**计算机图形学 Homework2**

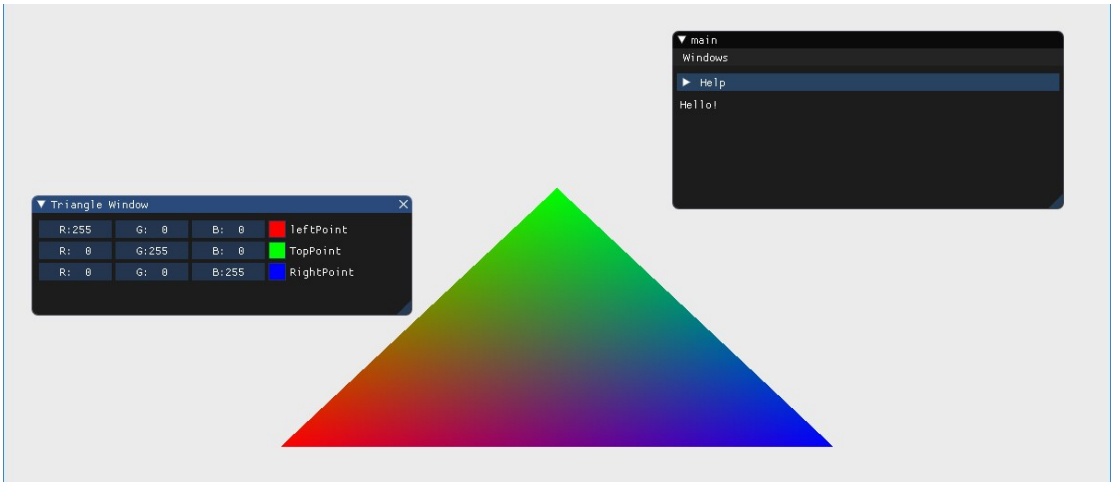
15331416 数字媒体技术方向 赵寒旭

# 目录

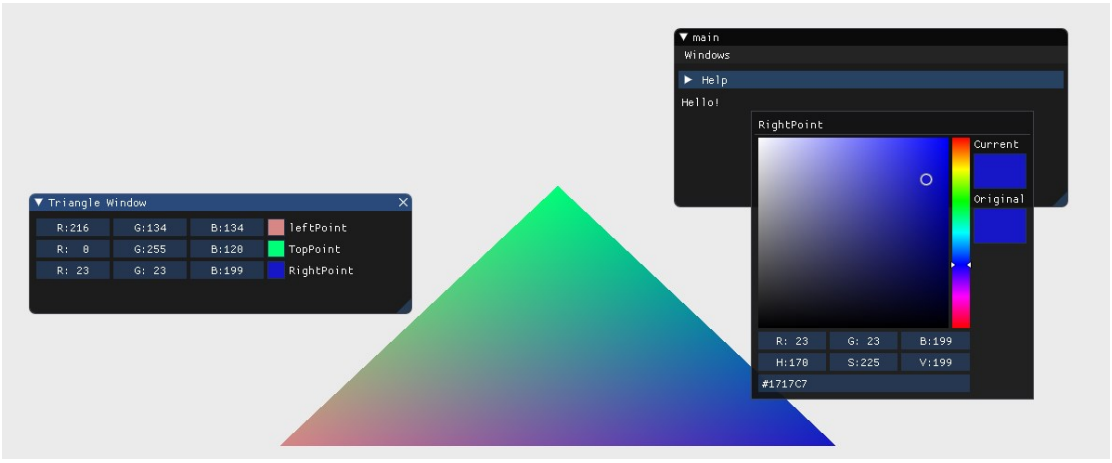
# 1. 运行结果展示

## 1.1 Basic

初始界面：

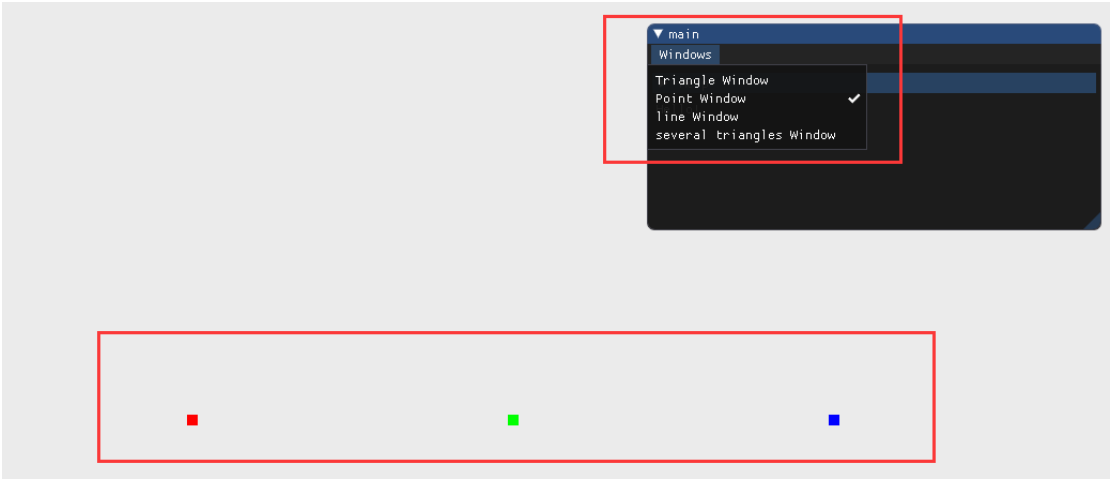



添加 GUI，从菜单栏改变三角形顶点颜色：

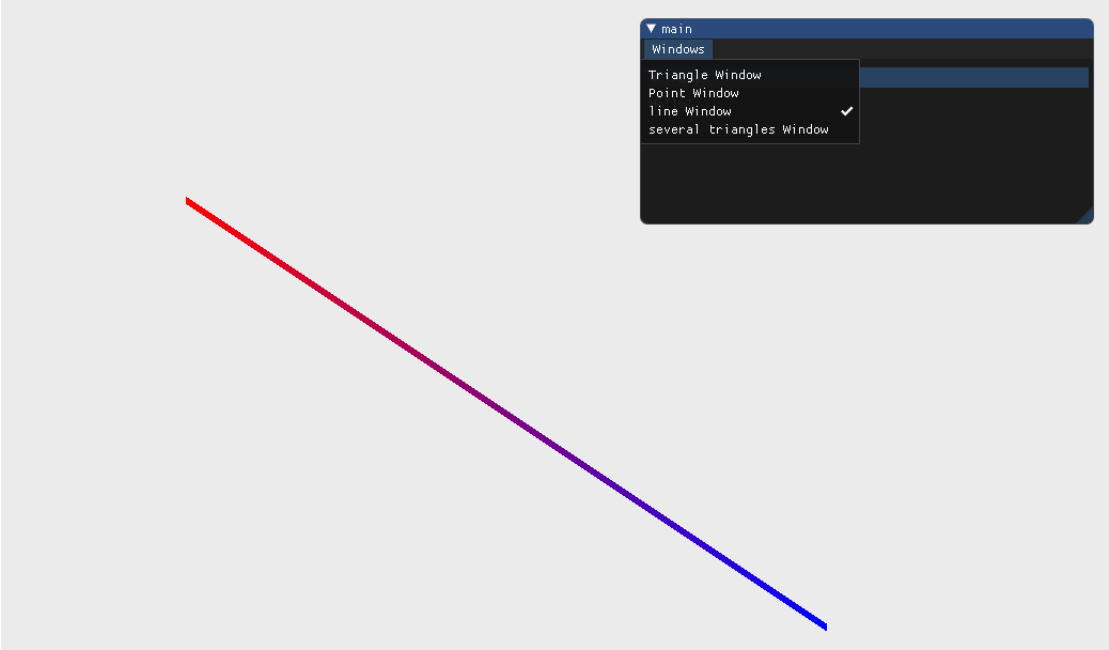



## 1.2 Bonus

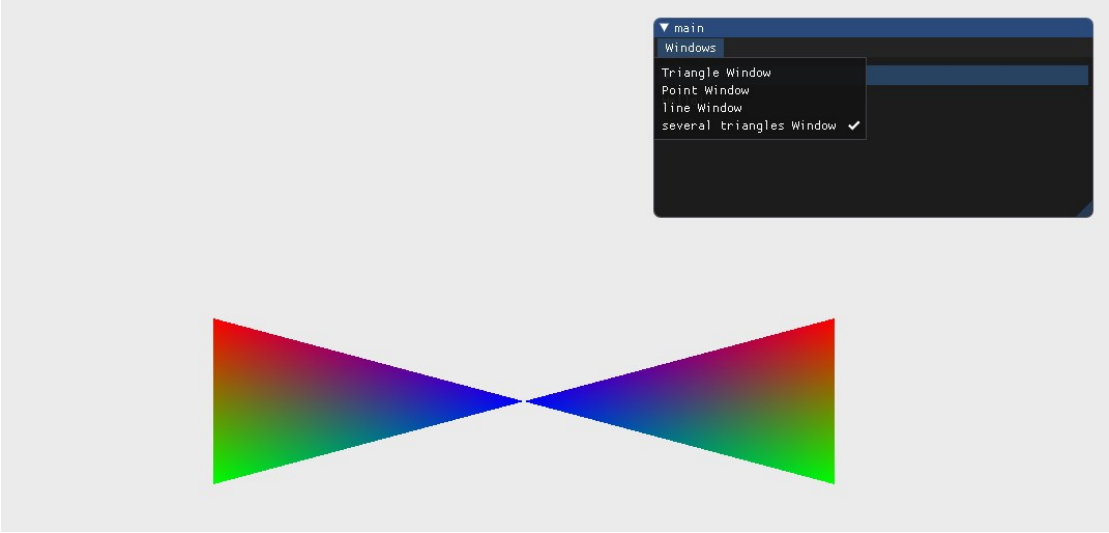
点的绘制：



线的绘制：




使用 EBO(Element Buffer Object)绘制多个三角形：




另有展示视频录制在 doc 目录下 HelloTriangle.mp4。

## 2. 回答作业中提出的问题

问：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。

我们只定义了顶点位置和对应的颜色，最终结果显示为一个调色板，是因为在片段着色器中进行了片段插值。在渲染一个三角形时，光栅化阶段通常会造成比原指定顶点更多的片段。光栅会根据每个片段在三角形形状上所处相对位置决定这些片段的位置。

