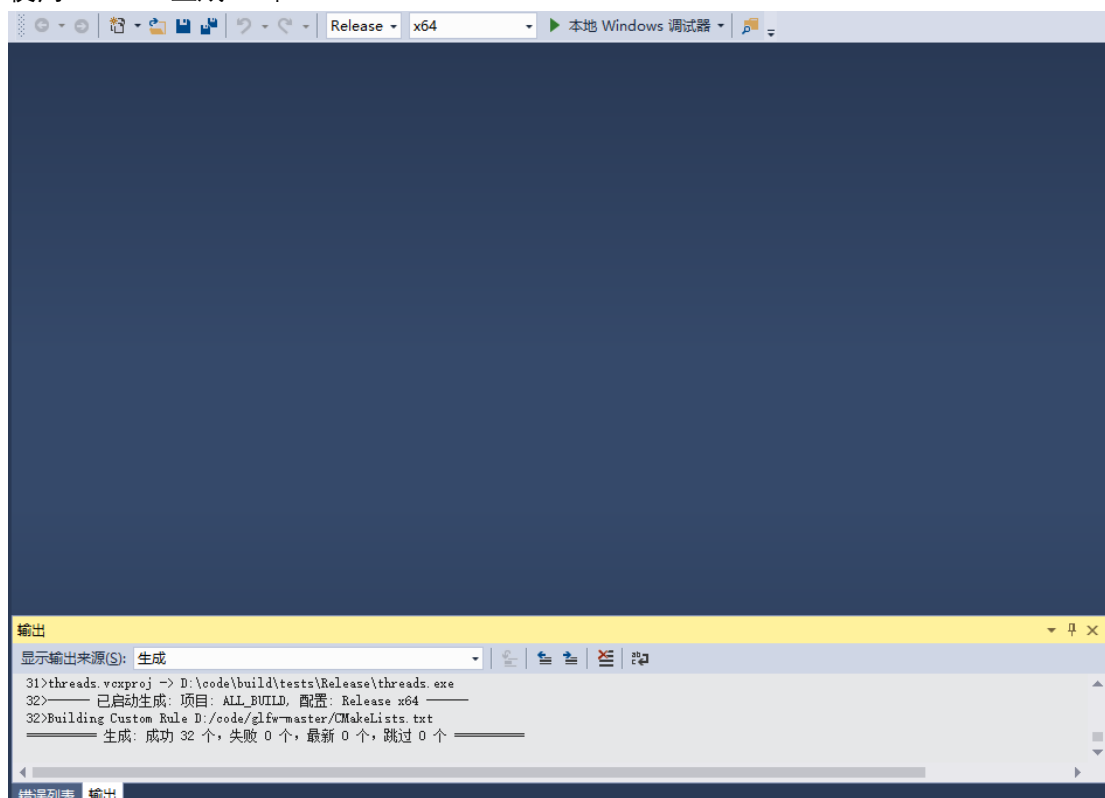


经过 configure 和 generate 之后得到:

此电脑 > 新加卷 (D:) > code > build >

名称	修改日期	类型	大小
CMakeFiles	2020-05-16 18:31	文件夹	
examples	2020-05-16 18:31	文件夹	
src	2020-05-16 18:31	文件夹	
tests	2020-05-16 18:31	文件夹	
ALL_BUILD.vcxproj	2020-05-16 18:31	VC++ Project	24 KB
ALL_BUILD.vcxproj.filters	2020-05-16 18:31	VC++ Project Fil...	1 KB
cmake_install.cmake	2020-05-16 18:31	CMAKE 文件	5 KB
cmake_uninstall.cmake	2020-05-16 18:31	CMAKE 文件	2 KB
CMakeCache.txt	2020-05-16 18:31	文本文档	16 KB
GLFW.sln	2020-05-16 18:31	Visual Studio Sol...	40 KB
INSTALL.vcxproj	2020-05-16 18:31	VC++ Project	11 KB
INSTALL.vcxproj.filters	2020-05-16 18:31	VC++ Project Fil...	1 KB
uninstall.vcxproj	2020-05-16 18:31	VC++ Project	22 KB
uninstall.vcxproj.filters	2020-05-16 18:31	VC++ Project Fil...	1 KB
ZERO_CHECK.vcxproj	2020-05-16 18:31	VC++ Project	20 KB
ZERO_CHECK.vcxproj.filters	2020-05-16 18:31	VC++ Project Fil...	1 KB

使用 VS2017 生成 lib 和 dll:



得到 glfw3.lib:

新加卷 (D:) > code > build > src > Release

名称	修改日期	类型	大小
glfw3.lib	2020-05-16 18:33	Object File Library	632 KB

创建 opengl_final 文件夹, 并创建 libs 文件夹, 将刚生成的 glfw3.lib 放入:

此电脑 > 新加卷 (D:) > code > opengl_final > libs				
	名称	修改日期	类型	大小
	glfw3.lib	2020-05-16 18:33	Object File Library	632 KB

```
#include <GLFW/glfw3.h>
```

```
int main(void)
```

```
{
```

```
    GLFWwindow* window;
```

```
    /* Initialize the library */
```

```
    if (!glfwInit())
```

```
        return -1;
```

```
    /* Create a windowed mode window and its OpenGL context */
```

```
    window = glfwCreateWindow(640, 480, "Hello World", NULL,  
NULL);
```

```
    if (!window)
```

```
    {
```

```
        glfwTerminate();
```

```
        return -1;
```

```
    }
```

```
    /* Make the window's context current */
```

```
    glfwMakeContextCurrent(window);
```

```
/* Loop until the user closes the window */

while (!glfwWindowShouldClose(window))

{

    /* Render here */

    glClear(GL_COLOR_BUFFER_BIT);


    /* Swap front and back buffers */

    glfwSwapBuffers(window);


    /* Poll for and process events */

    glfwPollEvents();

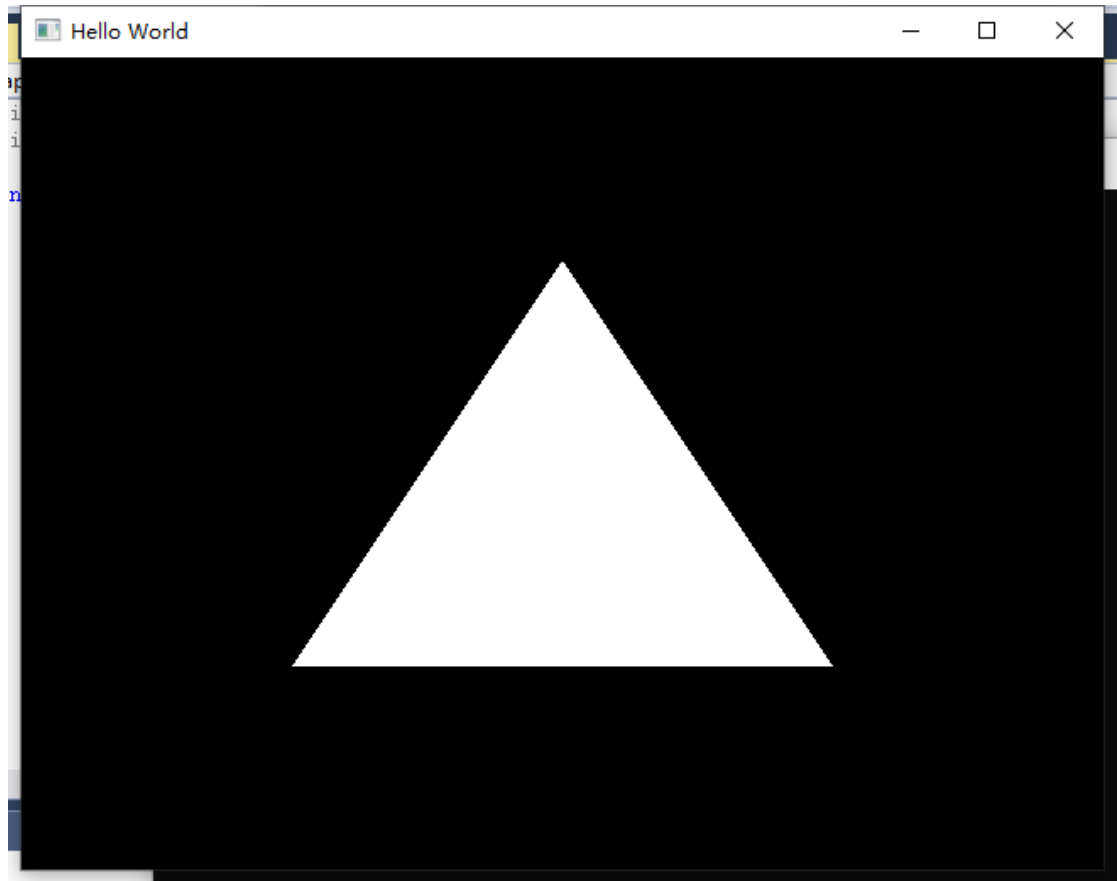
}


glfwTerminate();

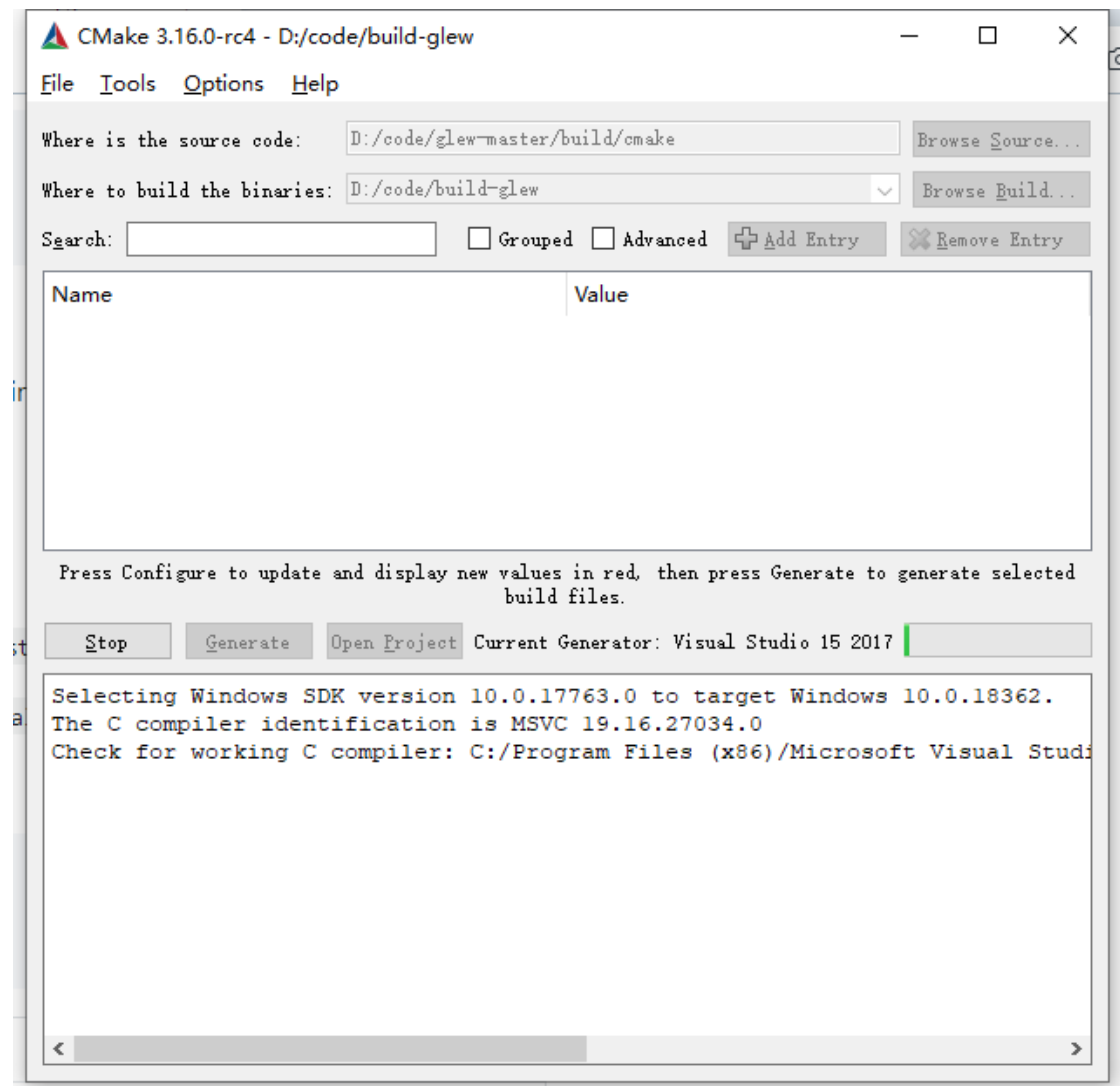
return 0;

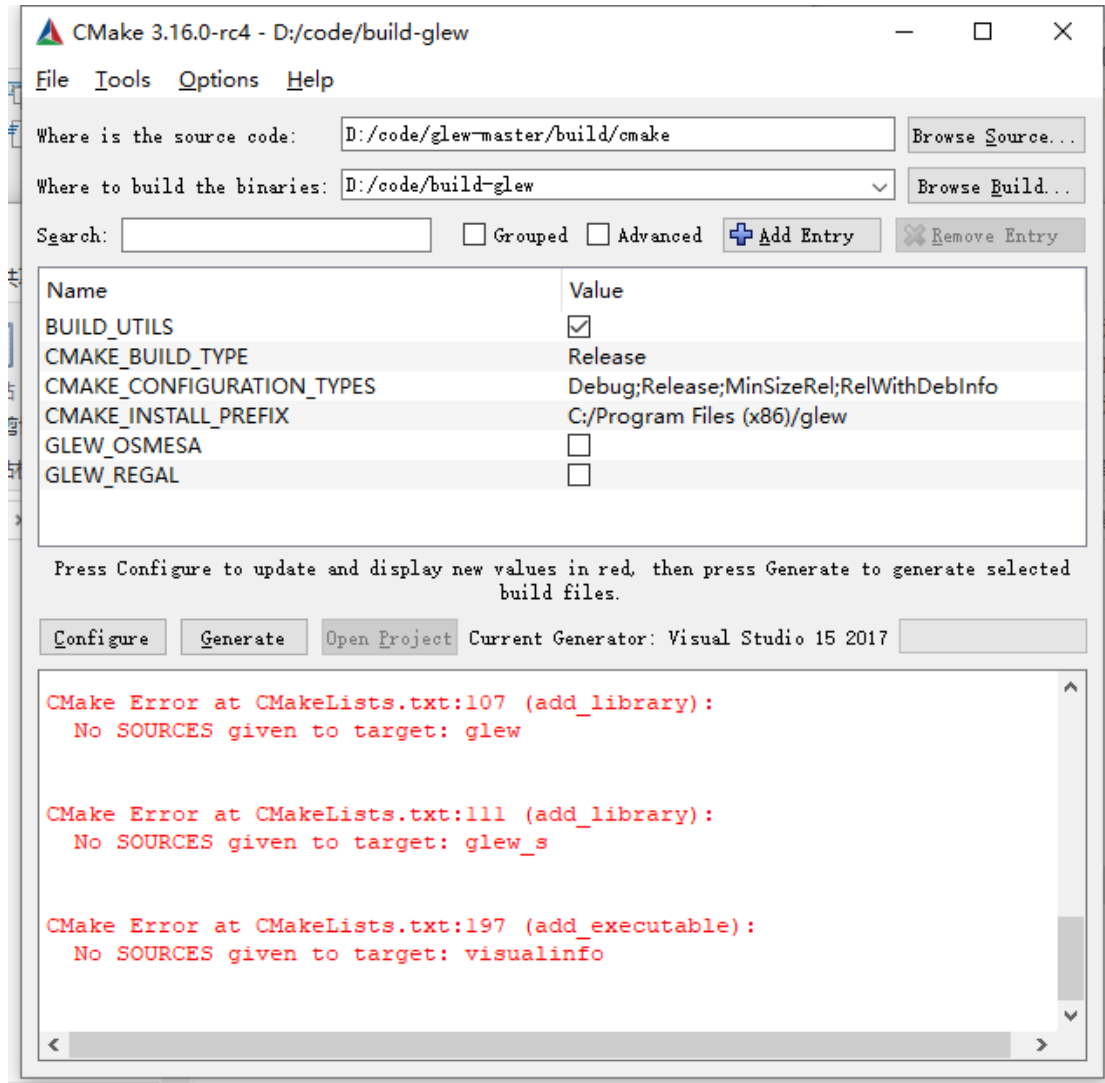
}
```

