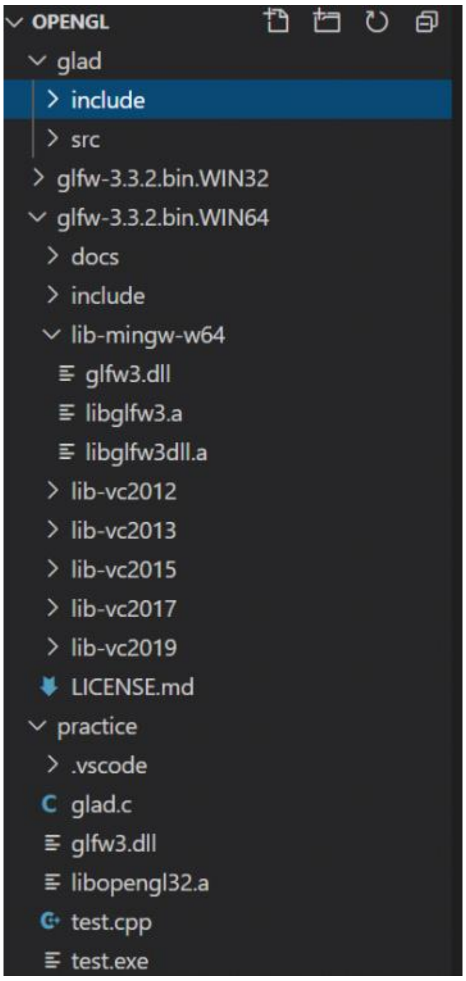
项目目录结构

Sunday, 01 November 2020 00:43

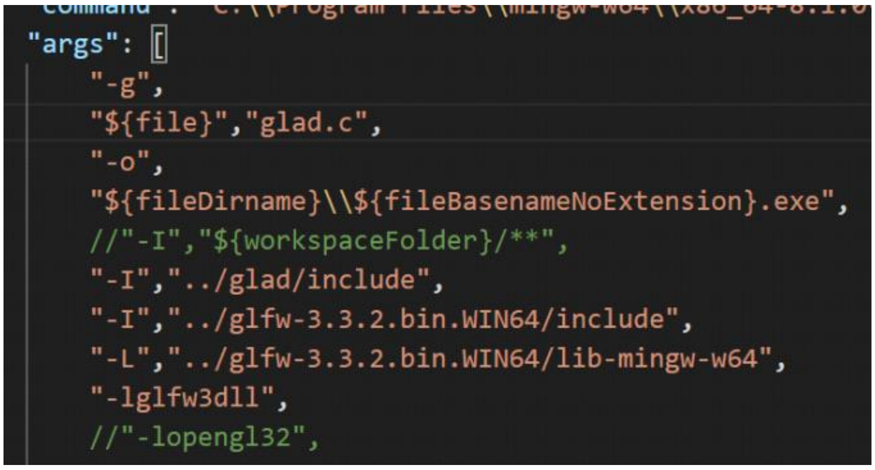


```
OPENGL
  glad
    include
    src
  glfw-3.3.2.bin.WIN32
  glfw-3.3.2.bin.WIN64
    docs
    include
    lib-mingw-w64
      glfw3.dll
      libglfw3.a
      libglfw3dll.a
    lib-vc2012
    lib-vc2013
    lib-vc2015
    lib-vc2017
    lib-vc2019
    LICENSE.md
  practice
    .vscode
    glad.c
    glfw3.dll
    libopengl32.a
    test.cpp
    test.exe
```