基于这些位置，它会插值所有片段着色器的输入变量。以线段 AB 为例，如果 A 端点为红色，B 端点为蓝色，若一个片段着色器在线段上的某一位置运行，它在该位置的颜色输入属性就会是一个红色和蓝色的线性组合。

在定义了三角形的三个顶点和三个颜色之后，片段着色器会插值得到三角形内所有的像素颜色，即三角形内部像素点的颜色会是三个顶点颜色的线性组合。

## 3. 实现思路

（简要说明实现思路，以及主要 function/algorithm 的解释）

### 3.1 使用 OpenGL 绘制三角形

#### 1）初始化及窗口创建

（1）实例化 GLFW 窗口

```
// 初始化GLFW
glfwInit();
// 配置GLFW
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

（2）创建一个窗口对象

窗口对象存放了所有和窗口相关的数据，而且会被 GLFW 的其他函数频繁地用到。创建完窗口我们就可以通知 GLFW 将我们窗口的上下文设置为当前线程的主上下文了。

```
// 创建窗口对象，存放所有和窗口相关的数据
GLFWwindow* window = glfwCreateWindow(1200, 800, "HellloTriangle", NULL, NULL);
```

（3）初始化 GLAD

GLAD 是用来管理 OpenGL 的函数指针的，所以在调用任何 OpenGL 的函数之前我们需要初始化 GLAD。

```
// 初始化GLAD
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    cout << "Failed to initialize GLAD" << endl;
    return -1;
}
```

#### 2）着色器

（1）创建并编译着色器程序

- 编写顶点着色器

```
// 顶点着色器源码
#version 440 core
layout (location = 0) in vec3 Position;
layout (location = 1) in vec3 Color;
out vec3 vetexColor;
void main()
{
    gl_Position = vec4(Position, 1.0);
   vetexColor = Color;
};
```

输入：位置 Position、颜色 Color
输出：顶点颜色 vetexColor

■ 编写片段着色器

