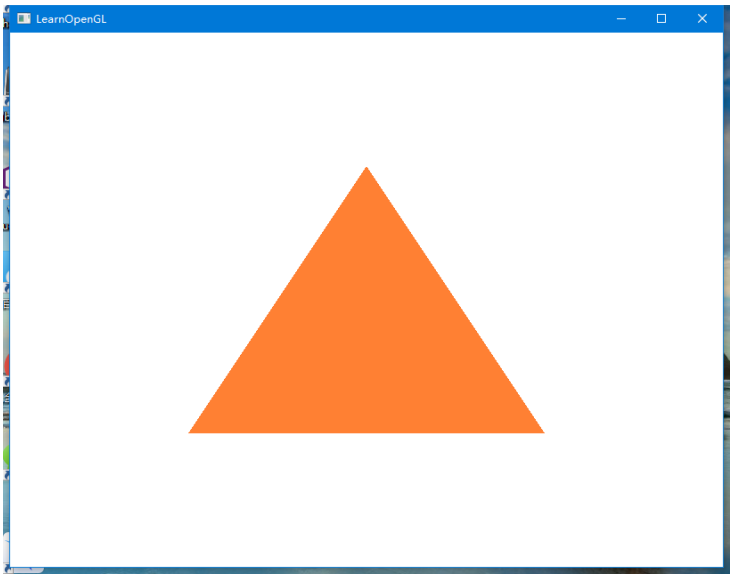
Homework 2 - GUI and Draw simple graphics

15331314 吴林蔓

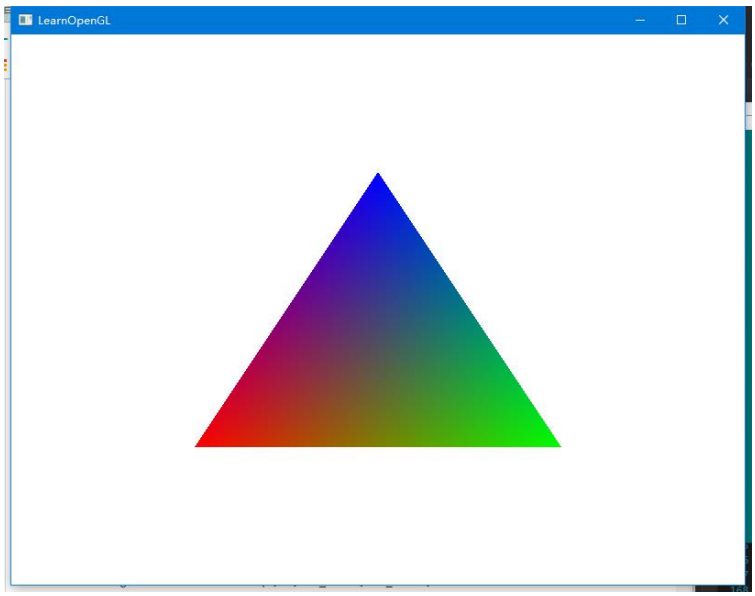
一、实验结果

Basic:

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



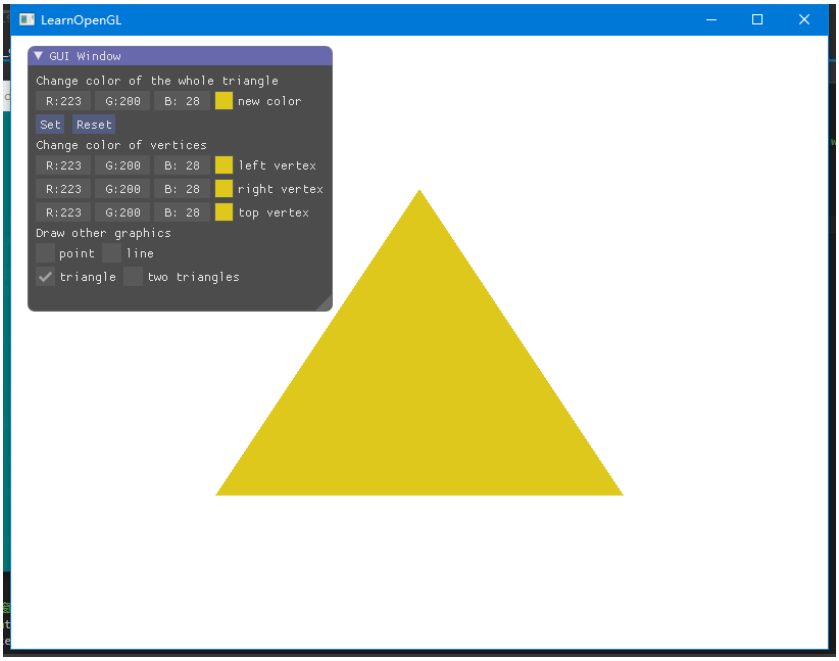
2. 对三角形的三个顶点分别改为红绿蓝。并解释为什么会出现这样的结果。



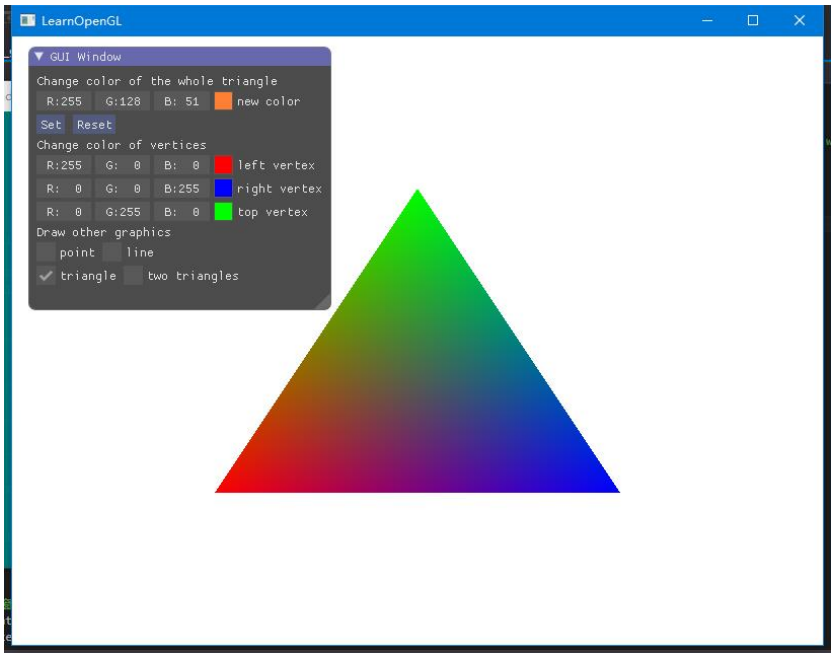
结果分析：我们分别将顶点设为了红绿蓝三种颜色，但在实际产生图像的过程中，会生成很多片元。而片元的颜色是通过片段着色器中的颜色插值（Color Interpolation）来生成的。其中一种方式是线性插值。如生成一个线段上的某位置的颜色，而该线段两个端点分别为红色和蓝色，则该位置颜色为 n 红色+(1+n)蓝色（n 为该位置离红色端点的距离/线段总长度）。

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。

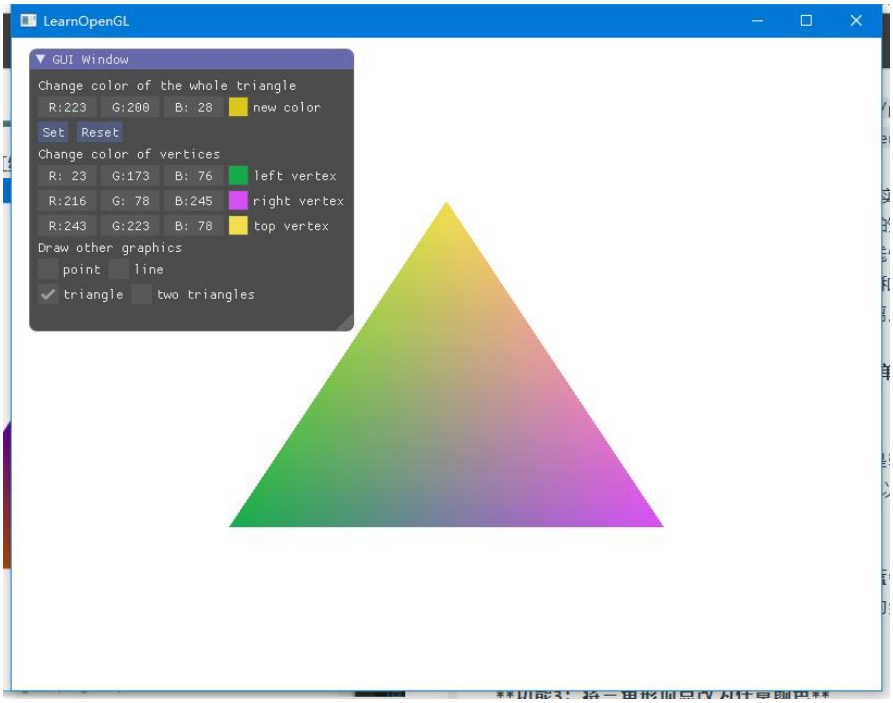
功能 1：把三角形改为任意一种颜色。 在 new color 选色框中选取任意一种颜色，或是输入颜色的 RGB 值，点击 set 按钮，即可改变整个三角形的颜色。具体操作可以看演示视频。