glfw相关：其中glfw.dll是在window运行时的动态库，libglfwdll.a是编译时用的动态库

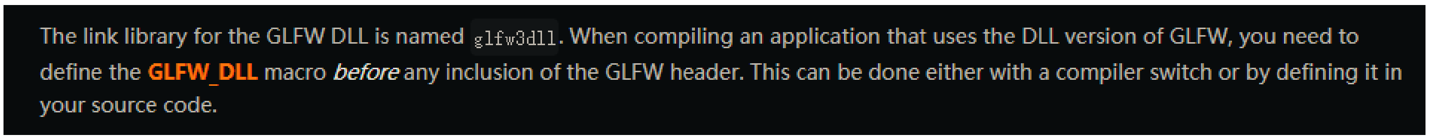
编译参数

2020年11月1日　　13:11



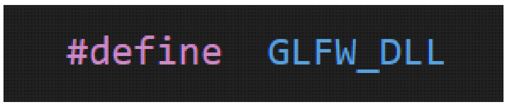
```
"command": "C:\\Program Files\\mingw-w64\\x86_64-8.1.0
"args": [
    "-g",
    "${file}","glad.c",
    "-o",
    "${fileDirname}\\${fileBasenameNoExtension}.exe",
    //"-I","${workspaceFolder}/**",
    "-I","../glad/include",
    "-I","../glfw-3.3.2.bin.WIN64/include",
    "-L","../glfw-3.3.2.bin.WIN64/lib-mingw-w64",
    "-lglfw3dll",
    //"-lopengl32",
```

注意这里的参数"-lglfw3dll"，而根据官方文档中提到



The link library for the GLFW DLL is named `glfw3dll`. When compiling an application that uses the DLL version of GLFW, you need to define the **GLFW_DLL** macro *before* any inclusion of the GLFW header. This can be done either with a compiler switch or by defining it in your source code.