输入测试代码，发现缺少了 opengl32.lib（报错：__imp_glcClear missing），添入依赖库之后问题解决，可以出现官网的示例代码。

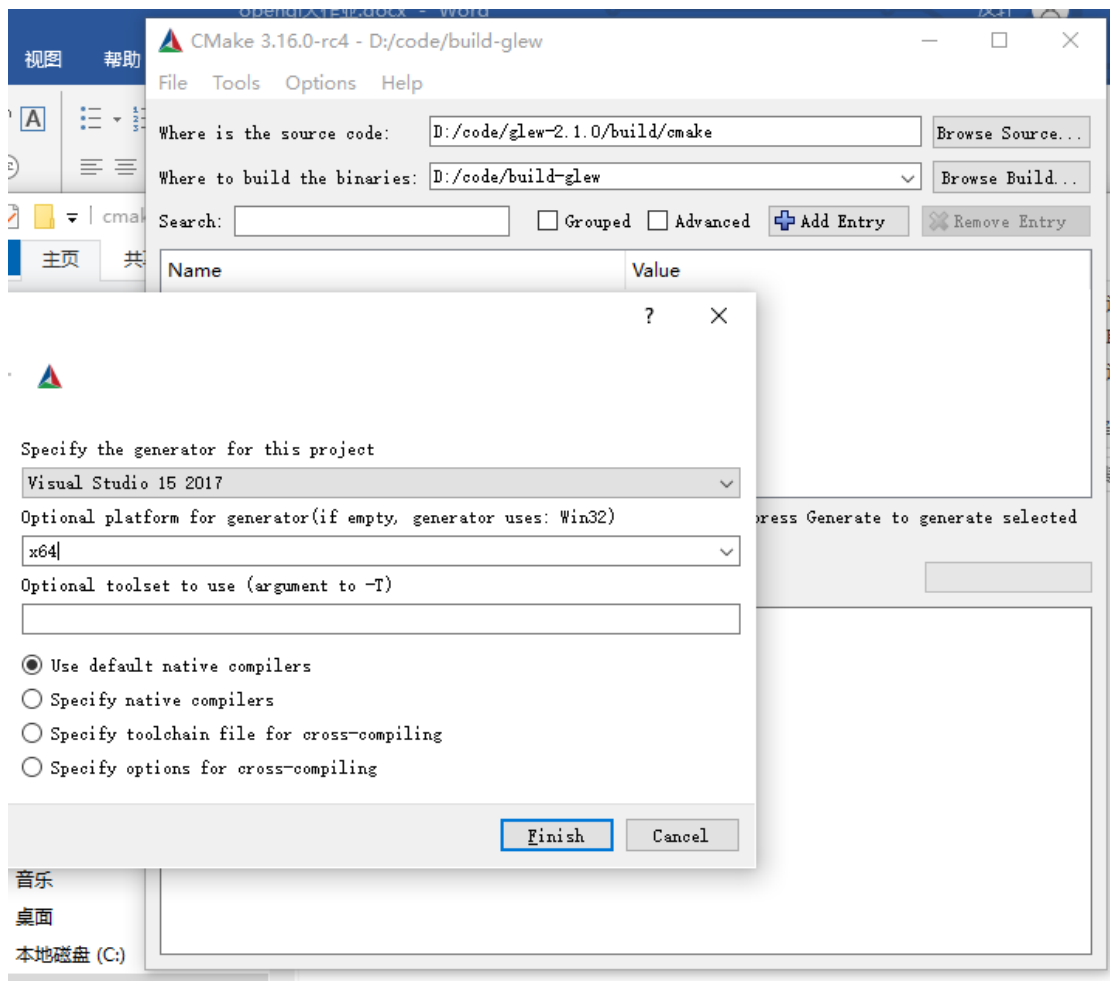


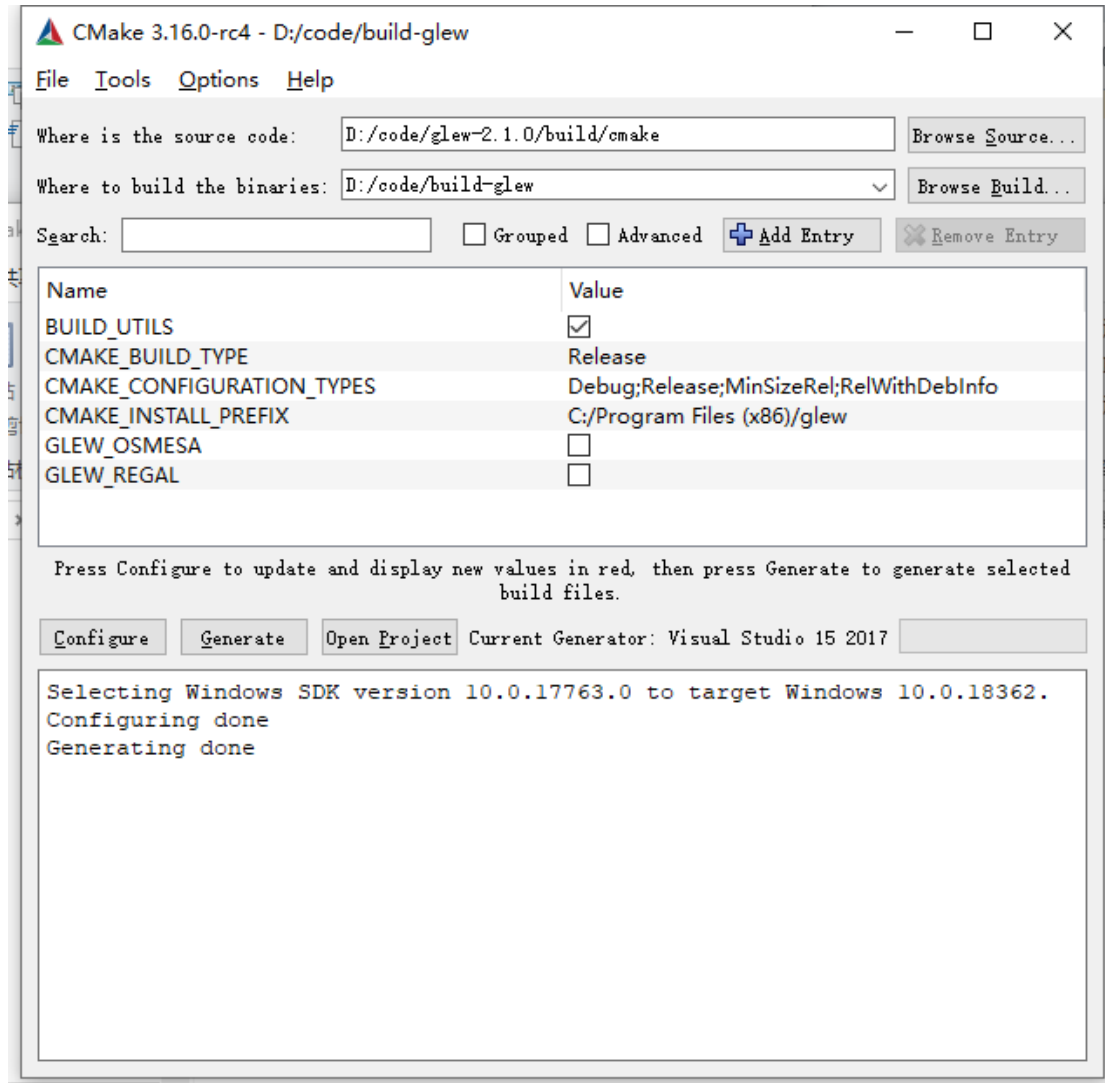
步骤 2: 在 [Github](#) 下载 glew 源码并编译

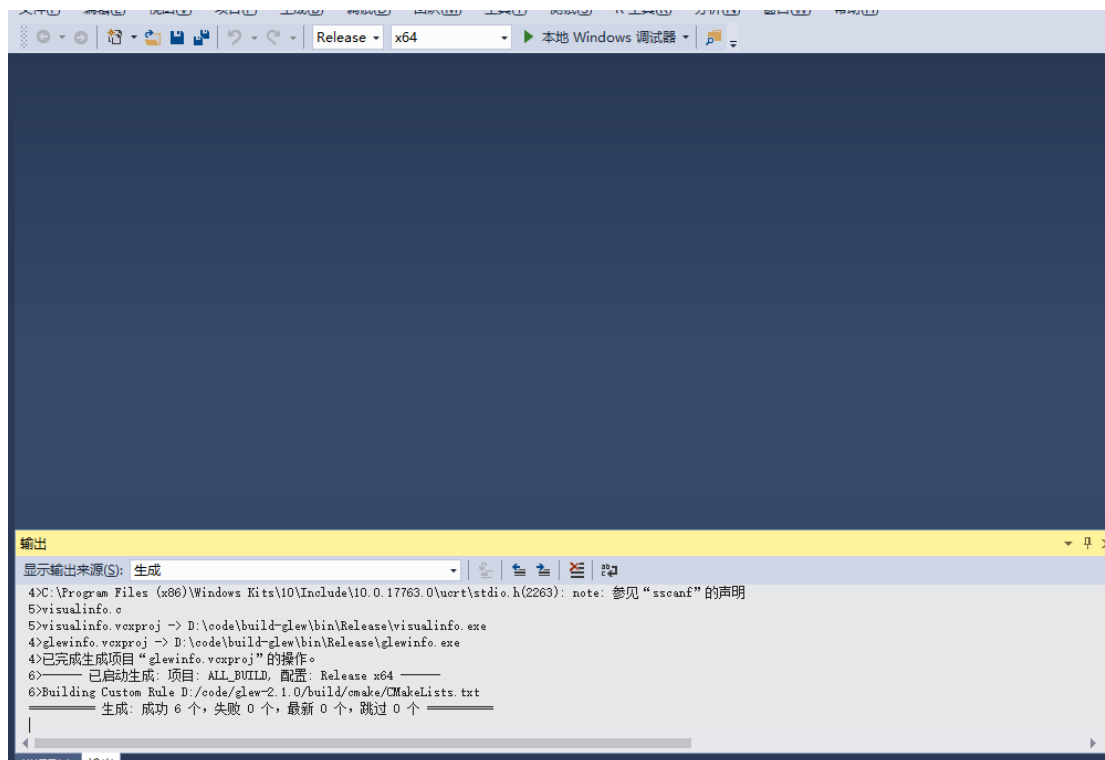




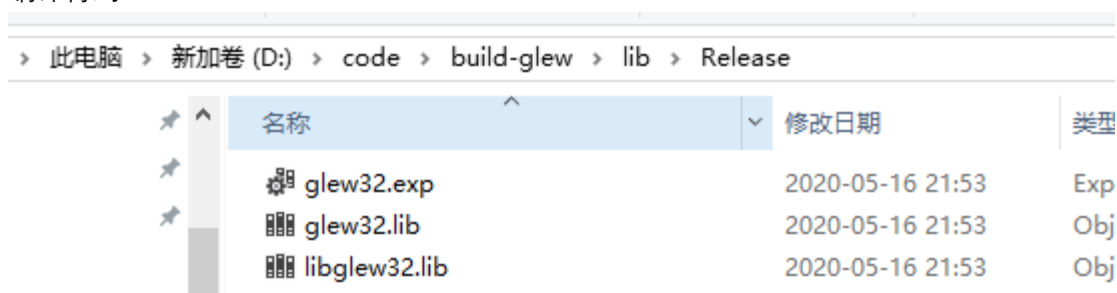
Github 上源码缺失，无法编译，找到历史版本 2.1.0 版本源码，重新编译。



