Homework 2 - GUI and Draw simple graphics

16340256 谢玮鸿

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

首先进行GLFW和GLAD的初始化，并创建窗口。

准备好**顶点着色器**和**片段着色器**的源码，存储在字符串中。

```
// 顶点着色器
const char *vertexShaderSource = "#version 330 core\n"
        "layout (location = 0) in vec3 aPos;\n"
        "out vec3 ourColor;\n"
        "void main() {\n"
        "   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
        "}\0";
// 片段着色器
const char *fragmentShaderSource = "#version 330 core\n"
        "out vec4 FragColor;\n"
        "in vec3 ourColor;\n"
        "void main() {\n"
        "   FragColor = vec4(ourColor, 1.0f);\n"
        "}\n\0";
```

然后可以创建着色器对象，并编译着色器源码。以顶点着色器对象为例：

```
    // 创建顶点着色器对象，编译着色器源码
        unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
        glCompileShader(vertexShader);
        // 检查着色器编译错误
        int success;
        char infoLog[512];
        glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
        if (!success)
        {
                glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
                std::cout << "编译顶点着色器源码出现错误\n" << infoLog << std::endl;
        }
```

使用`glCreateProgram()`创建着色器程序对象。着色器程序对象(Shader Program Object)是多个着色器合并之后并最终链接完成的版本。我们需要将刚才编译的着色器链接成为一个着色器程序对向，之后渲染对象时激活着色器程序。

```
    // 链接着色器
        unsigned int shaderProgram;
        shaderProgram = glCreateProgram(); // 创建一个着色器程序对象
        glAttachShader(shaderProgram, vertexShader);
        glAttachShader(shaderProgram, fragmentShader);
        glLinkProgram(shaderProgram);
```

完成上述准备工作后，就可以开始绘制三角形了。绘制简单的三角形要考虑三个顶点的位置。使用顶点缓冲对象 Vertex Buffer Objects 即VBO，管理顶点数据在GPU的内存。

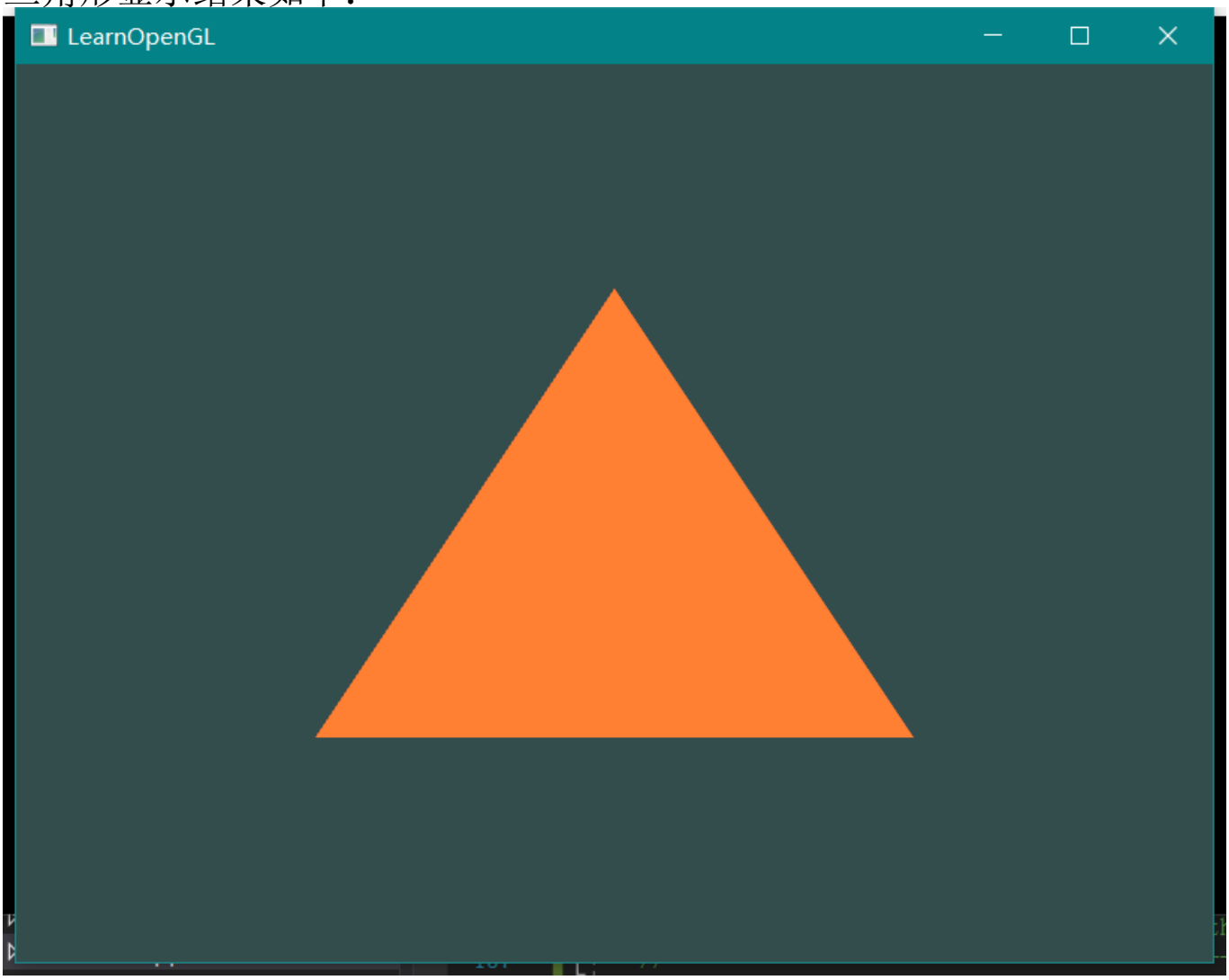
```
    unsigned int VBO;
        glGenBuffers(1, &VBO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 复制顶点数据到缓冲内存中（数据不会被改变）
```