于是在test.cpp最前面加上宏定义

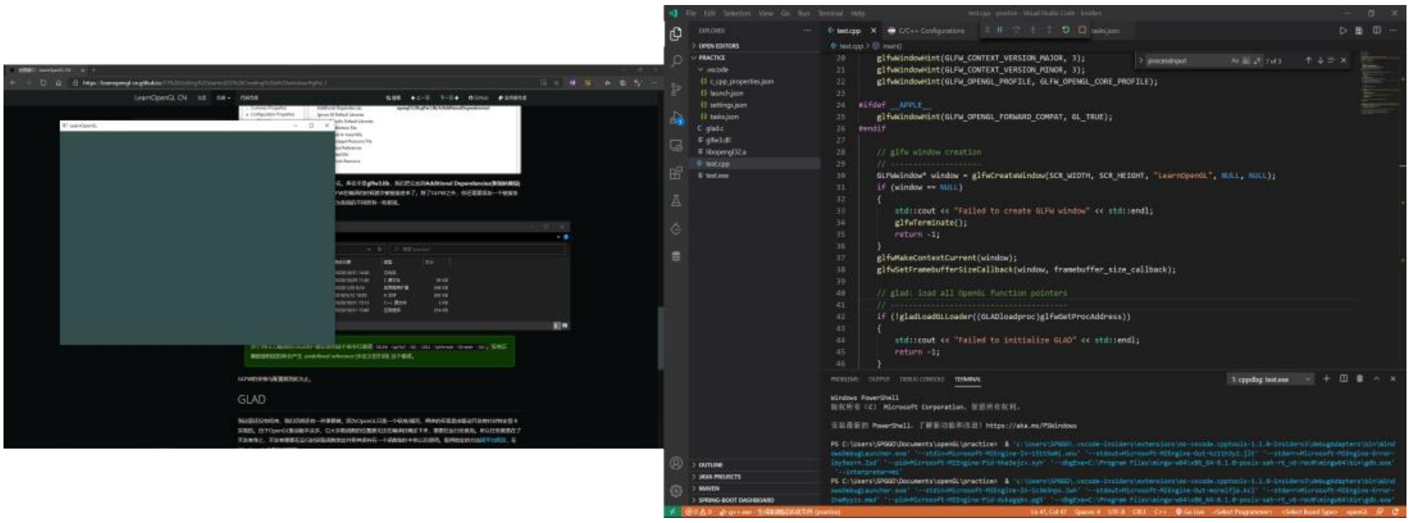


```
#define  GLFW_DLL
```

编译通过

2020年11月1日　　13:12

使用https://learnopengl.com/（下文称为官网）的代码，编译通过，左侧成功创建窗口

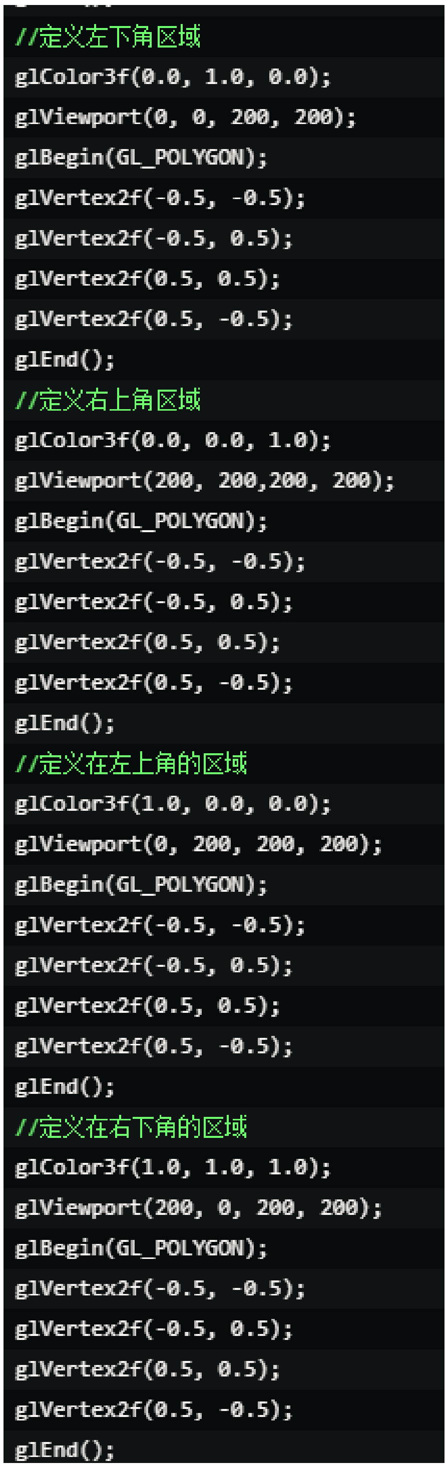


```cpp
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    // --------------------
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }
```

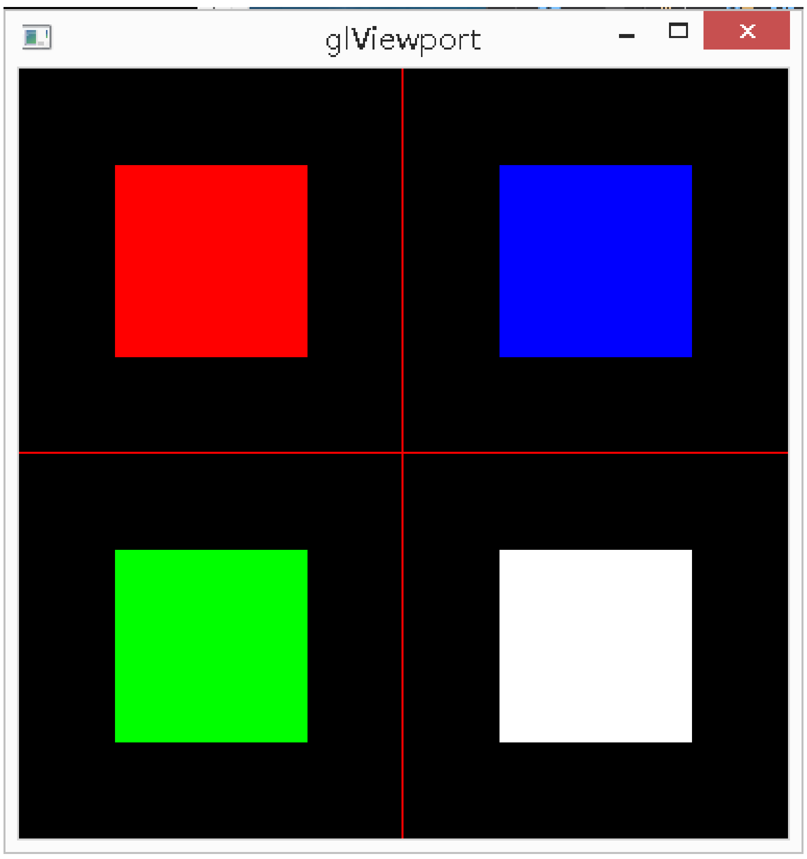
```
Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS C:\Users\SPGGO\Documents\openGL\practice> & [illegible]
PS C:\Users\SPGGO\Documents\openGL\practice> & [illegible]
```