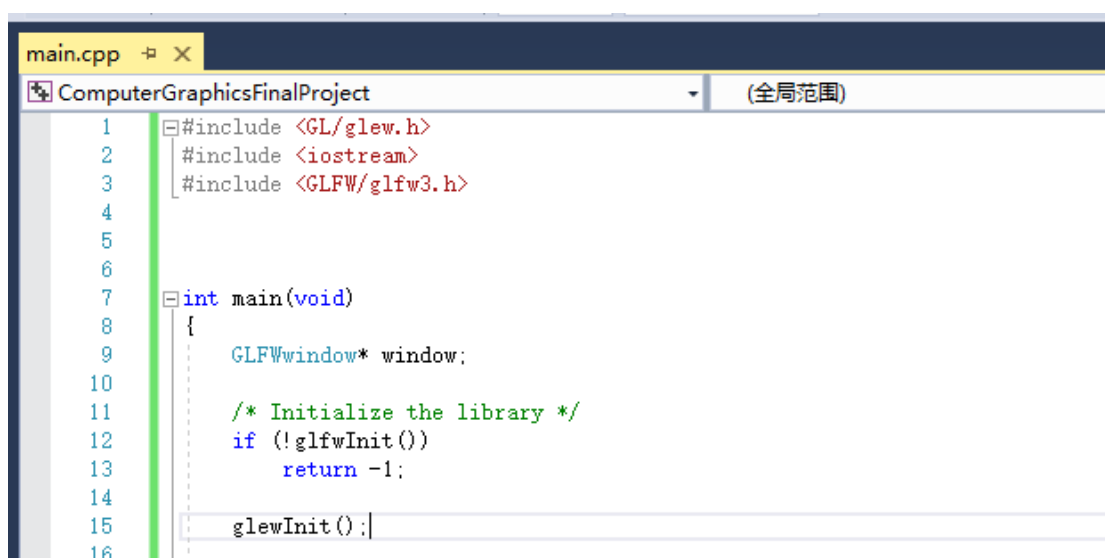




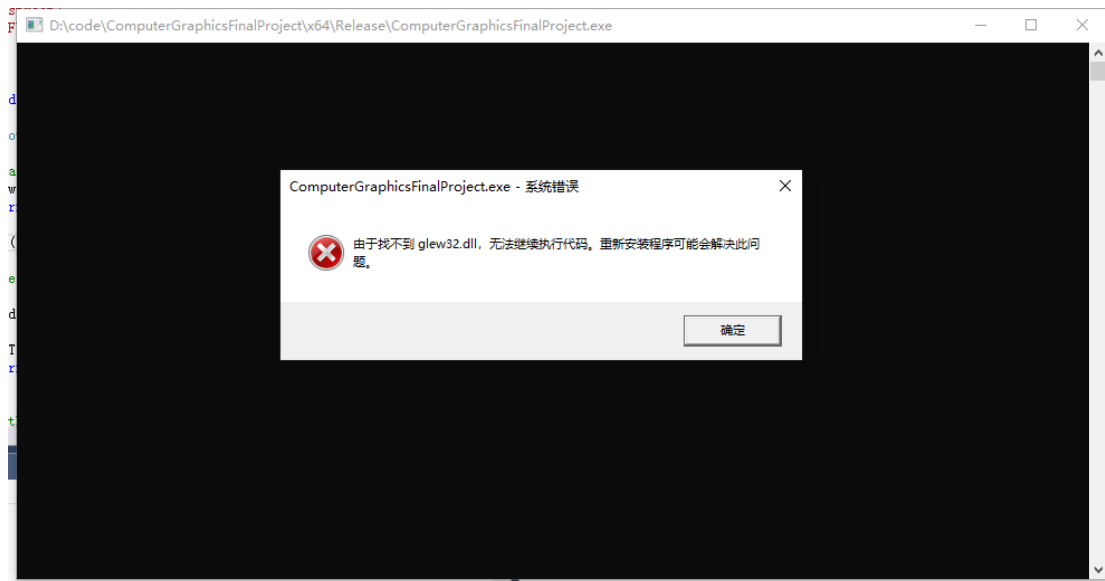
编译得到 lib:



配置 include 和 lib 后测试代码: (将 glew.h 的引入放在最前面)



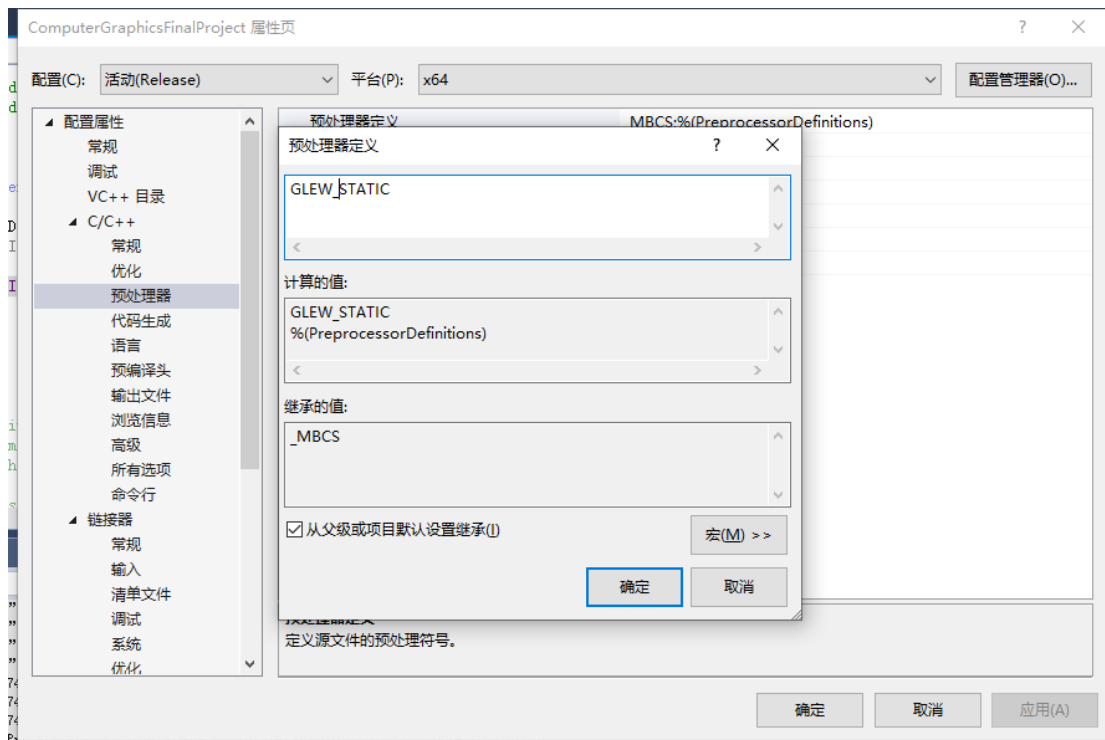
然后引入 glewinit(), 报错:



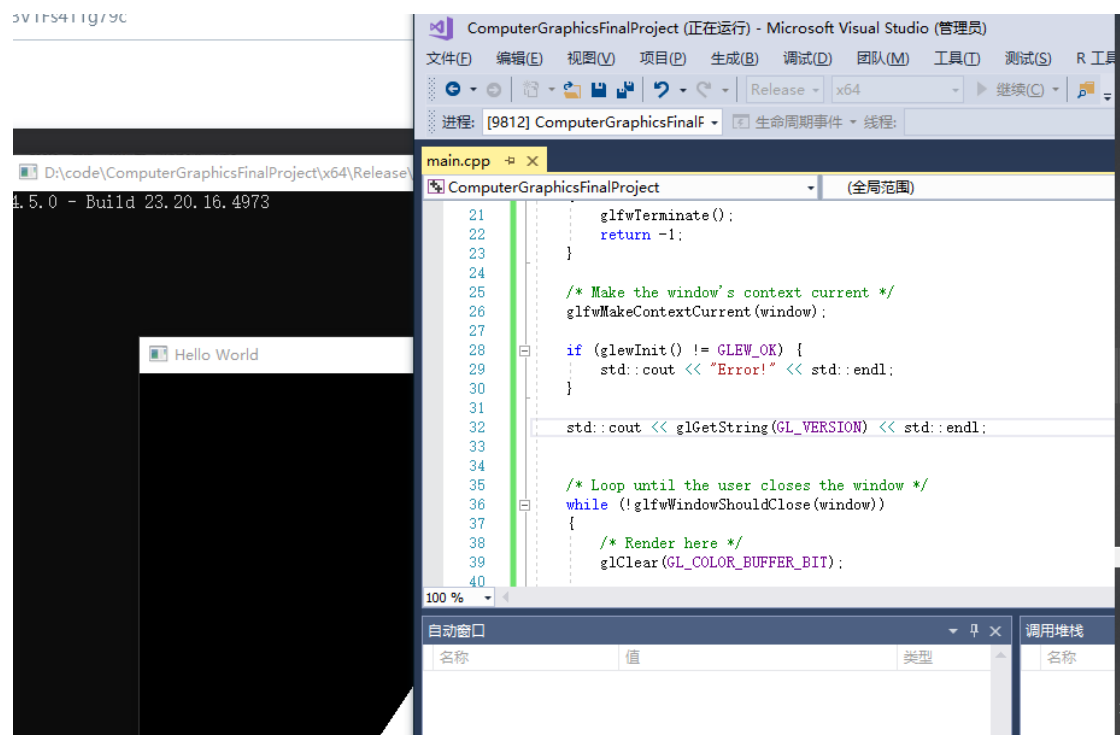
定位库代码:

```
199  
200  #ifdef GLEW_STATIC  
201      # define GLEWAPI extern  
202  #else  
203      # ifdef GLEW_BUILD  
204          # define GLEWAPI extern __declspec(dllexport)  
205      # else  
206          # define GLEWAPI extern __declspec(dllimport)  
207      # endif  
208  #endif
```

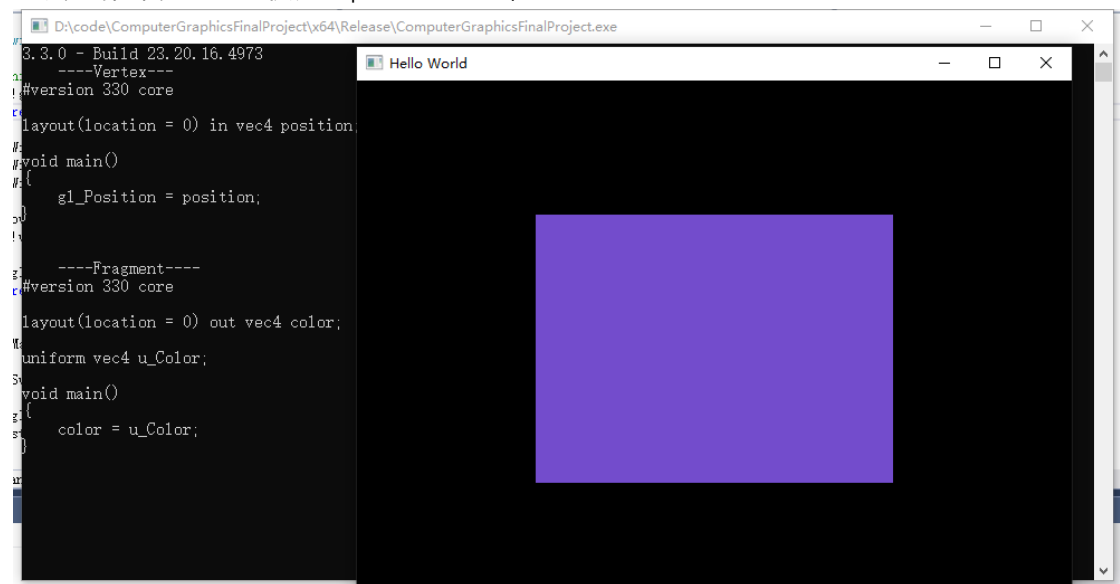
这里应该是调用了 dll 模式, 但是需要静态链接, 在项目中添加预编译宏: GLEW_STATIC



