计算机图形学实验报告

学号：16340054

姓名：戴馨乐

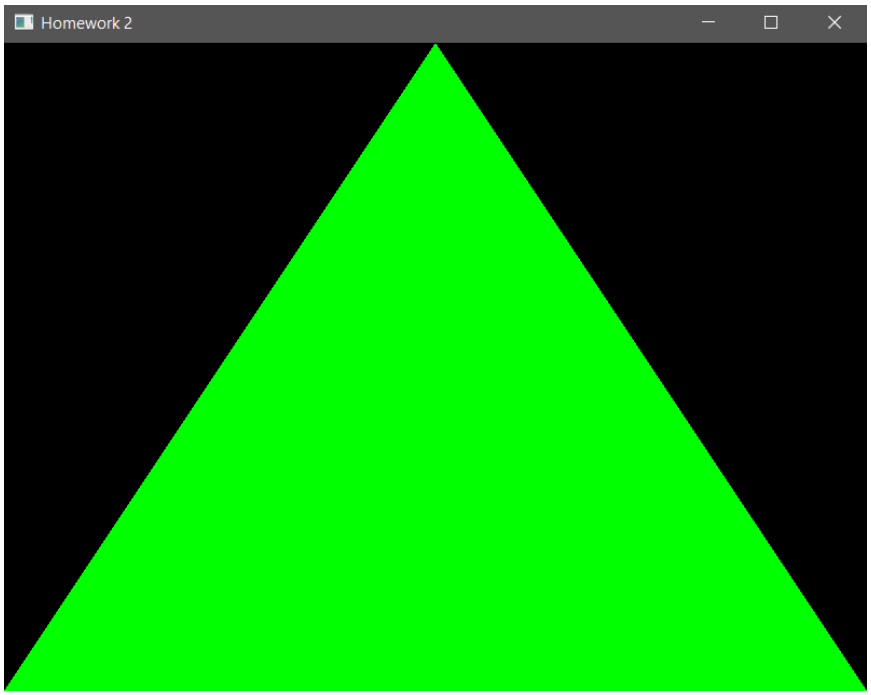
学院：数据科学与计算机学院

作业：Homework2

Basic：

1. 使用 OpenGL(3.3 及以上)+GLFW 或 freeglut 画一个简单的三角形

   a) 运行截图：



   b) 实现思路：

   首先是基本的设置着色器程序，其中包括编写编译顶点着色器和片段着色器，然后附加链接到一个着色器程序上供我们使用；

然后将 3 个顶点以及顶点的颜色的坐标用数组表示：

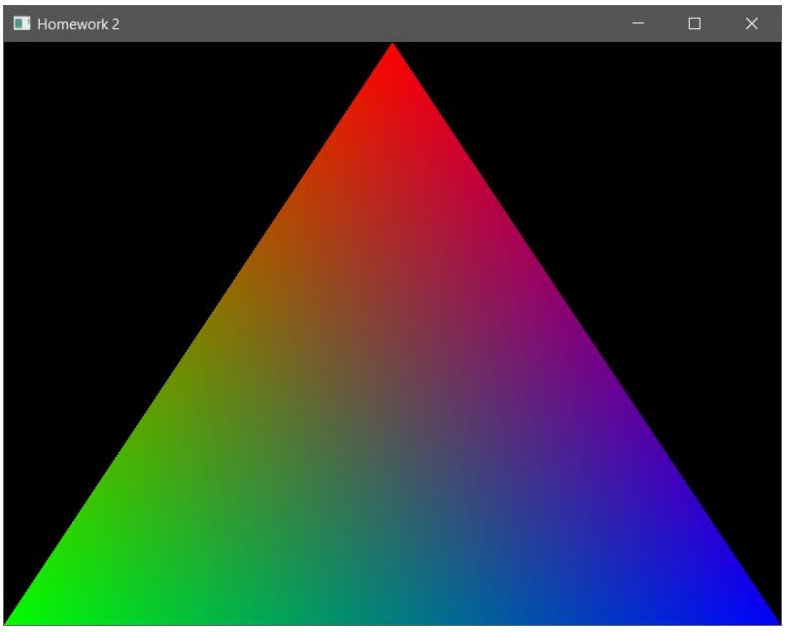
```
float vertices[] = {
    // 位置
     0.0f,  1.0f, 0.0f, 0.0f, 1.0f, 0.0f, // 右下
    -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, // 左下
     1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f  // 顶部
};
```