```
// 片段着色器源码
#version 440 core
out vec4 FragColor;
in vec3 vetexColor;
void main()
{
   FragColor = vec4(vetexColor, 1.0f);
};
```

输入：顶点颜色 vetexColor
输出：最终的输出颜色 FragColor

■ 编译并链接着色器：

这部分把编译的着色器链接为一个着色器程序对象(Shader Program Object)。

着色器程序对象是多个着色器合并之后并最终链接完成的版本。在渲染对象的时候激活这个着色器程序。已激活着色器程序的着色器将在我们发送渲染调用的时候被使用。

创建程序后，把之前编译的着色器附加到程序对象上，每个着色器的输出链接到下个着色器的输入，最后用 glLinkProgram 链接。

着色器创建及编译：

```
// 1. 顶点着色器
int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
```

```
// 2. 片段着色器
int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
```

着色器链接：

```
// 链接着色器
int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
```

```
// 链接完成后删除着色器对象
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

（3）激活着色器程序对象

```
// 激活着色器程序对象
glUseProgram(shaderProgram);
```

**3）顶点输入**

（1）定义顶点数据

```
// 三角形顶点设置
float vertices[] = {
    // positions         // colors
    -0.5f, -0.2f, 0.0f, 1.0f, 0.0f, 0.0f,  // bottom left
    0.0f,  0.5f, 0.0f, 0.0f, 1.0f, 0.0f,    // top
    0.5f, -0.2f, 0.0f, 0.0f, 0.0f, 1.0f  // bottom right
};
```

（2）顶点缓冲对象和顶点数组对象

```
// 顶点缓冲对象，顶点数组对象
unsigned int VBO;
unsigned int VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
// 绑定
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

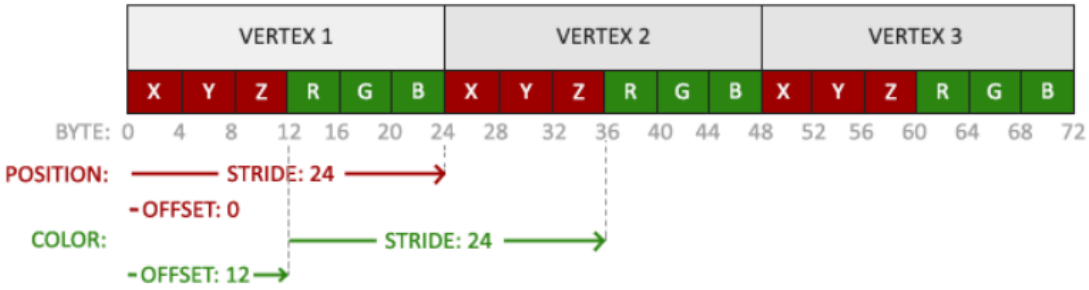
顶点缓冲对象(VBO)负责在 CPU 内存中储存大量顶点，可以一次性的发送一大批数据到显卡上，而不必每个顶点发送一次。

顶点数组对象(VAO)可以像顶点缓冲对象那样被绑定，任何随后的顶点属性调用都会储存在这个 VAO 中。配置顶点属性指针时，只需要将那些调用执行一次，之后再绘制物体的时候只需要绑定相应的 VAO。不同顶点数据和属性配置之间切换只需要绑定不同的 VAO。

（3）链接顶点属性

```
// 链接顶点属性
// 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);
```

VBO 内存中数据表示：




使用 glVertexAttribPointer 函数告诉 OpenGL 该如何解析顶点数据（应用到逐个顶点属性上）：

以位置数据为例，函数调用时 6 个参数含义及表示：

| 序号 | 定义 | 传入参数 | 具体表示 |
|---|---|---|---|
| 1 | 顶点属性 | 0 | 着色器设定位置值(Location=0) |
| 2 | 顶点属性的大小 | 3 | vec3，由 3 个值组成 |
| 3 | 指定数据的类型 | GL_FLOAT | GLSL 中 vec*都是由浮点数值组成的 |
| 4 | 是否希望数据被标准化 | GL_FALSE | 不需要在此处把数据映射到 0 到 1 之间 |
| 5 | 步长(Stride) | 6 * sizeof(float) | 连续的顶点属性组之间的间隔 |
| 6 | 偏移量(Offset) | (void*)0 | 位置数据在数组的开头 |

**4）渲染三角形**

（1）顶点数组复制到顶点缓冲中