另外，还要使用顶点数组对象 Vertex Array Objects，即VAO，让OpenGL知道该如何处理我们的顶点输入。VAO生成绑定的步骤和VBO类似。之后需要解析顶点数据，然后开始绘制三角形。

```
    // 创建和绑定
    unsigned int VAO;
        glGenVertexArrays(1, &VAO);
        glBindVertexArray(VAO);

        // 解析顶点数据
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
        glEnableVertexAttribArray(0);
```

最后循环渲染即可绘制出一个简单的三角形。在循环渲染的过程中，需要激活着色器程序对象、绑定VAO以及绘制三角形图元。

三角形显示结果如下：



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。

修改顶点着色器的源码为

```
const char *vertexShaderSource = "#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "layout (location = 1) in vec3 aColor;\n"
    "out vec3 ourColor;\n"
    "void main() {\n"
    "   gl_Position = vec4(aPos, 1.0);\n"
    "   ourColor = aColor;\n"
    "}\0";
```

这样，顶点着色器会接受坐标(location = 0)和颜色(location = 1)两个输入。另外，还需要把颜色数据加进顶点数据中。将颜色数据添加为3个float值至vertices数组。将把三角形的三个角分别指定为红色、绿色和蓝色：

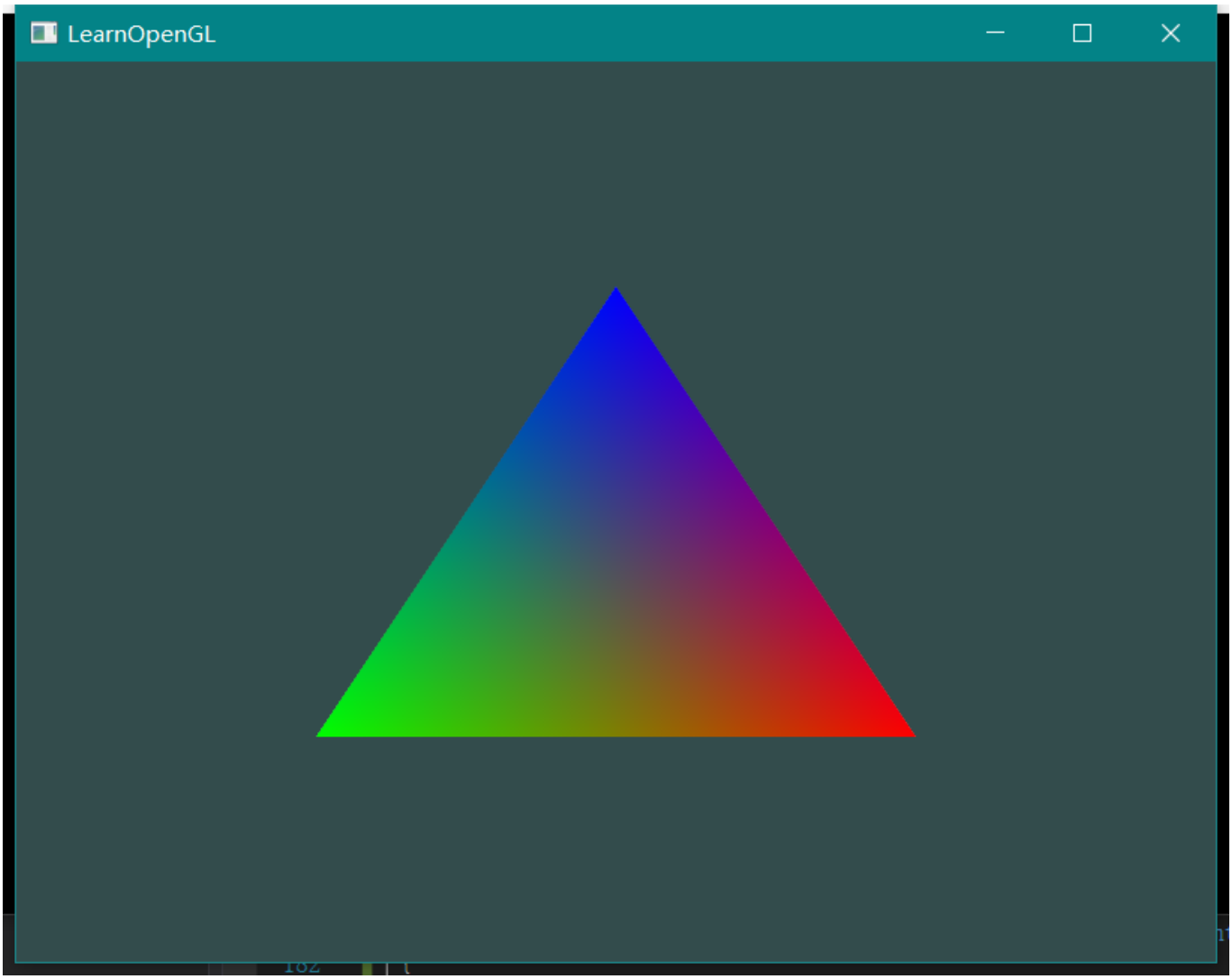
```
float vertices[] = {
    // 位置              // 颜色