然后生成 VAO 顶点数组对象，告诉 openGL 要怎么处理点；然后将数据绑定到生成的 VBO 顶点缓冲对象上。最后设置顶点的指针，使得 openGL 可以正确处理这些顶点以及顶点属性。

```
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
```

2. 对三角形的三个顶点分别改为红绿蓝，像下面这样。并解释为什么会出现这样的结果。

   a) 运行截图：



b) 解释：

这里我们设置了 3 个顶点的颜色为红，绿，蓝，但是整个三角形却出现了调色板这样的效果，这是因为片段着色器在上色时候进行片段插值的结果。例如一个点在一个线段上，它和红色的距离为线段的 70%，与绿色的距离为 30%，那么，这个点就是 70%的红色与 30%的绿色混合而成。以此类推可以知道这个三角形中调色板的效果，其实就是不同点在不同位置进行片段插值的结果。

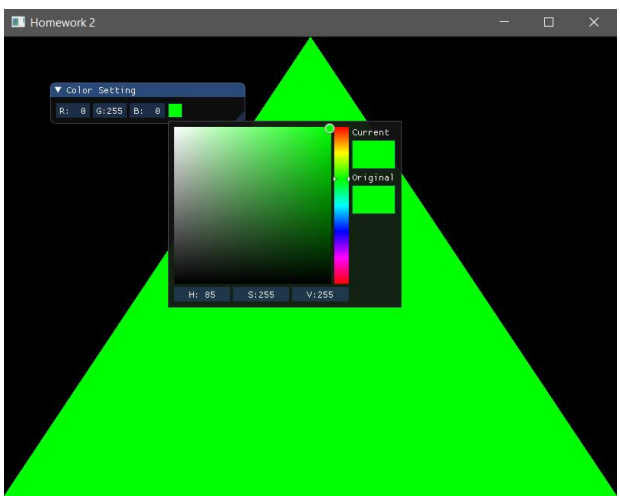
c) 实现思路：

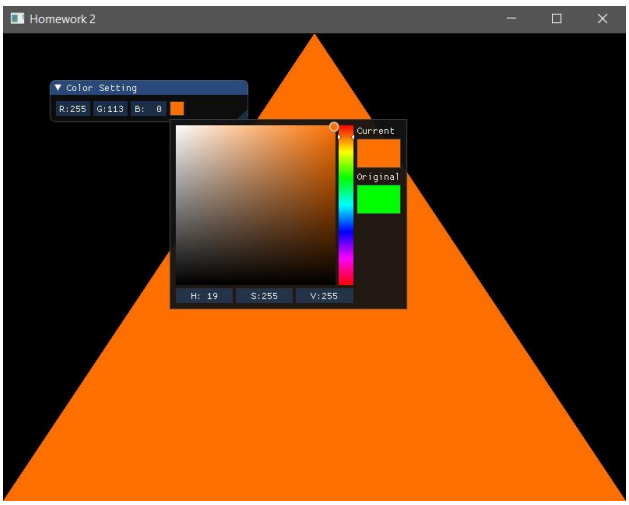
这里和 Basic 1 很类似，仅仅需要的修改是将顶点颜色改为红绿蓝

```
float vertices[] = {
    // 位置
     0.0f,  1.0f, 0.0f, 1.0f, 0.0f, 0.0f, // 右下
    -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f, // 左下
     1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f  // 顶部
};
```