Viewport

Sunday, 01 November 2020 00:44



```
//定义左下角区域
glColor3f(0.0, 1.0, 0.0);
glViewport(0, 0, 200, 200);
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
//定义右上角区域
glColor3f(0.0, 0.0, 1.0);
glViewport(200, 200,200, 200);
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
//定义在左上角的区域
glColor3f(1.0, 0.0, 0.0);
glViewport(0, 200, 200, 200);
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
//定义在右下角的区域
glColor3f(1.0, 1.0, 1.0);
glViewport(200, 0, 200, 200);
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
```

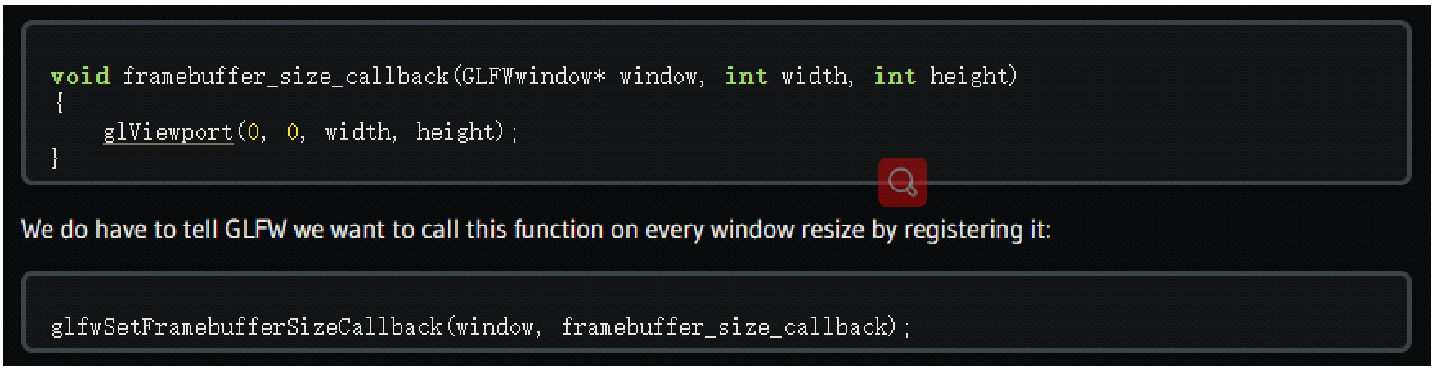


如上，一个窗口（window）里面可包含多个视口（viewport），以红框框区域为例，它的四个参数（0，200，200，200），前两个表示红框框区域左下角，后两个表示长宽（窗口大小为400*400）。而在一个viewport中，坐标的范围是（-1，1），即红框框区域的中心点是（0，0），那红方块四个顶点（vertex）的坐标就是0.5的那些东西。

Register a Callback Function

2020年11月1日 13:14

在glfw中，可以注册许多回调函数，比如



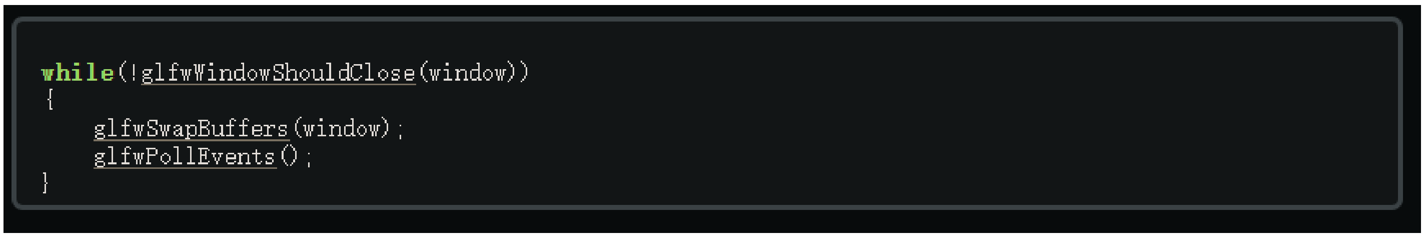
```
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

We do have to tell GLFW we want to call this function on every window resize by registering it:

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
```