步骤 3: VS 新项目测试环境

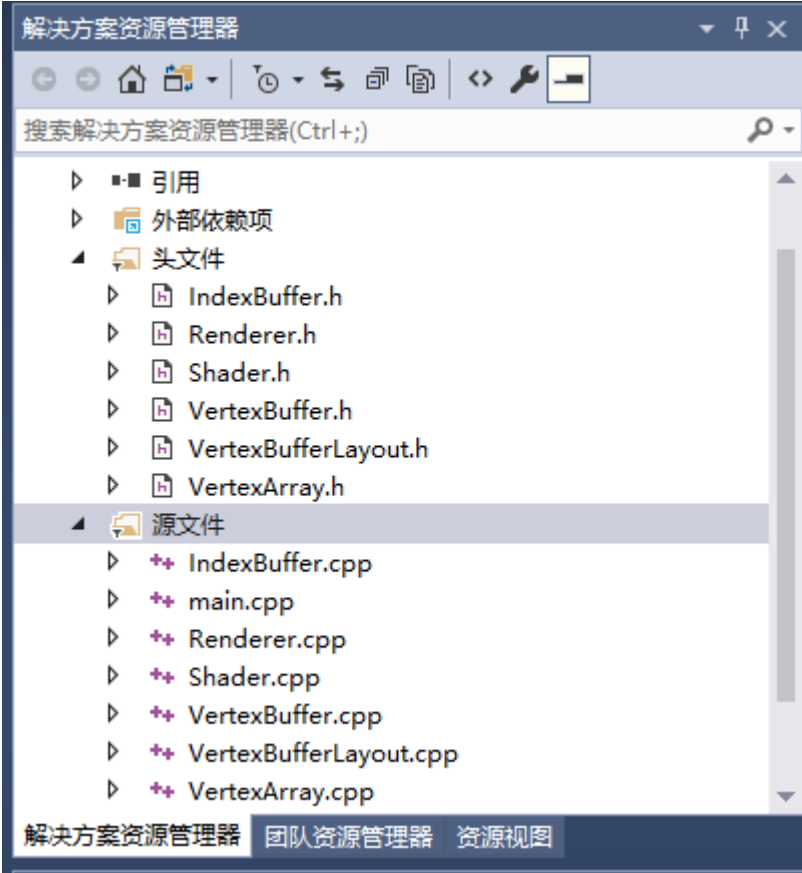


测试工作环境：正常使用 OpenGL4.5 版本。

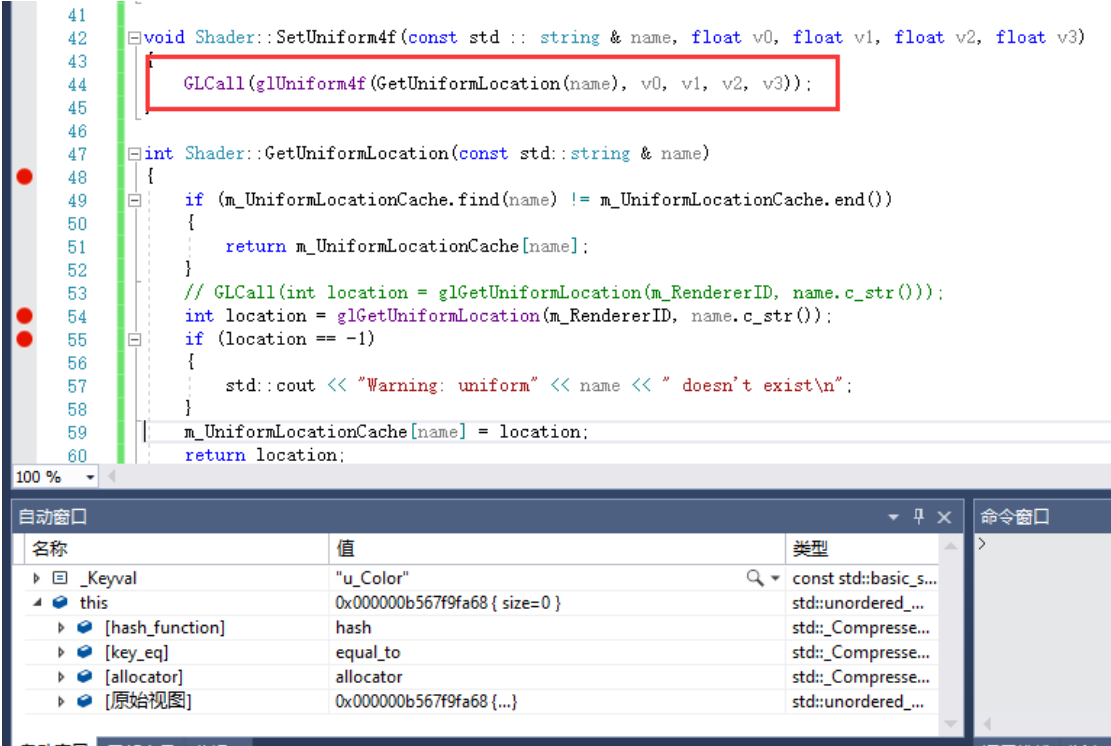


正常使用 OPENGL3.3 CORE。

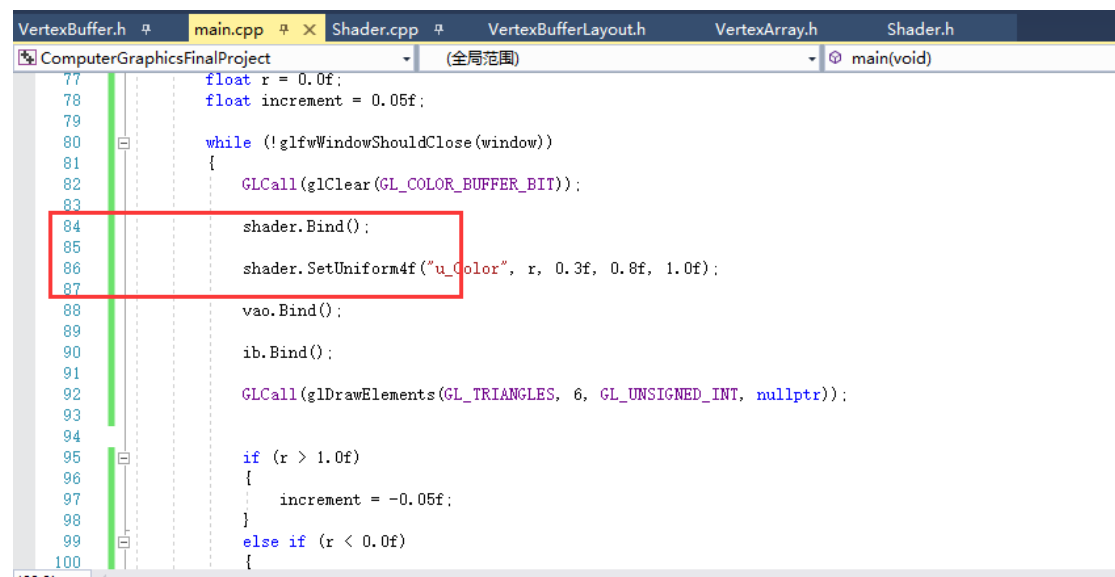
步骤 4: 编写各部分基础类



在抽象出 Shader 类的时候，遇到问题：

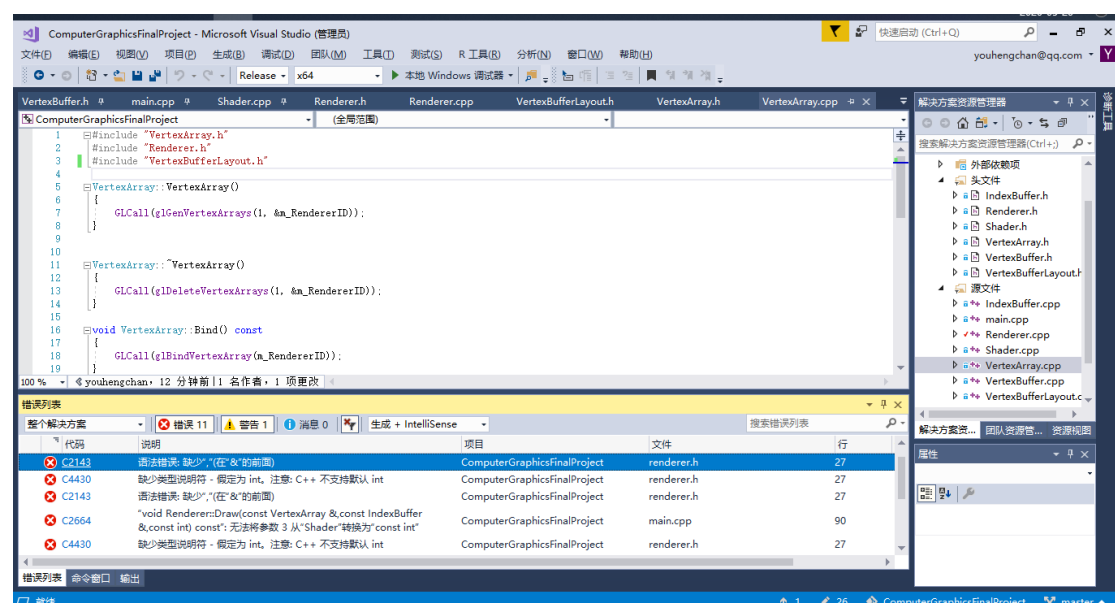


需要补上 67 行，尽管在绘制循环中已经进行了 Bind() 操作，但是这里如果不 Bind，之后调用 SetUniform4f 就会出错。

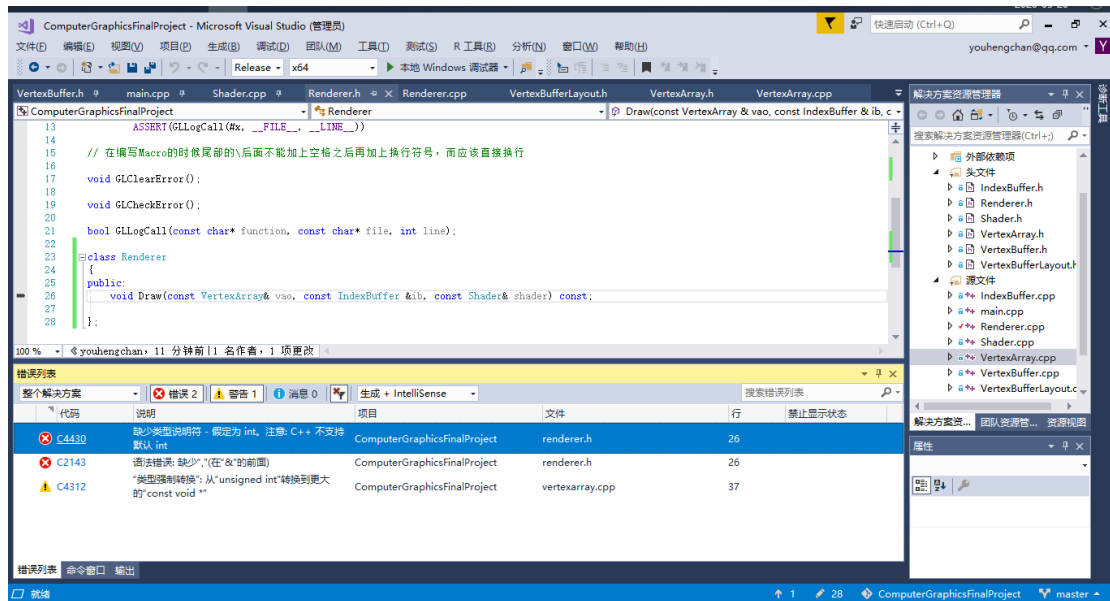


```
77 float r = 0.0f;
78 float increment = 0.05f;
79
80 while (!glfwWindowShouldClose(window))
81 {
82     GLCall(glClear(GL_COLOR_BUFFER_BIT));
83
84     shader.Bind();
85
86     shader.SetUniform4f("u_Color", r, 0.3f, 0.8f, 1.0f);
87
88     vao.Bind();
89
90     ib.Bind();
91
92     GLCall(glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr));
93
94
95     if (r > 1.0f)
96     {
97         increment = -0.05f;
98     }
99     else if (r < 0.0f)
100    {
```

总是需要在 SetUniform 之前首先进行 Bind 操作，这样 GPU 之中才能找到这个 Shader 的信息，否则后面进行 Uniform 参数设置的时候，颜色数据进到 GPU 中无处安放，引起报错。