3. 给上述工作添加一个 GUI，里面有一个菜单栏，使得可以选择并改变三角形的颜色。

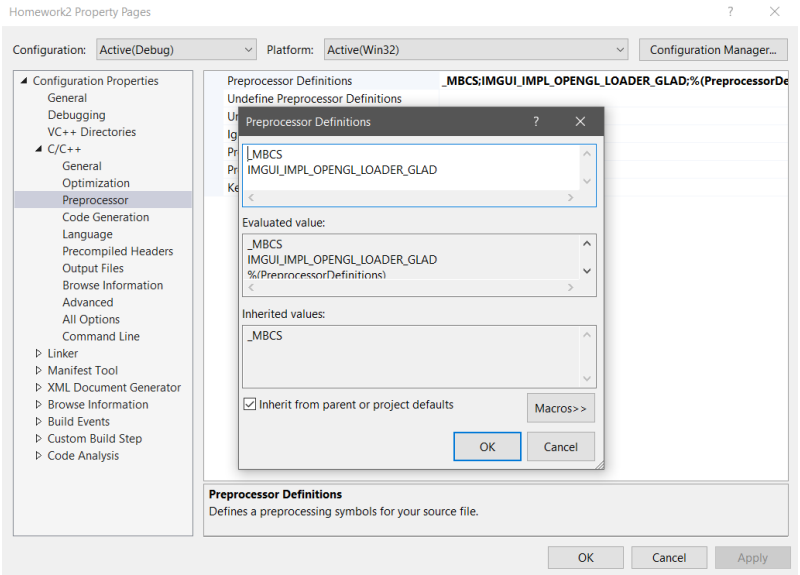
a) 运行截图：





b) 实现思路：

首先是要配置 ImGui，这个配置相对简单，只需要将需要 include 的头文件加入 vs 的头文件库，然后将对应的.cpp 文件添加进项目即可。唯一需要注意的是，ImGui 默认使用 gl3w 而不是 glad，故需要加上下面这一句：



然后就是模仿 example 里面的 glfw_opengl 那部分来学习 ImGui 了

先和 Task1 那样画好三角形，然后初始化 ImGui

```
#if __APPLE__
    const char* version = "#version 150";
#else
    const char* version = "#version 130";
#endif
    // 设置ImGui的上下文
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;
    // 设置ImGui的样式
    ImGui::StyleColorsDark();
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init(version);
```

之后，便可以在渲染循环中，使用这个组件了

```
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
{
    ImGui::Begin("Color Setting
    ImGui::ColorEdit3("", (float*)&triangleColor); // Ed
    ImGui::End();
}
change_triangle_color(shaderProgram, triangleColor);
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 3);
ImGui::Render();
ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
```

可以看到，框出来的部分，就是我们设置了 ImGui 面板上有什么内容，这次作业我们只需要调色板，故只用了 ColorEdit3 这个控件，其中 3 代表得到的是一个 RGB 的 vec3 的代表颜色的向量。

但是这里仅仅只是得到了颜色，还需要能够修改三角形的颜色

首先，在片段着色器的代码中，我加入了一个 uniform4 的变量来存储颜色，uniform 变量代表全局变量，这样我就可以在选择颜色之后修改它

```
const char* fragmentShaderCode = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "uniform vec4 triangleColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = triangleColor;\n"
    "}\0";
```

上面代码我们看到选择了颜色后，执行了一个 change_triangle_color( ) 的函数，这个函数如下：

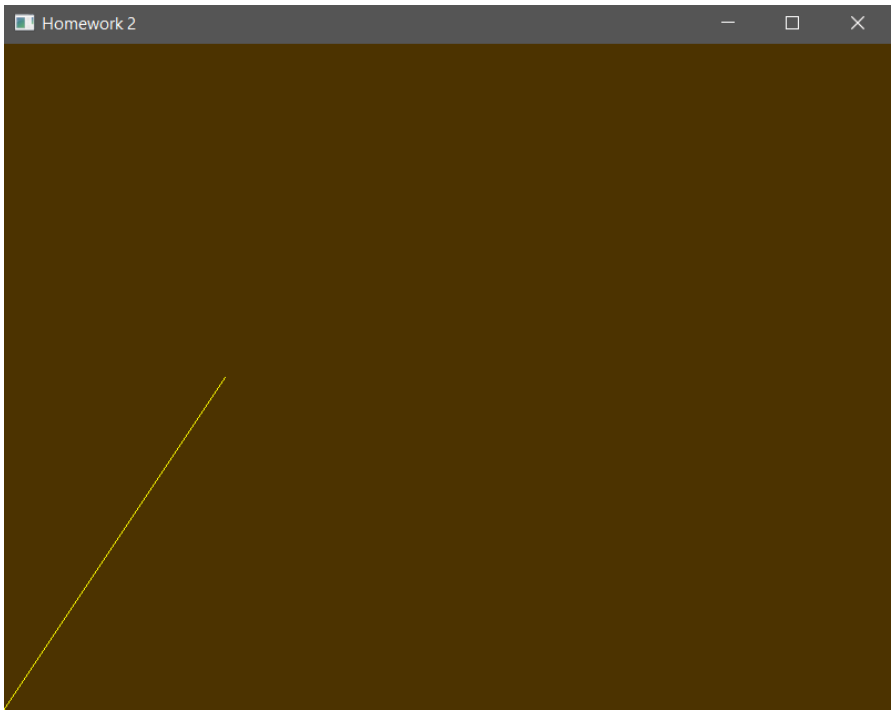
```
void change_triangle_color(unsigned int shaderProgram, ImVec4 triangleColor) {
    int vertexColorLocation = glGetUniformLocation(shaderProgram, "triangleColor");
    glUseProgram(shaderProgram);
    glUniform4f(vertexColorLocation, triangleColor.x, triangleColor.y, triangleColor.z, triangleColor.w);
}
```