功能 2：设三角形三个定点颜色分别为红绿蓝 点击 reset 按钮，即可将三角形变为 basic2 里的红绿蓝。



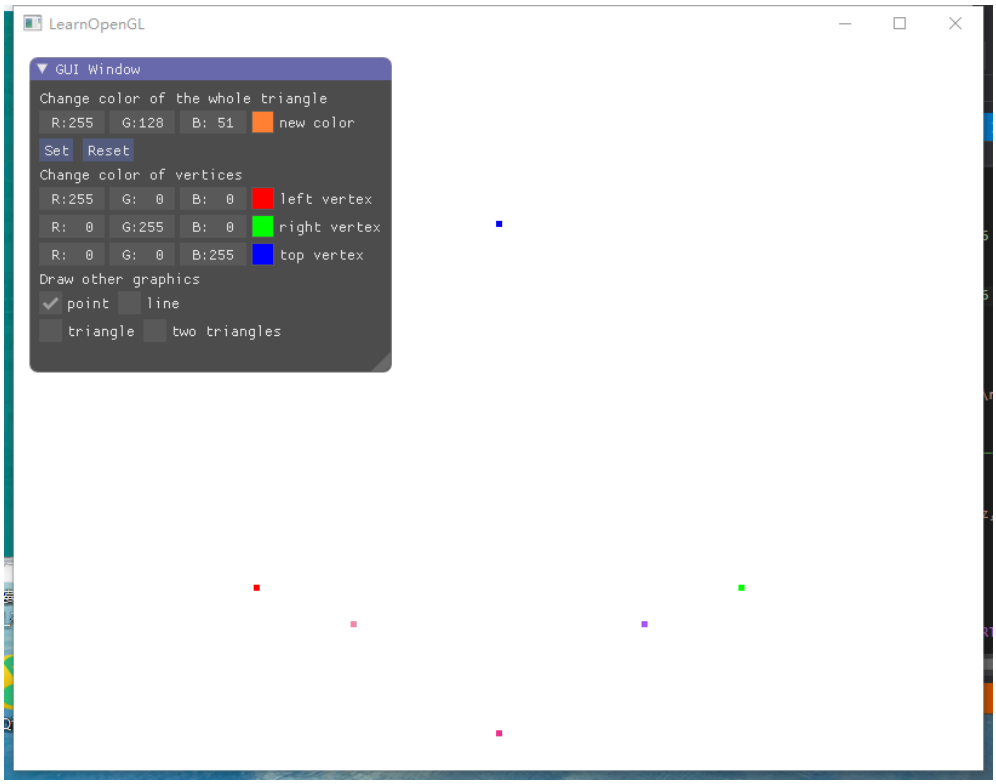
功能 3：将三角形顶点改为任意颜色 可在 set 和 reset 按钮下方的三个取色条中，分别选取三个顶点的颜色。三角形的颜色也会随之变化。