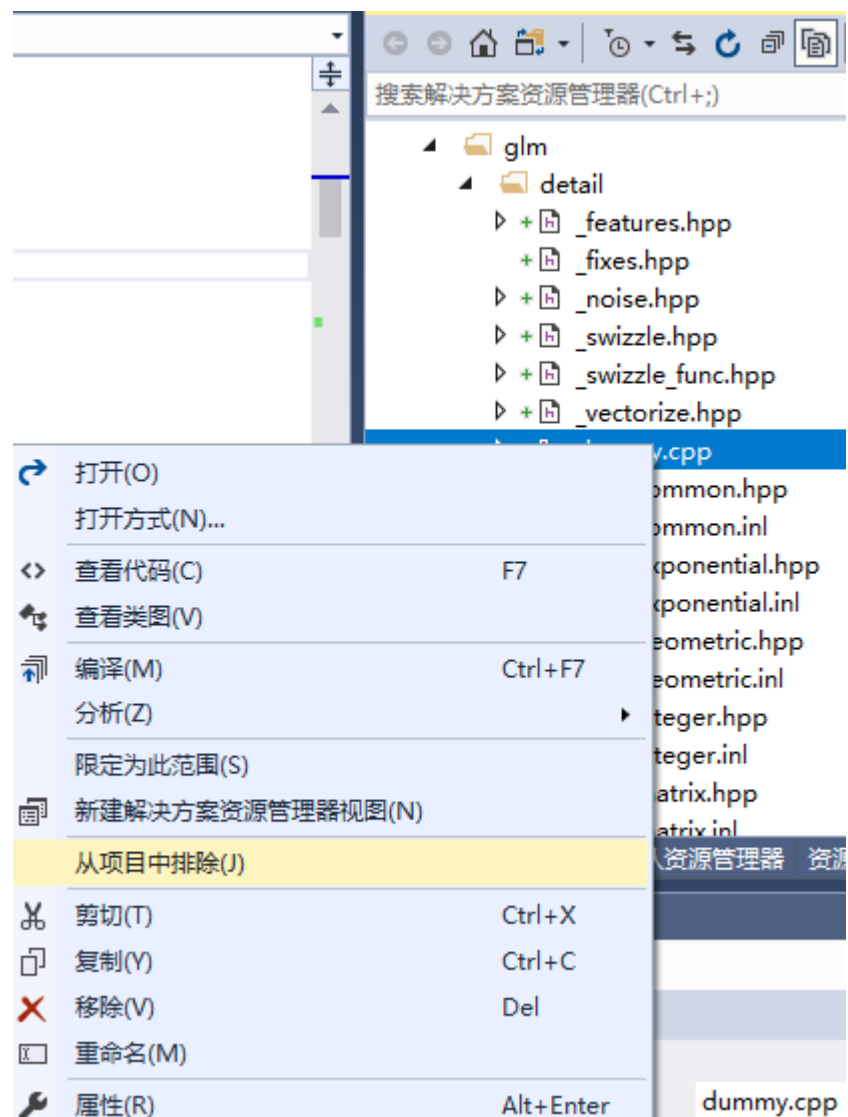
还有循环引用的错误，之后通过类提前声明解决。



步骤 5: [Github](#) 下载导入 glm 库

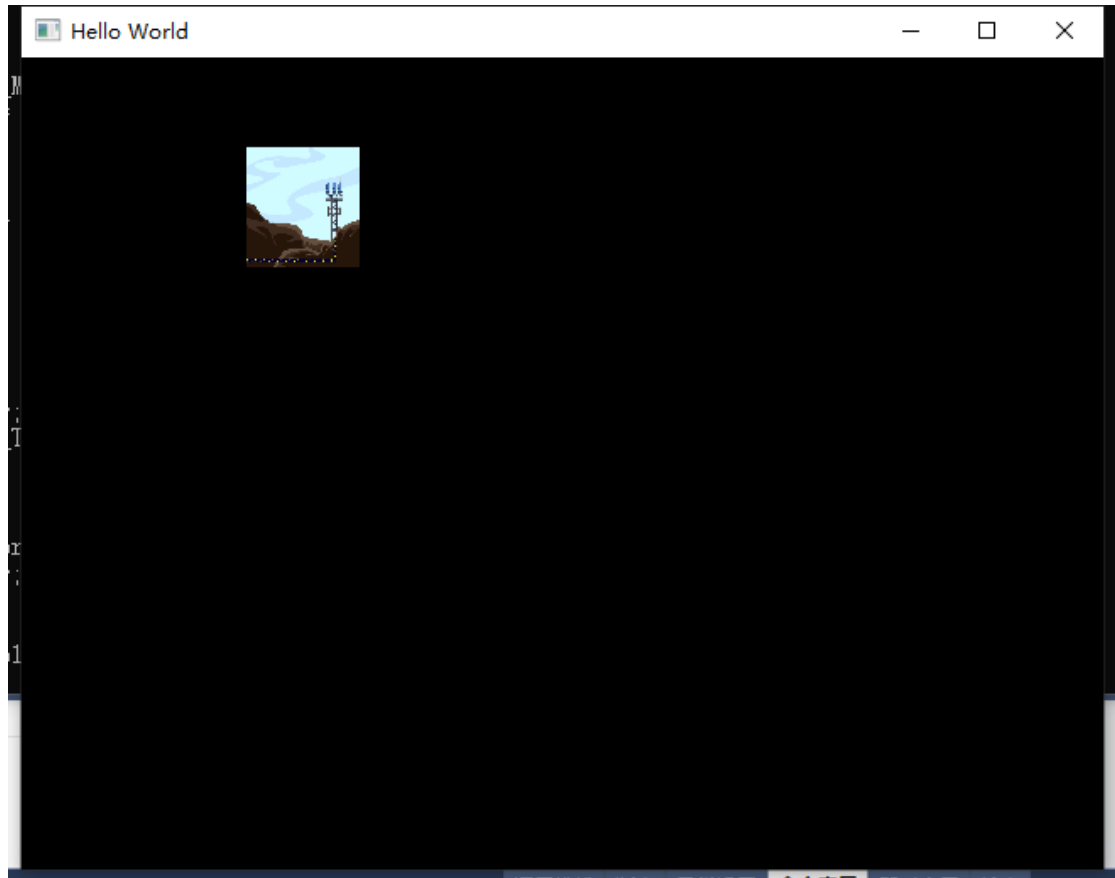
GLM 是一个不需要链接，直接导入就能用的数学库。

发现在 detail 文件夹下有一个 cpp 文件，含有 main 函数，对 dummy.cpp 选择“从项目中排除”。



添加 Include 路径即可，这比配置 GLEW 和 GLFW 简单。

测试正常：



步骤 6: [Github](#) 添加 imgui 制作 UI 界面

将下载的 release 包中的 basefile 部分首先放到项目中:

名称	修改日期	类型
.github	2020-05-21 0:29	文件夹
docs	2020-05-21 0:29	文件夹
examples	2020-05-21 0:29	文件夹
misc	2020-05-21 0:29	文件夹
.editorconfig	2020-04-12 11:18	EDITORCONFIG 文件
.gitattributes	2020-04-12 11:18	GITATTRIBUTE 文件
.gitignore	2020-04-12 11:18	GITIGNORE 文件
imconfig.h	2020-04-12 11:18	C Header File
imgui.cpp	2020-04-12 11:18	C++ Source File
imgui.h	2020-04-12 11:18	C Header File
imgui_demo.cpp	2020-04-12 11:18	C++ Source File
imgui_draw.cpp	2020-04-12 11:18	C++ Source File
imgui_internal.h	2020-04-12 11:18	C Header File
imgui_widgets.cpp	2020-04-12 11:18	C++ Source File
imstb_rectpack.h	2020-04-12 11:18	C Header File
imstb_textedit.h	2020-04-12 11:18	C Header File
imstb_truetype.h	2020-04-12 11:18	C Header File
LICENSE.txt	2020-04-12 11:18	文本文档

然后将源码中的 main.cpp 排除项目。

解决方案“ComputerGraphicsFinalProject”(1 个项目)

ComputerGraphicsFinalProject

- 外部依赖项
- Debug
- Resources
 - LIBS
 - IMGUI
 - imconfig.h
 - imgui.cpp
 - imgui.h
 - imgui_demo.cpp
 - imgui_draw.cpp
 - imgui_impl_opengl3.cpp
 - imgui_impl_opengl3.h
 - imgui_internal.h
 - imgui_widgets.cpp
 - imstb_rectpack.h
 - imstb_textedit.h
 - imstb_truetype.h
 - main.cpp**
 - Textures
 - x64
 - Basic.shader

该库已经进行了宏检查：

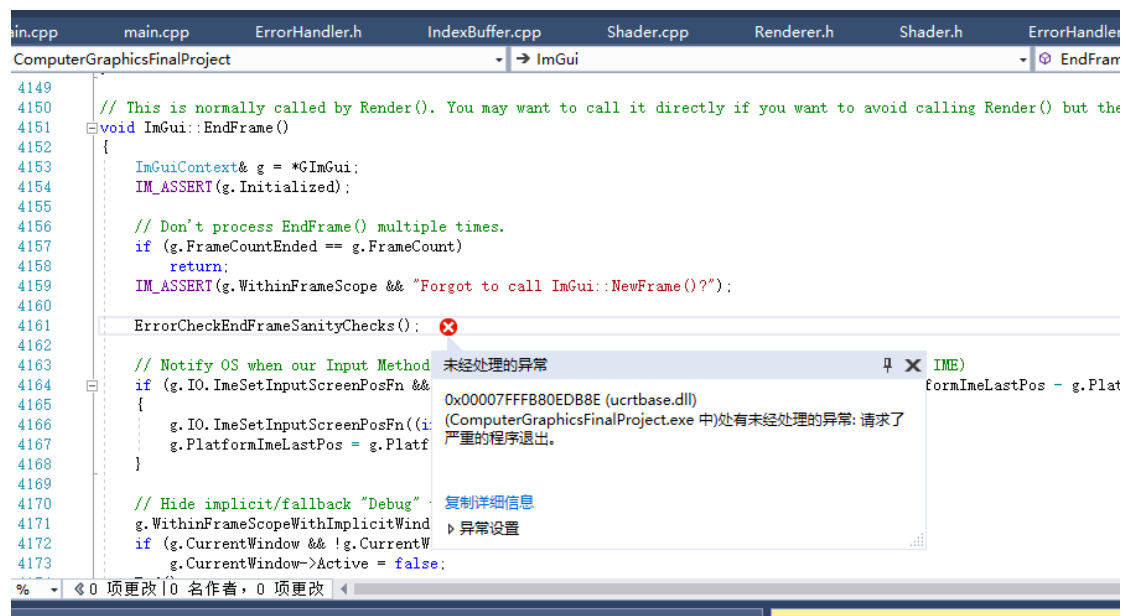
```

110     |#endif
111     |#else
112     |// About Desktop OpenGL function loaders:
113     |// Modern desktop OpenGL doesn't have a standard portable
114     |// Helper libraries are often used for this purpose! Here
115     |// You may use another loader/header of your choice (eg
116     |#if defined(IMGUI_IMPL_OPENGL_LOADER_GL3W)
117     |#include <GL/gl3w.h>           // Needs to be initialized before opengl_
118     |#elif defined(IMGUI_IMPL_OPENGL_LOADER_GLEW)
119     |#include <GL/glew.h>          // Needs to be initialized before opengl_
120     |#elif defined(IMGUI_IMPL_OPENGL_LOADER_GLAD)
121     |#include <glad/glad.h>        // Needs to be initialized before opengl_
122     |#elif defined(IMGUI_IMPL_OPENGL_LOADER_GLBINDING2)
123     |#define GLFW_INCLUDE_NONE    // GLFW including OpenGL headers will
124     |#include <glbinding/Binding.h> // Needs to be initialized before opengl_
125     |#include <glbinding/gl/gl.h>
126     |using namespace gl;

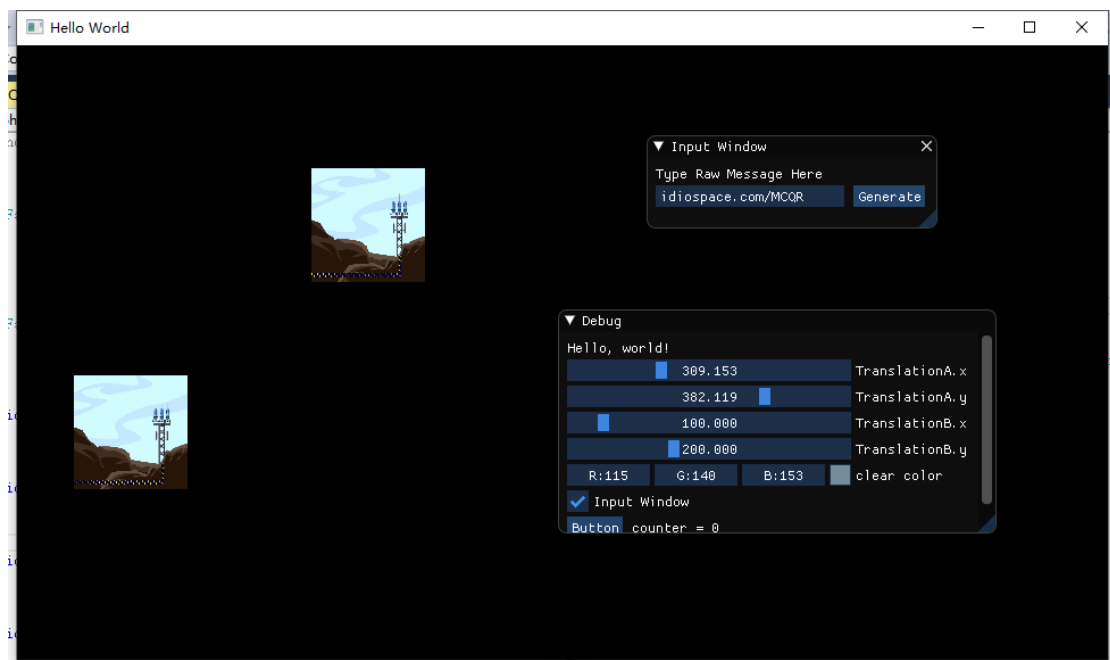
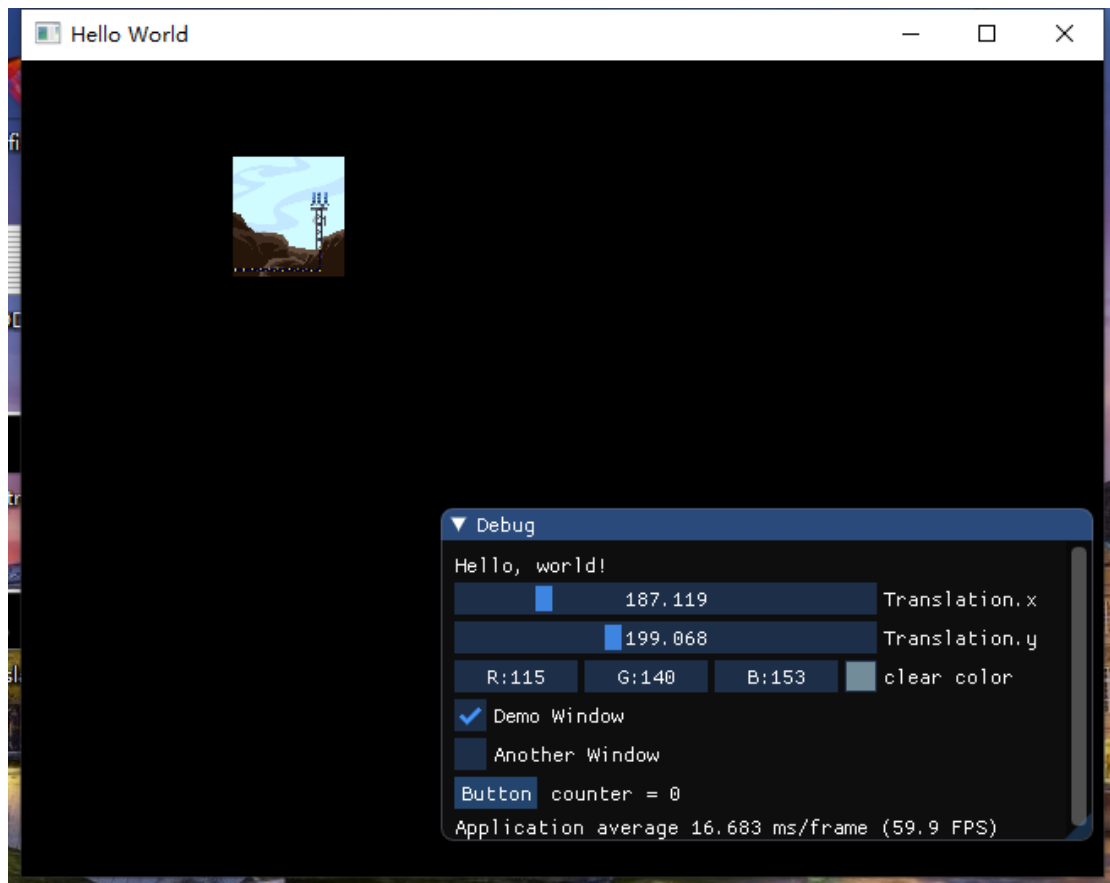
```