     0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,   // 右下
    -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,   // 左下
```

```
        0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f    // 顶部
};
```

值得注意的是，解析顶点数据信息的时候要将坐标和颜色信息分开处理，如下所示。处理位置属性，需要从数组开头，每处理3个，跳过6个；处理颜色属性，需要从前3个元素之后开始读取，处理3个元素，并跳到6个元素之后。

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

这样，三角形的三个顶点就被赋予了不同的颜色。



而显示结果中，三角形呈现出了一种混色的效果，这是由于在片段着色器中进行片段插值**(Fragment Interpolation)**的结果。渲染三角形时，光栅化会造成比指定顶点更多的片段。每个片段在三角形上的相对位置都不同，基于这些位置，光栅会插值**(Interpolate)**所有片段着色器的输入变量。因此，除去**3**个顶点外，三角形中其他像素都是由片段着色器进行插值颜色的，也就是说呈现出的是**RGB**三色线性组合的颜色。

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。

首先对ImGui进行初始化。

```
        // 初始化ImGui环境
        IMGUI_CHECKVERSION();
        ImGui::CreateContext();
        ImGuiIO &io = ImGui::GetIO();
        (void)io;
        ImGui::StyleColorsDark();
        ImGui_ImplGlfw_InitForOpenGL(window, true);      // 绑定渲染器
        ImGui_ImplOpenGL3_Init(glsl_version);
```