第一个函数是在窗口大小发生变化时的回调函数，第二个则是向glfw注册第一个回调函数。

除此以外，还可以注册由键盘或者鼠标触发的回调函数，而这些函数往往被如下调用



```
while(!glfwWindowShouldClose(window))
{
    glfwSwapBuffers(window);
    glfwPollEvents();
}
```

其中glfwPollEvents()函数就是检查是否有注册的函数对应的事件发生（类似中断查询）。这个整个while循环用于一直渲染窗口，使窗口保持open。

Double Buffer

2020年11月1日 13:14

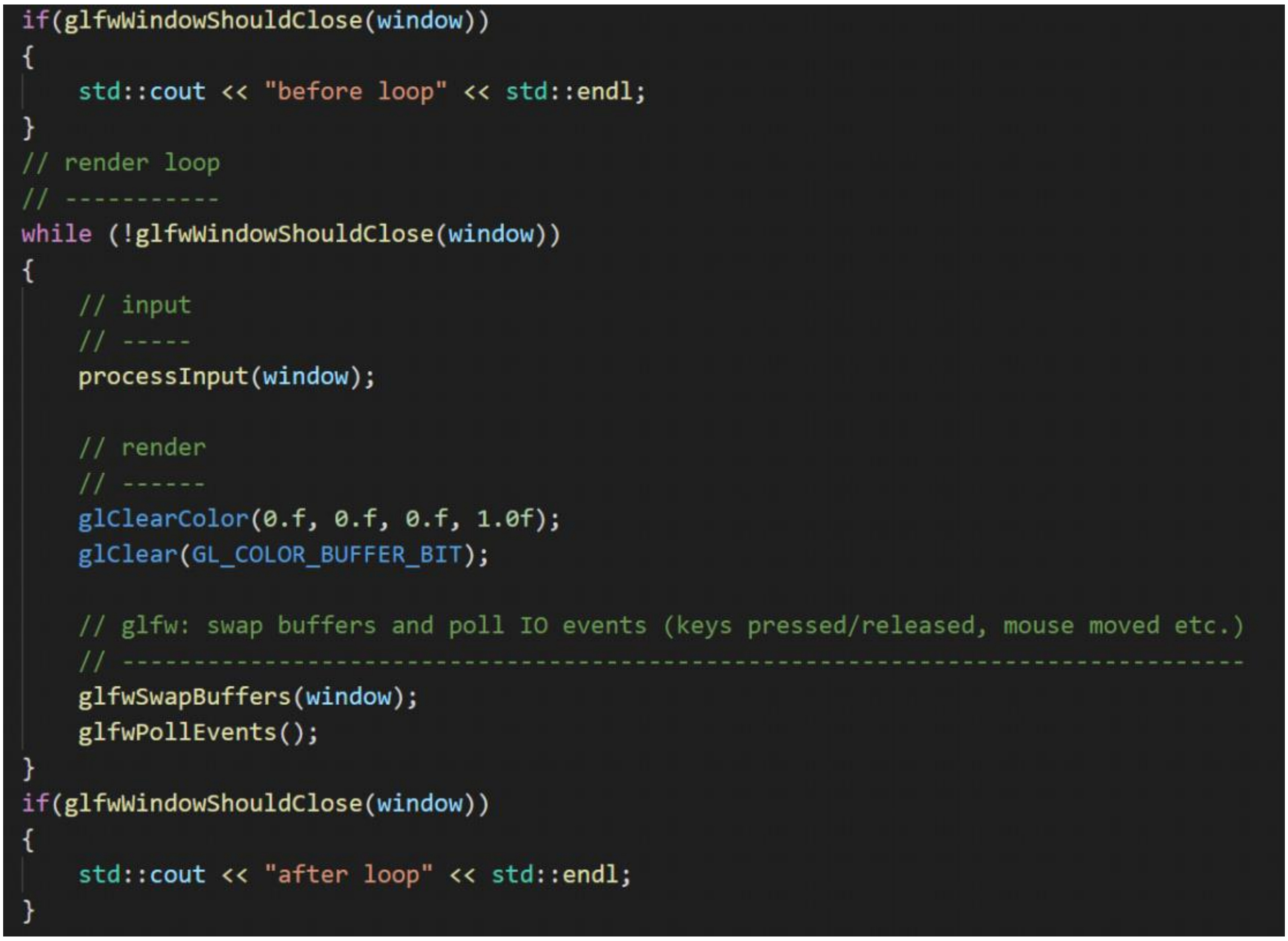
When an application draws in a single buffer the resulting image may display flickering issues. This is because the resulting output image is not drawn in an instant, but drawn pixel by pixel and usually from left to right and top to bottom. Because this image is not displayed at an instant to the user while still being rendered to, the result may contain artifacts. To circumvent these issues, windowing applications apply a double buffer for rendering.
The front buffer contains the final output image that is shown at the screen, while all the rendering commands draw to the back buffer. As soon as all the rendering commands are finished we swap the back buffer to the front buffer so the image can be displayed without still being rendered to, removing all the aforementioned artifacts