检测到使用 GLEW 库之后会自动调用 GLEW 的接口。

按照示例文件 main.cpp 调用该 GUI 框架。

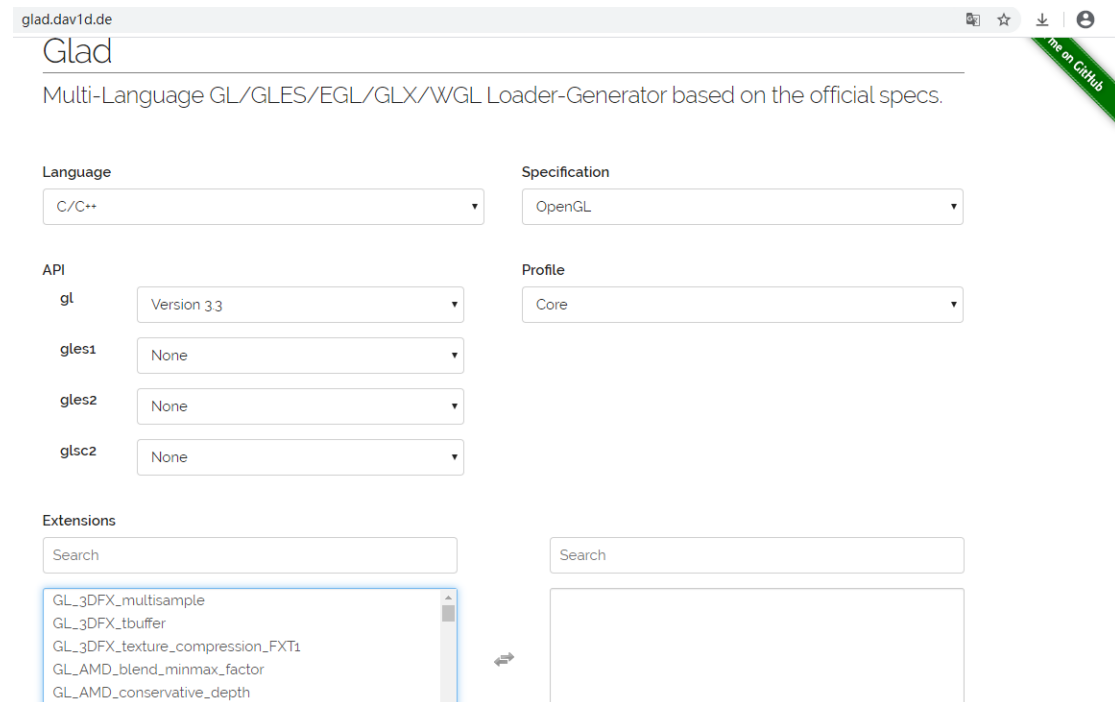


问题很多，这个库。



总体上，加入了小的控制窗口。

步骤 7: 安装 GLAD



glad.dav1d.de

Glad

Multi-Language GL/GLES/EGL/GLX/WGL Loader-Generator based on the official specs.

Language: C/C++

Specification: OpenGL

API: gl, Version 3.3

Profile: Core

Extensions: Search

GL_3DFX_multisample

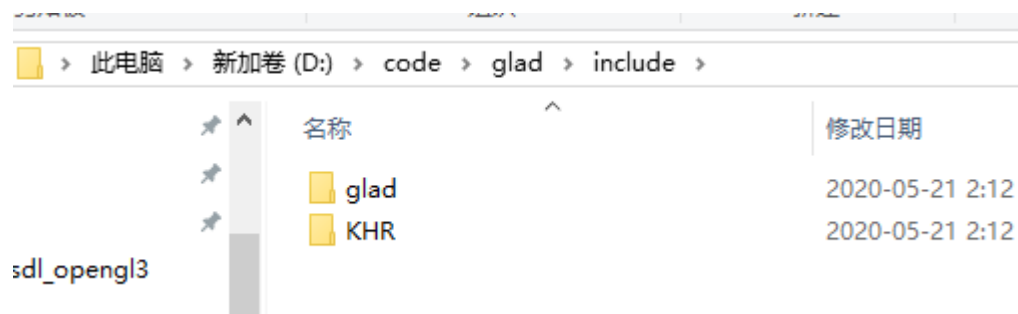
GL_3DFX_tbuffer

GL_3DFX_texture_compression_FXT1

GL_AMD_blend_minmax_factor

GL_AMD_conservative_depth

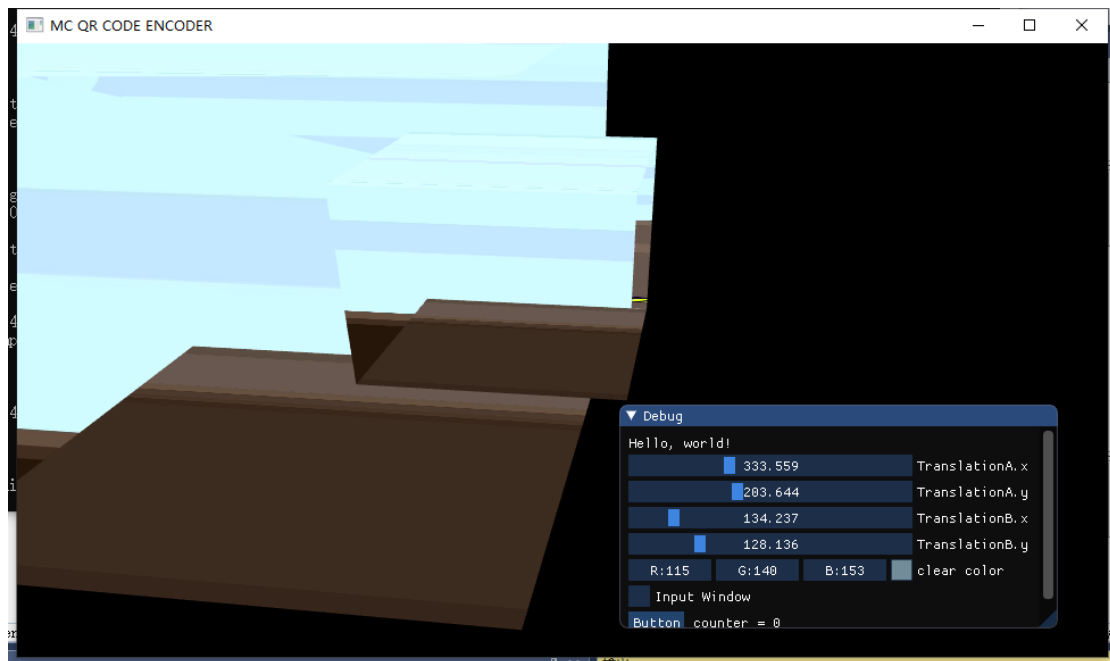
打开 glad 的在线安装服务。
将得到的 glad.zip 中的 cpp 和 h 接入项目。