这个函数主要做的事情就是，获取 uniform 变量，然后将选择颜色赋值给了 uniform 变量，这样子我们就成功修改了三角形的颜色。

Bonus：

1. 绘制其他的图元，除了三角形，还有点、线等

   a) 运行截图：



   b) 实现思路：

   需要改的地方是：

   顶点只用 2 个（使用了 EBO，在后面会说明）

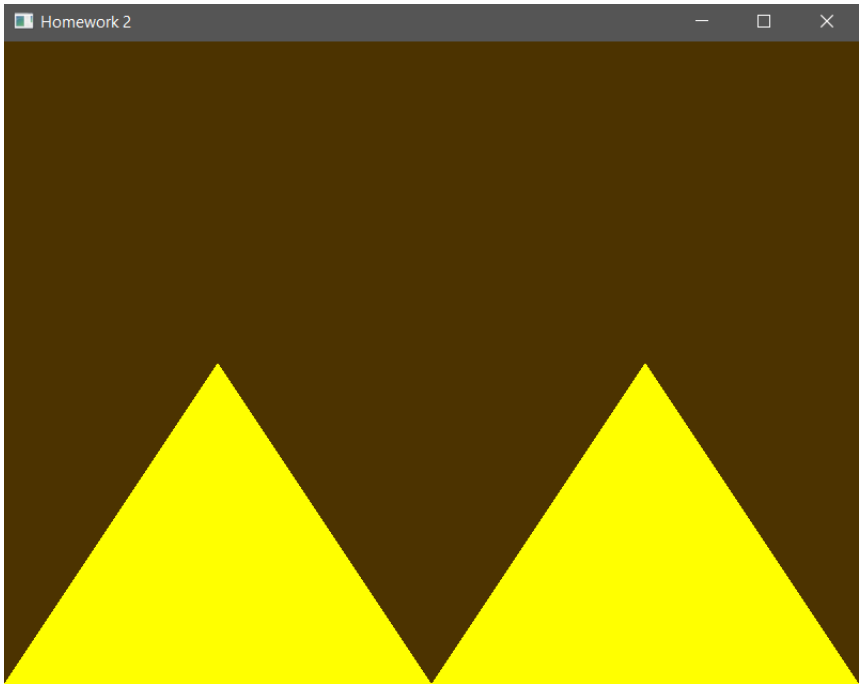
```
float vertices[] = {
    // 位置
    -0.5f,  0.0f, 0.0f, // 左中
    -1.0f, -1.0f, 0.0f, // 左下
     0.0f, -1.0f, 0.0f, // 正下
     0.5f,  0.0f, 0.0f, // 右中
     1.0f, -1.0f, 0.0f, // 右下
};

unsigned int indices[] = {
    0, 1
};
```

使用 2 个顶点，绘制线，类型为 GL_LINE_STRIP

```
glDrawElements(GL_LINE_STRIP, 2, GL_UNSIGNED_INT, 0);
```

2. 使用 EBO(Element Buffer Object)绘制多个三角形

a) 运行截图：



b) 实现思路：

EBO，索引缓冲对象，这里存储着顶点的索引。比如有一系列点，那么这些三角形由哪些点组成，这些信息保存在索引数组中；与 VBO 保存顶点数据类似，EBO 保存的是构成三角形的顶点的索引。

顶点和索引数据如下：

```
float vertices[] = {
    // 位置
    -0.5f,  0.0f, 0.0f, // 左中
    -1.0f, -1.0f, 0.0f, // 左下
     0.0f, -1.0f, 0.0f, // 正下
     0.5f,  0.0f, 0.0f, // 右中
     1.0f, -1.0f, 0.0f, // 右下
};

unsigned int indices[] = {
    0, 1, 2,
    2, 3, 4
};
```

可以看到，我们需要 2 个三角形，所以一共由 6 个索引，其中点 0，1，2 构成第一个三角形，点 2，3，4 构成第二个三角形。点数据就保存在顶点数组中。

然后，是需要配置并将索引数据附加到 EBO 对象上

```
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

可以看到这个过程和 VBO 很类似，只不过在绑定数据到缓冲区时候，需要指定类型为 GL_ELEMENT_ARRAY_BUFFER。

最后，在渲染循环中，使用 glDrawElements( )函数，便可以将 EBO 保存的索引中的所有三角形都画出来了。这里我们只画了 2 个，如果还需要多，只需要继续添加索引就可以了。

```
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```