Close Window

2020年11月1日 13:14

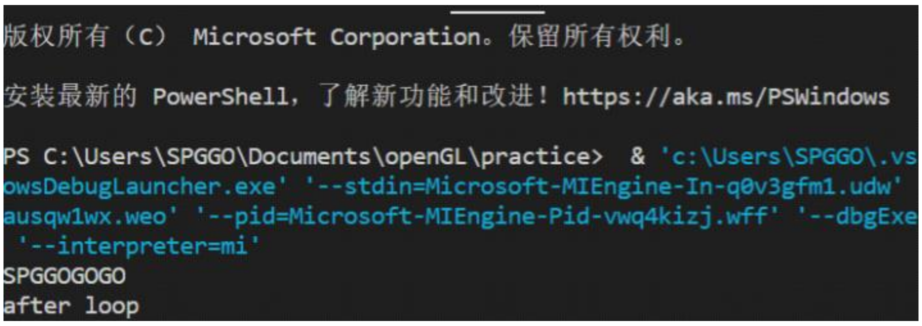
在点击“X”关闭窗口之后，实际上窗口并不会直接关闭，而是经由
glfwWindowShouldClose(window)
函数判断出，本窗口已经被点击了关闭，“should close”，然后退出render loop 的while循环，再调用
glfwTerminate()
释放资源，最后关闭窗口

测试如下：



```
if(glfwWindowShouldClose(window))
{
    std::cout << "before loop" << std::endl;
}
// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // input
    // -----
    processInput(window);

    // render
    // ------
    glClearColor(0.f, 0.f, 0.f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
    // -------------------------------------------------------------------------------
    glfwSwapBuffers(window);
    glfwPollEvents();
}
if(glfwWindowShouldClose(window))
{
    std::cout << "after loop" << std::endl;
}
```