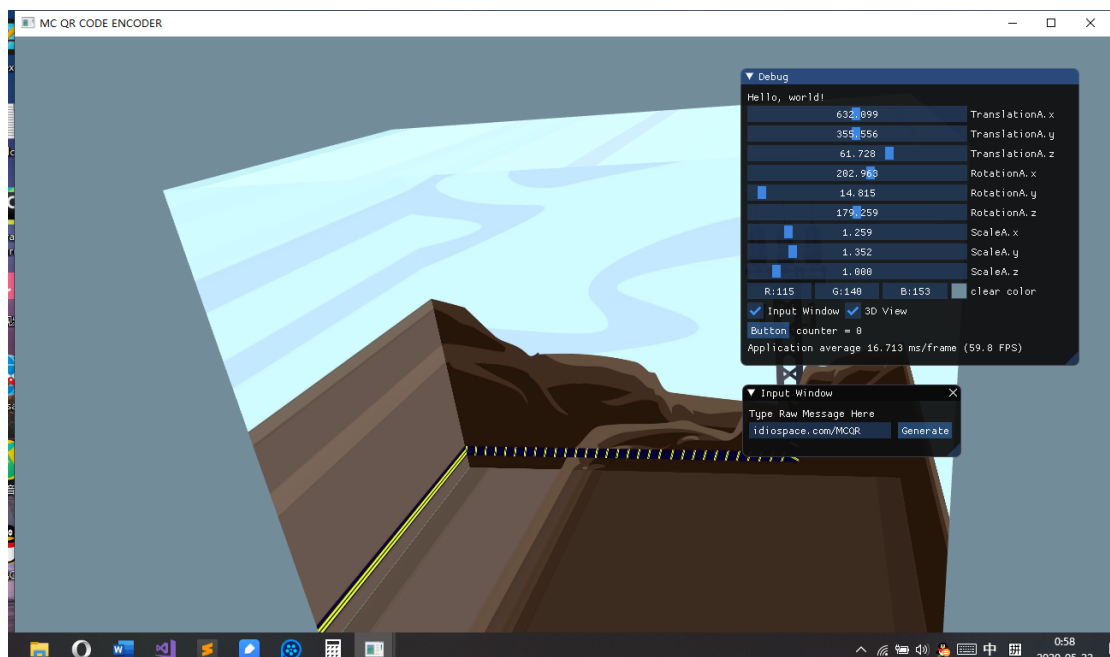
这个库后面废弃了，版本不兼容，错误太多。

步骤 8: 编写相机类

首先将 Project 中的测试的二维的图片换为三维的 Cube，并贴上合适的纹理测试。

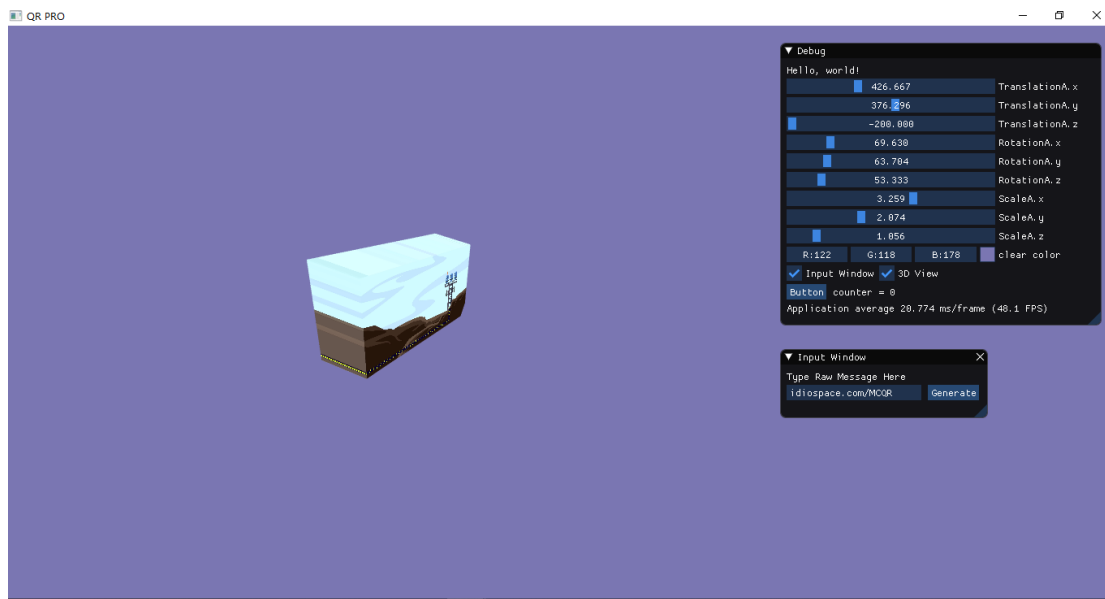


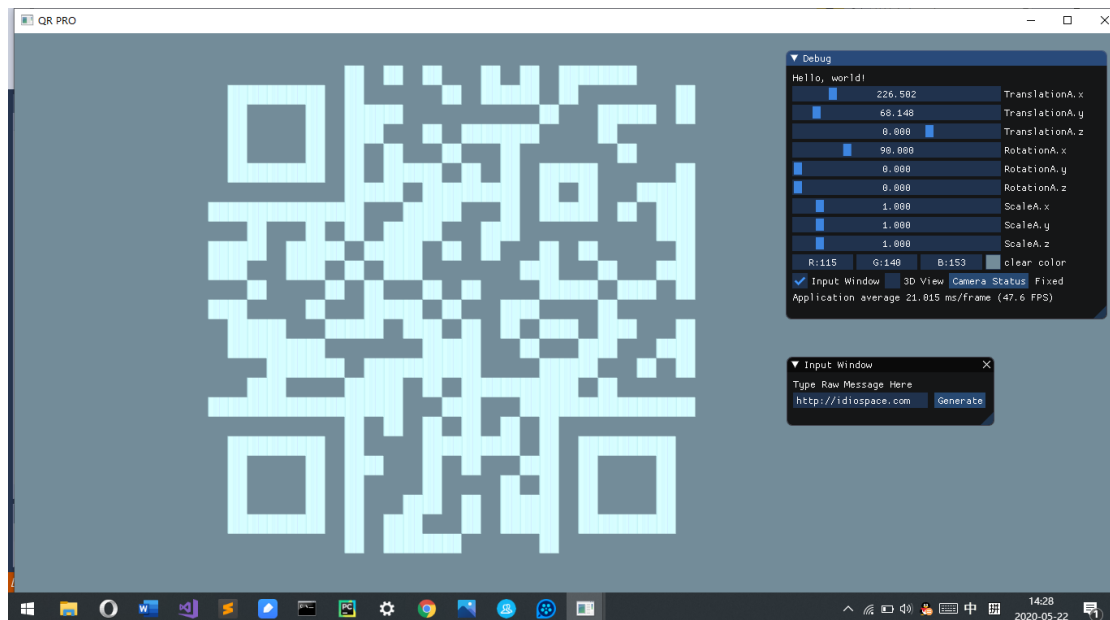
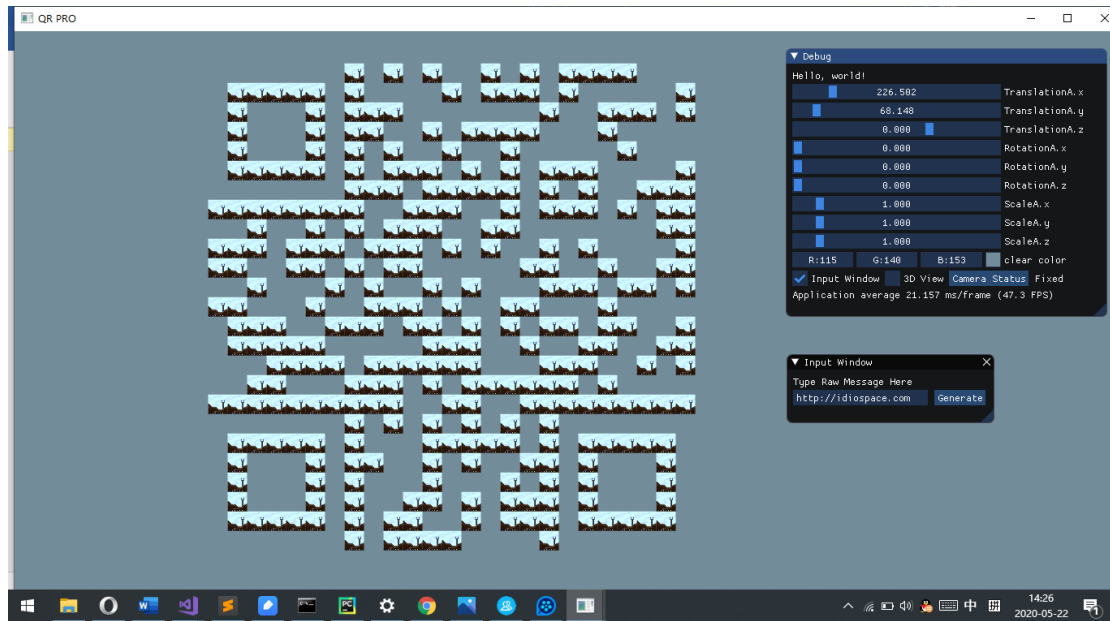
接着编写相机系统并能够精确地通过 GUI 控制相机的行为，为后面的输出做出准备。

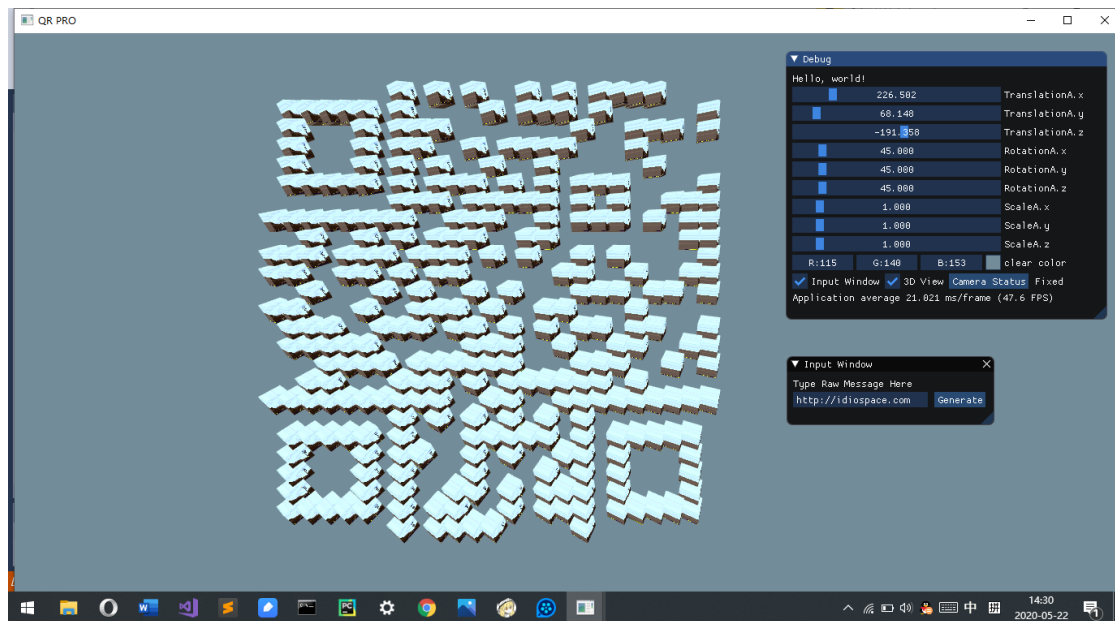



```
管理员: 命令提示符
(c) 2019 Microsoft Corporation. 保留所有权利。
C:\Users\Administrator>D:
D:\>cd code
D:\code>cd ComputerGraphicsFinalProject
D:\code\ComputerGraphicsFinalProject>git add -A
D:\code\ComputerGraphicsFinalProject>git commit -m "3D / 2D switch stable"
[master 6c9233f] 3D / 2D switch stable
 36 files changed, 143 insertions(+), 53 deletions(-)
 rewrite .vs/ComputerGraphicsFinalProject/v15/.suo (68%)
 create mode 100644 .vs/ComputerGraphicsFinalProject/v15/ipch/AutoPCH/25c83bf4ccf3f54b/MESH.ipch
 create mode 100644 ComputerGraphicsFinalProject/Mesh.cpp
 create mode 100644 ComputerGraphicsFinalProject/Mesh.h
 create mode 100644 ComputerGraphicsFinalProject/x64/Release/Mesh.obj
 rewrite x64/Release/ComputerGraphicsFinalProject.exe (62%)
 rewrite x64/Release/ComputerGraphicsFinalProject.ipdb (73%)
D:\code\ComputerGraphicsFinalProject>git relog
6c9233f HEAD@ {0}: commit: 3D / 2D switch stable
312b25b HEAD@ {1}: commit: Now turning to 3D
0a43edc HEAD@ {2}: commit: GUI statble
4f0d0be HEAD@ {3}: commit: Enable 2d texture drawing
1254312 HEAD@ {4}: commit (initial): Finished fundamentals with shaders, vao, vbo, index buffer, vertex array, layout abstraction
D:\code\ComputerGraphicsFinalProject>
```

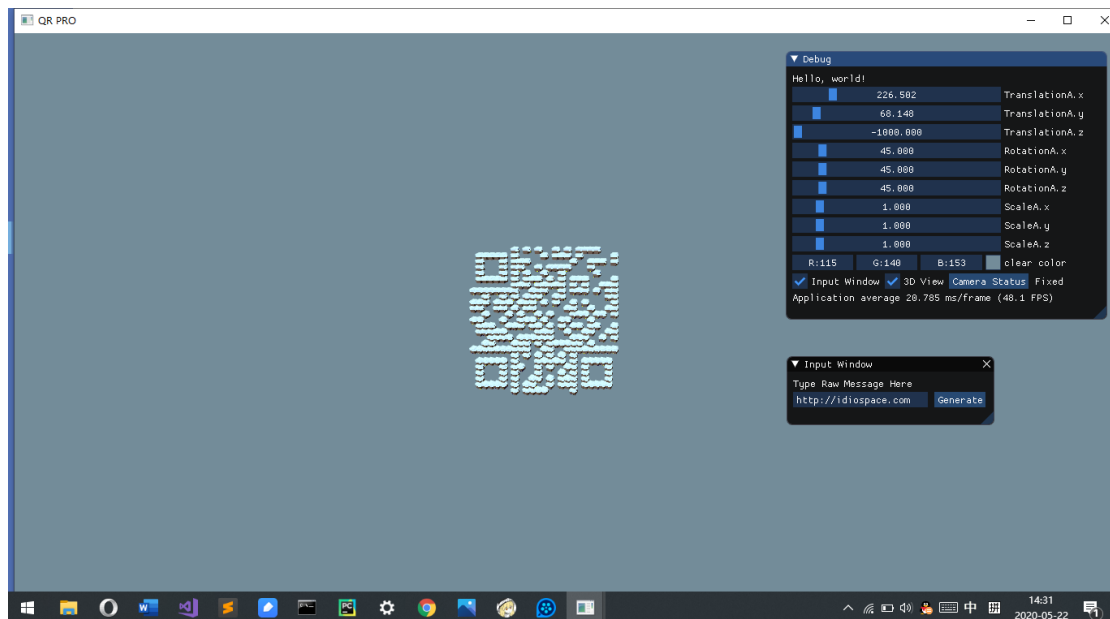
此时 3D 和 2D 的视角切换功能已经稳定, 开始编写摄像机, 摄像机需要同时考虑二维和三维的情况, 两种情况下视角不同。