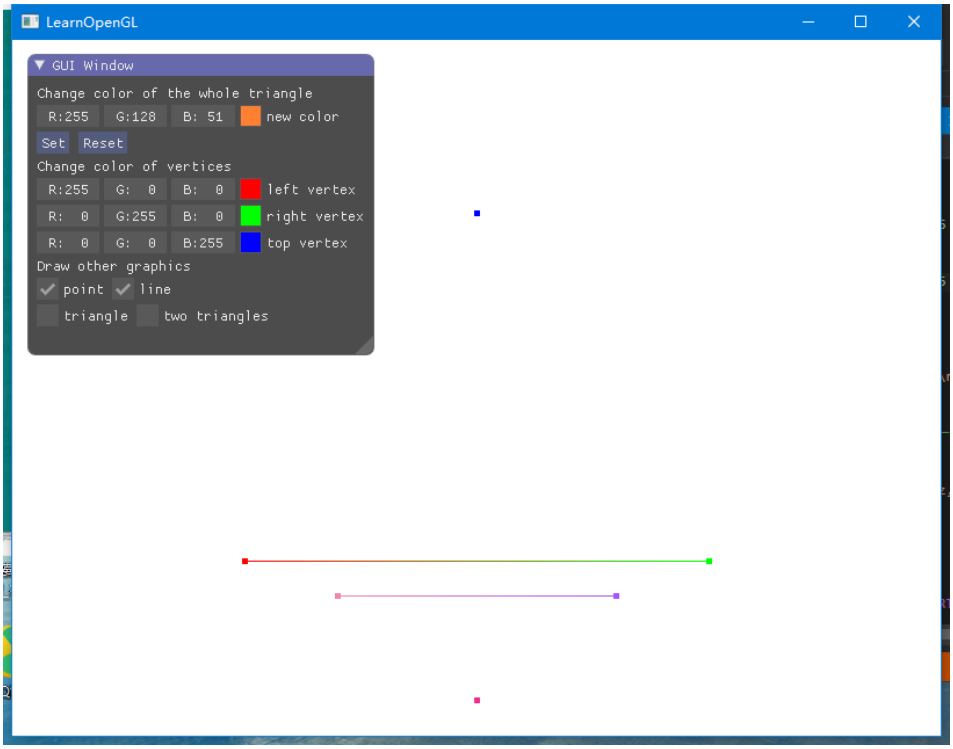
Bonus:

1. 绘制其他的图元，除了三角形，还有点、线等。 点击 Draw other graphics 菜单项中的 point 和 line 选项框，可分别绘制出点和线。

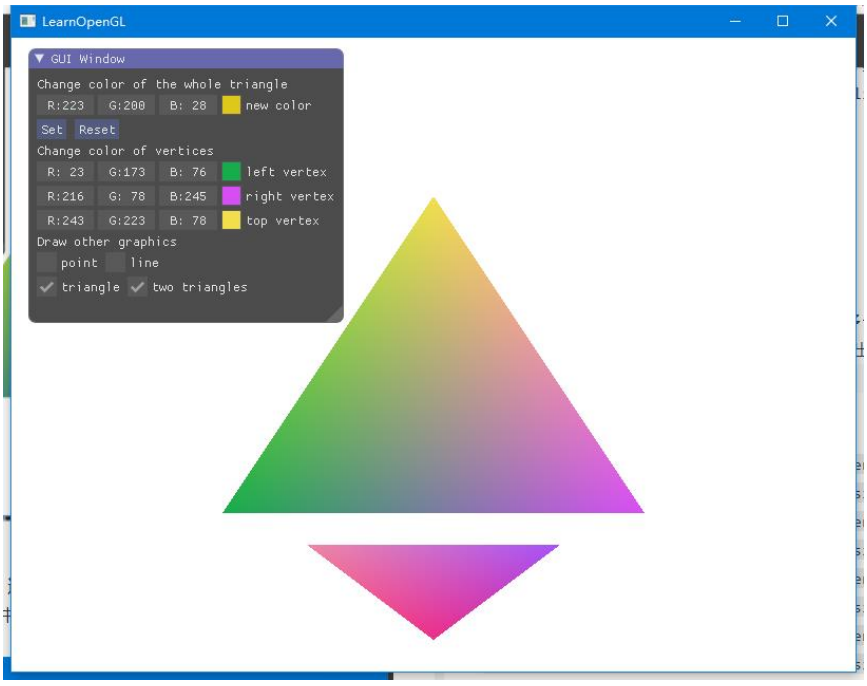
绘制点：



绘制线：



2. 使用 EBO(Element Buffer Object)绘制多个三角形。 点击 two triangles 选项框，可以利用 EBO 绘制出两个三角形。



二、实现思路

在此介绍一下实现本次任务的主要思路和流程，具体代码可参考 src 中的 `main.cpp`

1. 初始化 glfw 与 glad，创建窗口

2. 输入顶点具体数据，存放在 vertices 数组中

3. 创建并绑定 VAO、VBO，把顶点数组复制到缓存中供 OpenGL 使用。EBO 类似。

```
    // VAO
    unsigned int VAO;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    // VBO
    unsigned int VBO;
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW); //分配空间 传送数据
```