在循环渲染的过程中，基于用户的输入（颜色选择），不断重新渲染三角形。ImGui窗口设置如下：

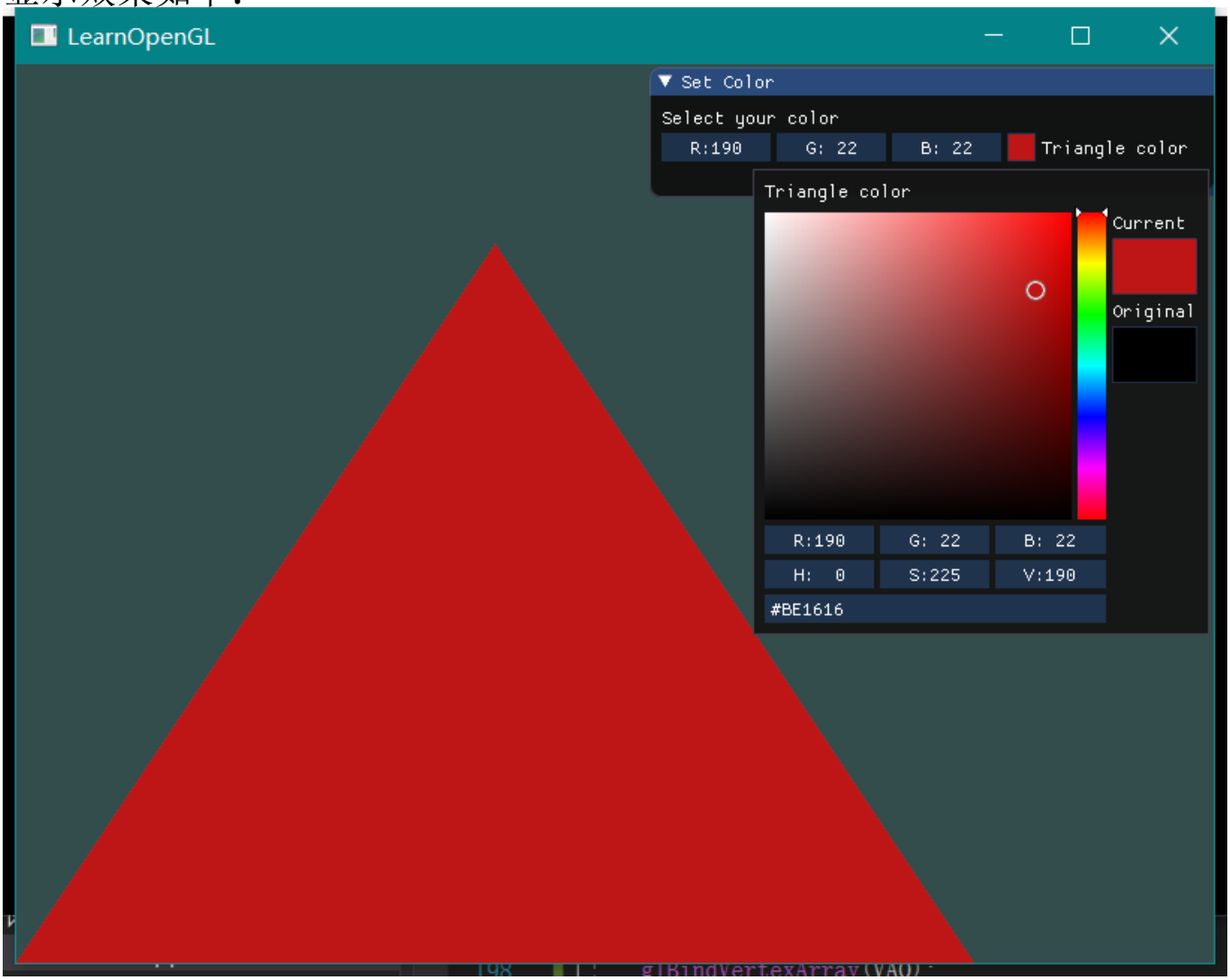
```
                // ImGui初始化
                ImGui_ImplOpenGL3_NewFrame();
                ImGui_ImplGlfw_NewFrame();
                ImGui::NewFrame();
                // 设置ImGui窗口样式
                ImGui::Begin("Set Color");
                ImGui::Text("Select your color");
                ImGui::ColorEdit3("Triangle color", (float*)&newColor);
                ImGui::End();

                // ImGui渲染
                ImGui::Render();
                int s_width, s_height;
                glfwMakeContextCurrent(window);
                glfwGetFramebufferSize(window, &s_width, &s_height);     // 根据窗
口的缓冲大小获取尺寸
                glViewport(0, 0, s_width, s_height);
                ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
```

关键步骤是渲染颜色如何确定。一开始，三角形默认呈现红绿蓝三色混色的效果，维护一个变量`colorSelected`用于判断用户有无输入，默认为`false`；一旦用户选择了新的颜色，`colorSelected`设为`true`，并获取用户选择的颜色，将三角形渲染成改颜色。

```
        // 创建顶点缓冲对象 Vertex Buffer Objects, 存储顶点, 缓冲类型是
GL_ARRAY_BUFFER
        unsigned int VBO;
        glGenBuffers(1, &VBO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        if (colorSelected)
                glBufferData(GL_ARRAY_BUFFER, sizeof(vertices2), vertices2,
GL_STATIC_DRAW);
        else {
```

```
                glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);        // 复制顶点数据到缓冲内存中（数据不会被改变）
                if (newColor.x > 0.0f)
                        colorSelected = true;
        }
```