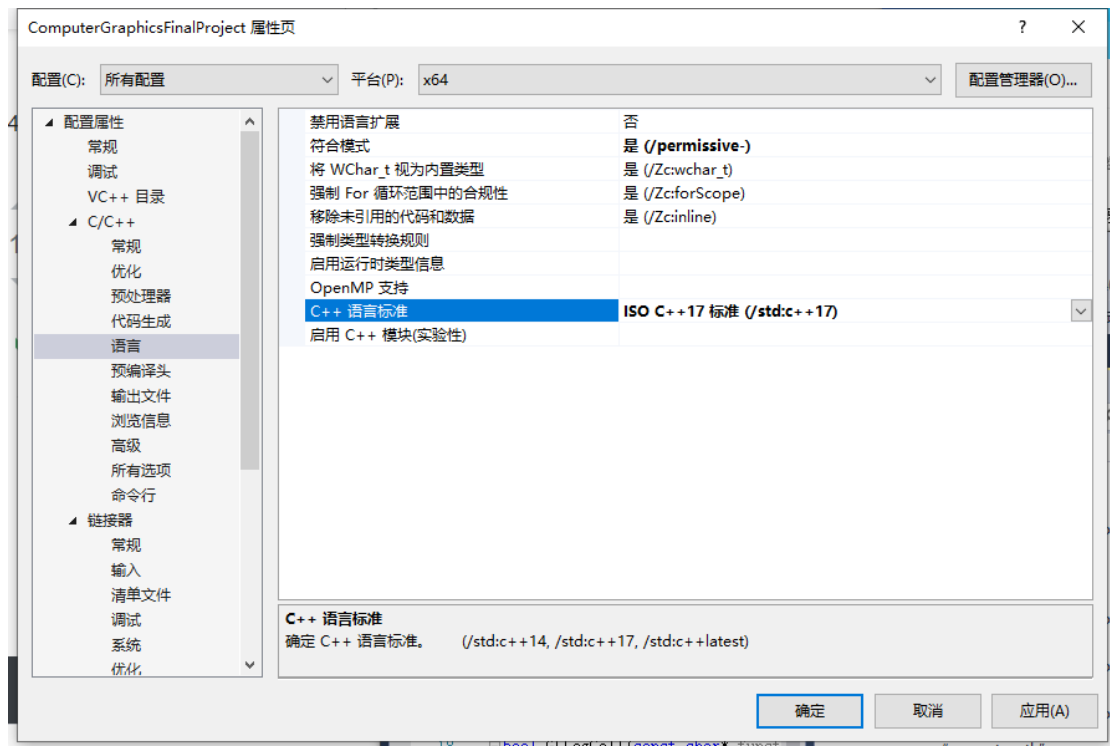




也可以自由地调整尺寸：



这一步需要构建一个完整的绘图的函数，能够在空间中自由精准地绘制目标物体。



在做 file browser 的时候编译一直无法通过，尝试更新 C++ 编译标准到 C++17。

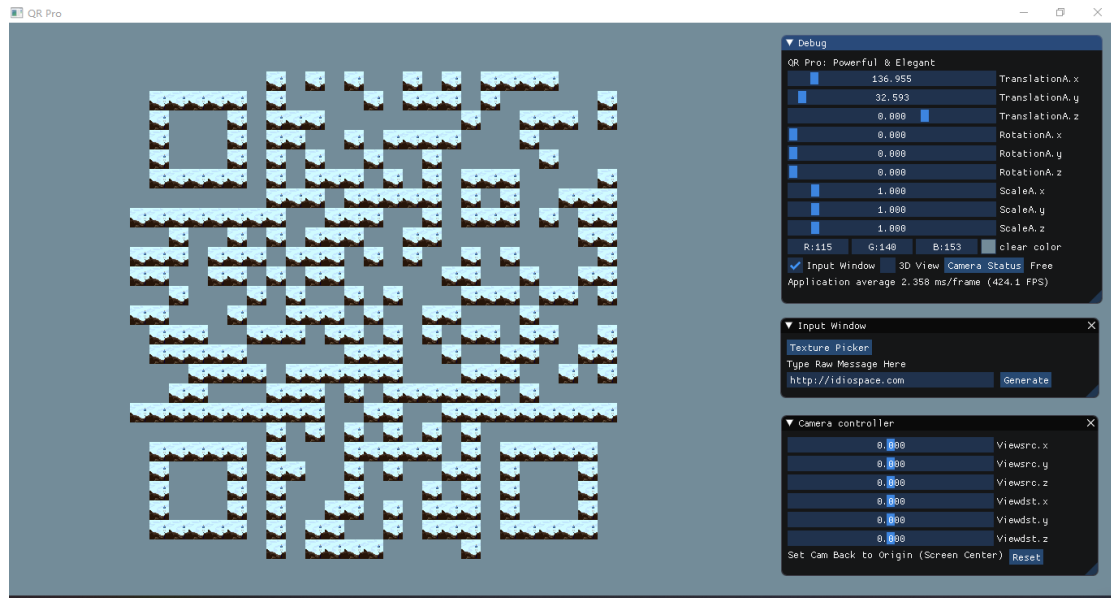
步骤 9：绑定二维码生成截屏到 GUI

设置回调函数，根据拿到的文本生成二维码。

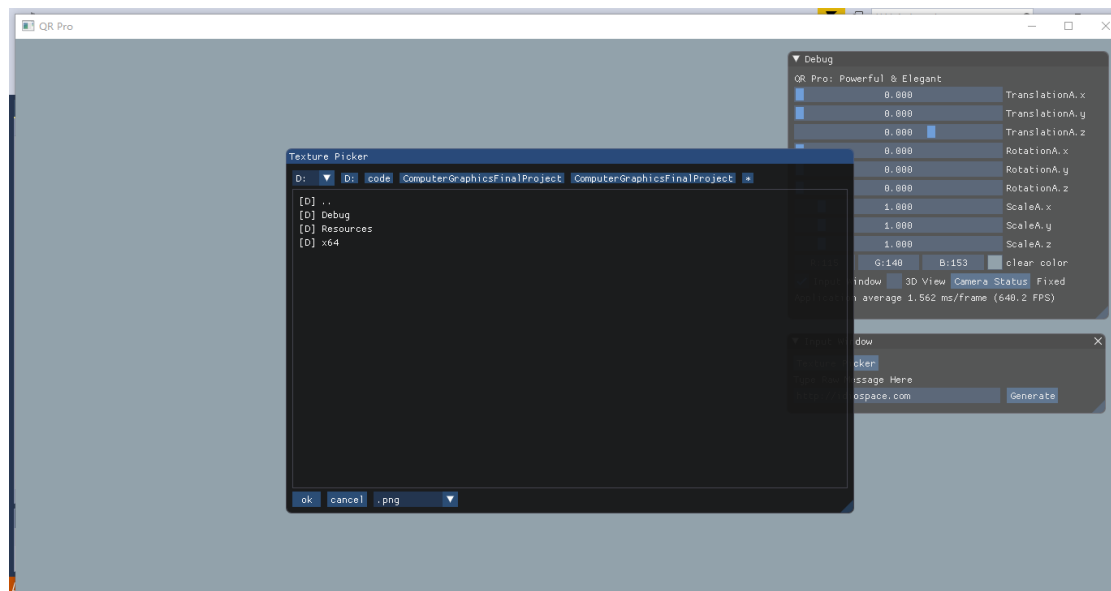
步骤 10：动态纹理更新

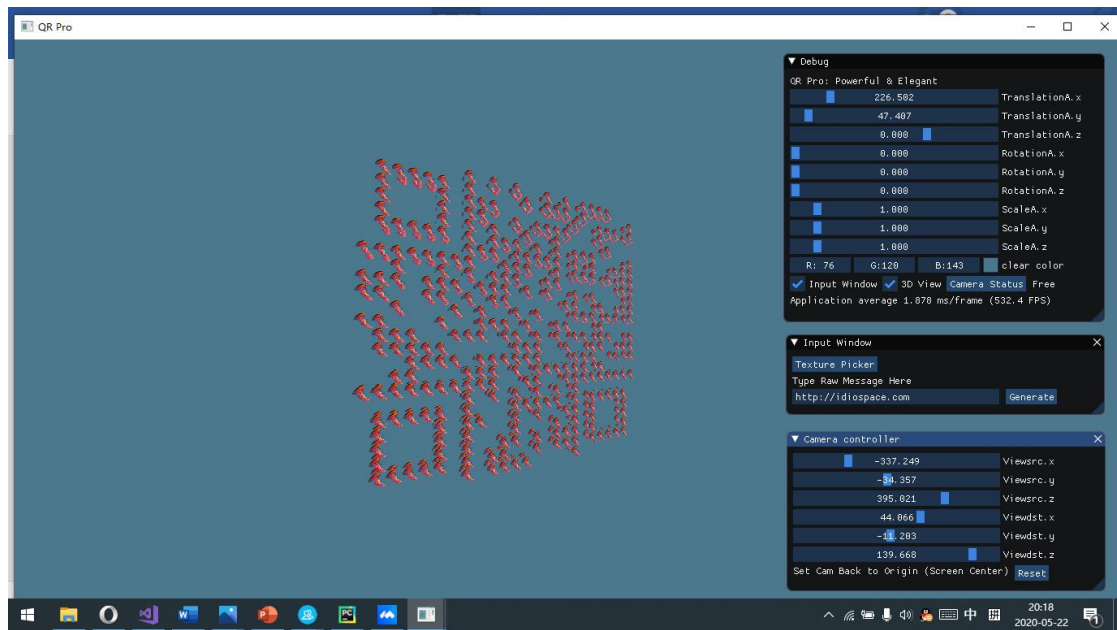
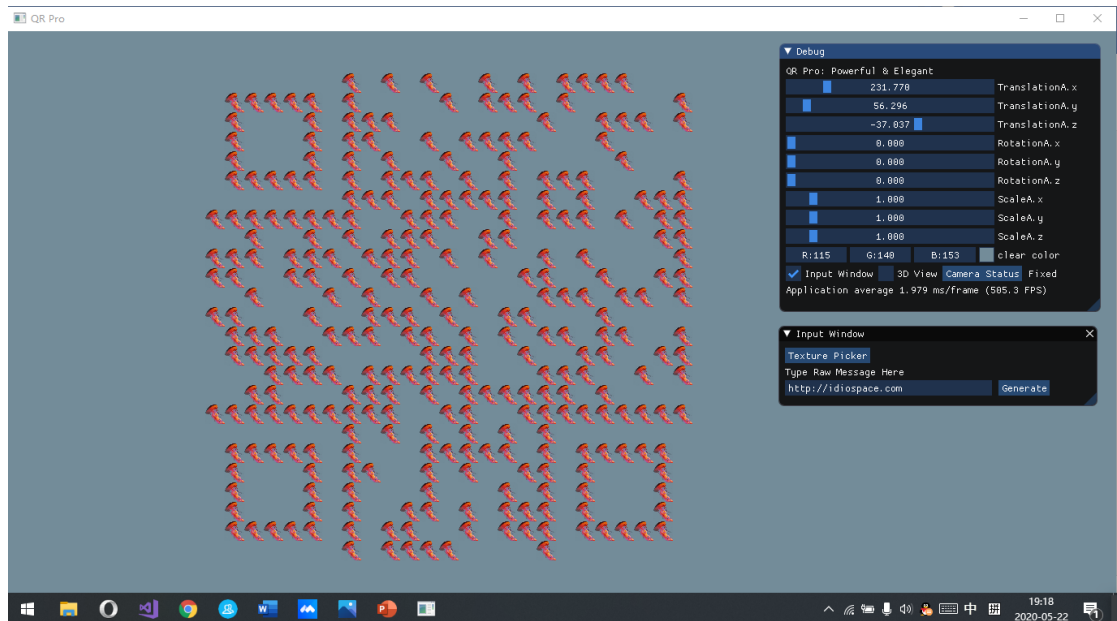
增加动态纹理更新功能，能够在渲染的过程中切换纹理。

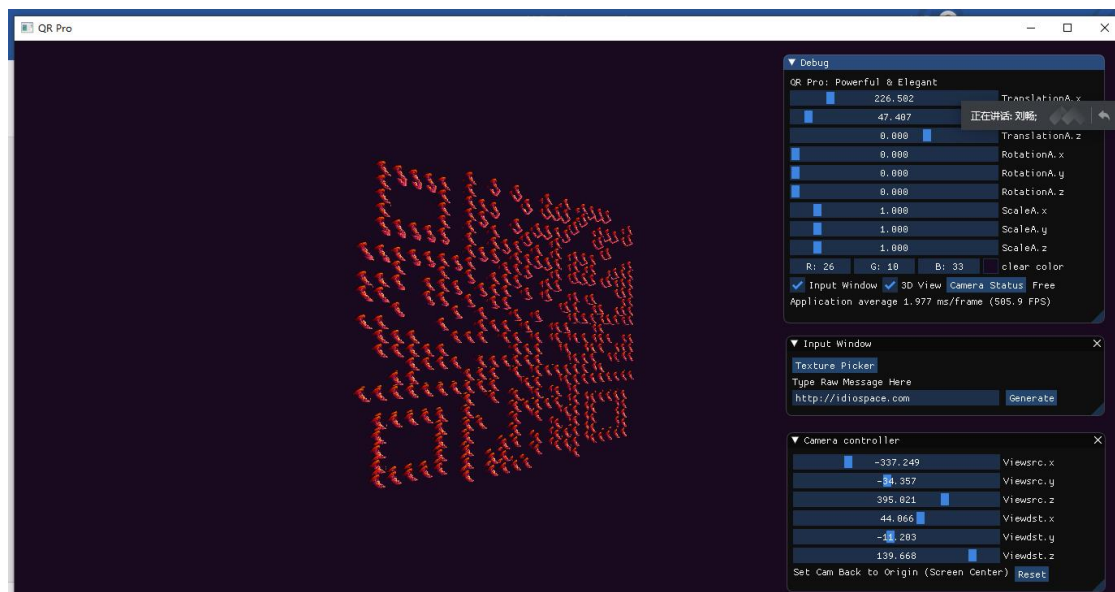
通过在 CPU 端预先加载多张纹理，提前发送到 GPU 端，在渲染的过程中，就可以使用之前封装好的 Texture 类中的 Unbind 和 Bind 进行快速切换，实现纹理的动态更新而不用退出。



在做好的 Texture Picker 中切换纹理：







附录

参考资料

1. <https://www.khronos.org/opengl/wiki/>
2. <http://docs.g>
3. SIGGRAPH University : "An Introduction to OpenGL Programming"
4. <https://www.jianshu.com/p/e556d0e65801> (Glad 安装)
5. https://www.youtube.com/watch?v=nVaQuNXueFw&list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2&index=22 (imgui 用法)
6. https://raw.githubusercontent.com/nothings/stb/master/stb_image.h (stb 库)
7. <https://www.youtube.com/watch?v=FBbPWSOQ0-w> (opengl Error Handling)