输出如下



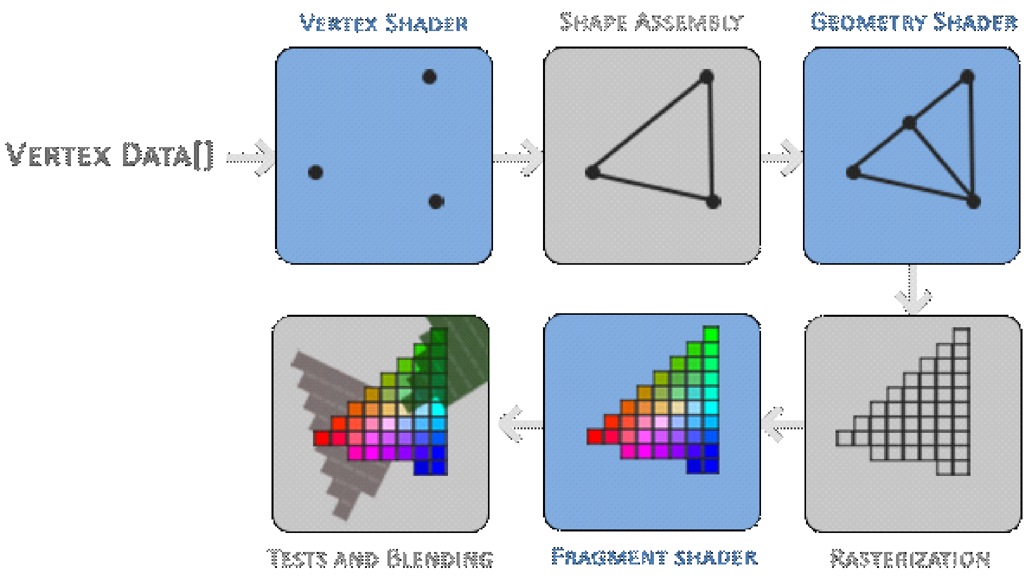
```
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS C:\Users\SPGGO\Documents\openGL\practice>  & 'c:\Users\SPGGO\.vs
owsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-q0v3gfm1.udw'
ausqw1wx.weo' '--pid=Microsoft-MIEngine-Pid-vwq4kizj.wff' '--dbgExe
 '--interpreter=mi'
SPGGOGOGO
after loop
```

由上可知，在没有点击“X”的情况下，glfwWindowShouldClose(window) 返回false，不会打印“before loop”；而在点击了“X”之后，glfwWindowShouldClose(window)返回true，于是退出while循环，并且打印“after loop”

Pipeline and Shader

2020年11月1日　　16:01

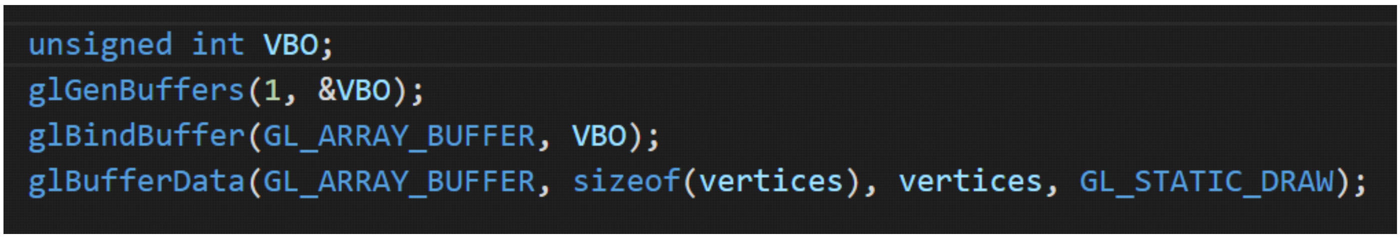
一个pipeline内有许多shader，一个shader负责一个具体的渲染步骤，经过所有的shader着色之后，一个pipeline就完成了对一个图像的渲染（加工）。下图中的整个过程就是在pipeline内完成的，或者说这就是一个pipeline，而每一个步骤就是一个shader。其中蓝色的就代表可以我们自定义的shader，其中最常见的是vertex shder 和 fragment shader。
(there are no default vertex/fragment shaders on the GPU



- Vertex shader 将顶点的三维坐标转化
- Assemble shader 根据指定的primitive shape组装顶点
- Geometry shader 将图形复杂化
- Rasterization shader 使图形光栅化，并且修剪多余的像素
- Fragment shader 给单个图形每一个像素确定颜色（根据颜色、不透明度、光照、阴影等）
- alpha test and blending shader 将多个图形堆叠在一起后最终每个像素的颜色

Vertex input

Vertex shader需要的数据输入就是点坐标的集合，每一个点都是三维的坐标，且范围在（-1，1），称为normalized device coordinates。
我们通过管理一块名为 vertex buffer objects (VBO) 的内存来存放点坐标数据，而Vertex shader也从这块内存中读入数据。下图就是创建一个VBO到存入数据的全部过程。通过 glBindBuffer(GL_ARRAY_BUFFER, VBO) 绑定VBO，且之后所有对于类型为GL_ARRAY_BUFFER的buffer的操作都是对这个VBO的操作



```
unsigned int VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Vertex shader