显示效果如下：

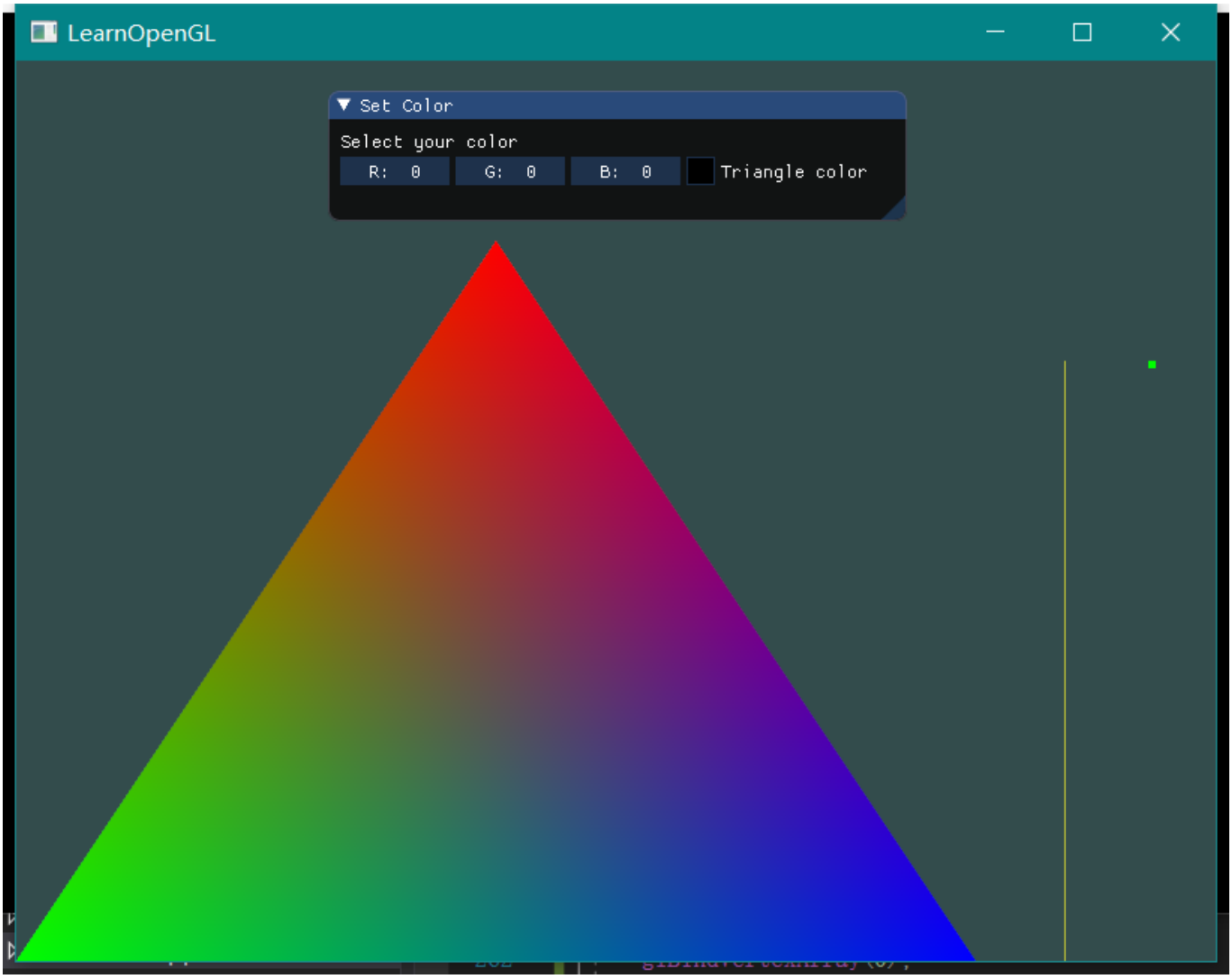


Bonus：

1. 绘制其他的图元，除了三角形，还有点、线等。

与绘制三角形一样，只不过调用`glDrawArrays()`需要注意修改第一个参数和第三个参数。以绘制线为例，一条线由两个顶点组成，因此需要这样调用：`glDrawArrays(GL_LINE_STRIP, 0, 2);`.

显示效果如下：


LearnOpenGL
Set Color
Select your color
R: 0
G: 0
B: 0
Triangle color