4. 设置顶点属性指针

分别设置位置和颜色属性，其中颜色属性中步长参数为 6

```
// vertex 位置属性
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
    (void*)0); glEnableVertexAttribArray(0);
    // vertex 颜色属性
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float),
(void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);
```

5. 实现着色器程序

创建顶点着色器和片段着色器，并编译着色器，最终将两个着色器程序链接起来。 其中顶点着色器的源代码如下，根据不同的属性位置值，提取顶点的位置和颜色属性。并将一个 vertexColor 变量输出，传入片段着色器：

```
const char *vertexShaderSrc = "#version 440 core\n"
        "layout (location = 0) in vec3 aPos;\n"
        "layout (location = 1) in vec3 Color;\n"
        "out vec3 vertexColor;\n" // 向片段着色器输出一个颜色
        "void main()\n"
        "{\n"
        "   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
        "   vertexColor = Color;\n"
        "}\0";
```

片段着色器源代码如下，接收来自顶点着色器的颜色值，设为对应片元的颜色：

```
const char *fragmentShaderSrc = "#version 440 core\n"
        "in vec3 vertexColor;\n"
        "out vec4 FragColor;\n"
        "void main()\n"
        "{\n"
        "   FragColor = vec4(vertexColor.x, vertexColor.y, vertexColor.z, 1.0f);\n"
        "}\0";
```

最后激活程序对象：

```
    glUseProgram(shaderProgram);
```

6. 实现渲染循环

在渲染循环中初始化缓冲区背景颜色、渲染物体并绘制图形，并在绘制结束后交换缓存等。点、线、三角形的绘制，使用到了 `glDrawArrays` 以及 `glDrawElements` 接口。一开始漏了重新绑定 VBO，就出现了无法更改三角形颜色的问题。

```
    // 渲染物体
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    // 重新绑定 VBO 将数据送入缓存
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

7. 设计 GUI

这次设计的 GUI 主要有 4 个功能：

①改变整个三角形颜色 ②改变三角形顶点颜色 ③绘制点或线 ④使用 EBO 绘制多个三角形

✧ 其中使用了 ImGui 中的 `ColorEdit3` 来实现颜色选取功能。选择的颜色会存放在第二个参数指向的地址空间中，如分别改变三角形三个顶点的颜色：

```
ImGui::ColorEdit3("left vertex", vertices + 3);
            ImGui::ColorEdit3("right vertex", vertices + 9);
            ImGui::ColorEdit3("top vertex", vertices + 15);
```

用这种方法，就能简洁又清楚地改变 vertices 数组中代表顶点颜色值的数据。避免了用循环去一个个赋值。

✧ 使用 `checkbox` 来选择绘制的图形。可以选择绘制 point、line、triangle 以及多个 triangle。通过判断该 checkbox 是否为 active，来决定是否绘制这个图形。绘制点的具体方法如下，因为画出来的点比较小，可通过 `glPointSize` 来使点更清楚。

```
if (draw_point) {
            glDrawArrays(GL_POINTS, 0, 3);
            glDrawArrays(GL_POINTS, 3, 3);
            glPointSize(5.0);
 }
```

8. 释放资源

结束渲染循环后，释放已使用的资源，如着色程序、顶点数组以及缓冲等。

三、个人心得

这次的作业是第一次尝试用 OpenGL 来实现图形学的一些基本操作，理解之后还算比较简单。但一开始接触真的一脸懵逼，也是看着官方教程一路摸索。在做每一个任务的过程中都会对前一个有更深的了解。但仍然存在着许多可以改进的地方：着色器程序部分还可以进行封装，显得代码更加简洁美观。还有就是 GUI 的设计其实是一项挺有意思的任务，可以通过观察 example 和尝试以获得设计灵感。最后，对 VAO、VBO 以及 EBO 之间关系、作用的了解还不够深入，应该继续思考和深入理解。