```
// 把三角形顶点数组复制到一个顶点缓冲中，供OpenGL使用
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

glBindBuffer 函数把新创建的缓冲绑定到 GL_ARRAY_BUFFER 目标上之后，我们使用的任何（在 GL_ARRAY_BUFFER 目标上的）缓冲调用都会用来配置当前绑定的缓冲(VBO)。在渲染循环中绘制三角形时，可以调用 glBufferData 函数，把之前定义的三角形顶点数据复制到缓冲的内存中。

（2）绘制物体

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```

glDrawArrays 函数使用当前激活的着色器，之前定义的顶点属性配置，和 VBO 的顶点数据（通过 VAO 间接绑定）来绘制图元。

绘制三角形选择参数 GL_TRIANGLES 进行绘制。

### 3.2 使用 OpenGL 绘制其他图元

**1）点的绘制**

```
// rendering point
if (show_point_window)
{
    glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);
    glPointSize(10.0f);
    glDrawArrays(GL_POINTS, 0, 3);

}
```

与三角形绘制的区别在于传入缓冲内存的顶点数据不同，且绘制时使用 GL_POINTS 作为参数用于画点。

**2）线的绘制**

```
// rendering lines
if (show_line_window)
{
    glBufferData(GL_ARRAY_BUFFER, sizeof(lines), lines, GL_STATIC_DRAW);
    glLineWidth(10.0f);
    glDrawArrays(GL_LINES, 0, 2);
}
```

将线段顶点数据传入缓冲内存，使用 GL_LINES 作为参数用于绘制直线。

### 3.3 使用 EBO(Element Buffer Object)绘制多个三角形

**1）定义顶点数据**

顶点数组：

```
float severalTriangles[] = {
    // first
    0.5f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f,
    0.5f, -0.2f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
    // second
    -0.5f, 0.2f, 0.0f, 1.0f, 0.0f, 0.0f,
    -0.5f, -0.2f, 0.0f, 0.0f, 1.0f, 0.0f,
    0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f
};
```

使用索引定义两个三角形的六个顶点：

```
unsigned int indices[] = {
    0, 1, 2,
    3, 4, 5
};
```

**2）创建索引缓冲对象**

```
unsigned int EBO;
glGenBuffers(1, &EBO);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

索引缓冲对象(EBO)专门储存索引，OpenGL 调用这些顶点的索引来决定该绘制哪个顶点。

**3）渲染多个三角形**

```
glBufferData(GL_ARRAY_BUFFER, sizeof(severalTriangles), severalTriangles, GL_STATIC_DRAW);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glBindVertexArray(EBO);
```

传递 GL_ELEMENT_ARRAY_BUFFER 作为缓冲目标，用 glDrawElements 替换 glDrawArrays 函数，指明我们从索引缓冲渲染，glDrawElements 会使用当前绑定的索引缓冲对象中的索引进行绘制

### 3.4 GUI 的添加

在 OpenGL 环境下完成绘制之后，使用 ImGui 的接口和方法，可以为项目添加一个 GUI。

#### 1）GUI 初始化

```
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;
ImGui_ImplGlfwGL3_Init(window, true);
```

#### 2）三角形顶点颜色改变的实现

（1）用 float 数组存储三角形三个点的颜色值

```
// 三角形顶点颜色存储及更新
float left_color[] = { 1.0, 0.0, 0.0 };
float top_color[] = { 0.0, 1.0, 0.0 };
float right_color[] = { 0.0, 0.0, 1.0 };
```

（2）三角形顶点颜色值的改变

```
ImGui::Begin("Triangle Window", &show_triangle_window);
ImGui::ColorEdit3("leftPoint", (float*)&left_color);
vertices[3] = left_color[0];
vertices[4] = left_color[1];
vertices[5] = left_color[2];
```

在渲染循环中新建一个 ImGui 窗口实现三角形的变色功能。

以三角形左下角点为例，可以对应赋值替换三角形顶点数组中的颜色数据实现顶点的颜色改变。