计算机图形学作业报告

廖蕾 16340135

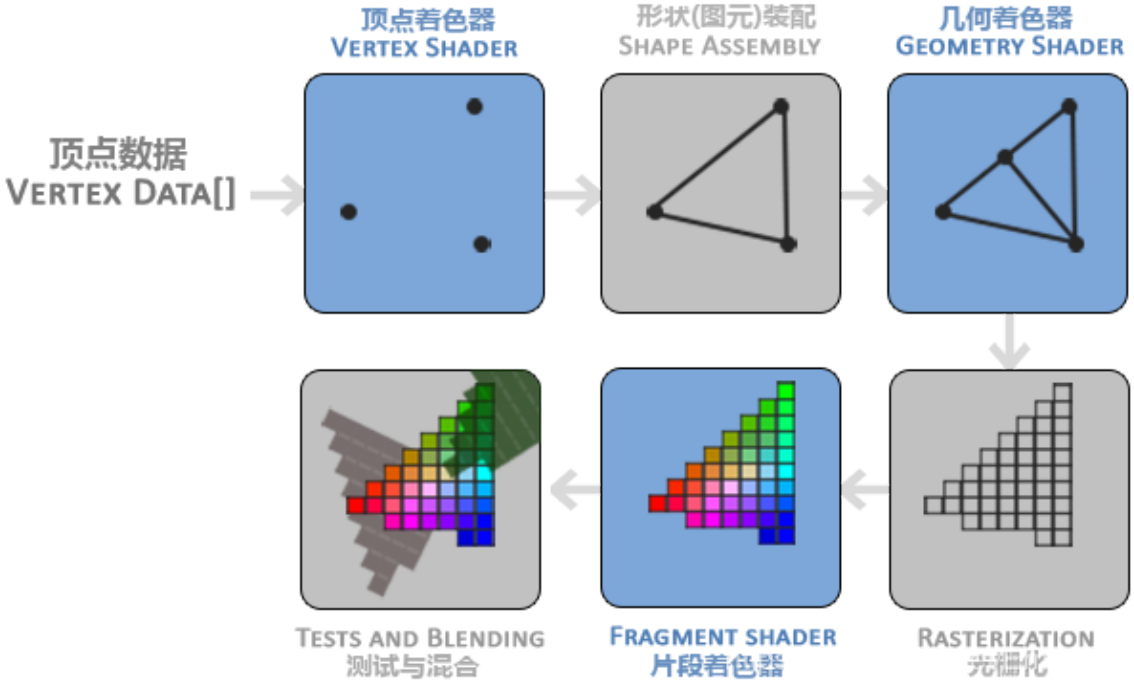
- 使用OpenGL(3.3及以上)+GLFW或freeglut画一个简单的三角形。 这部分的代码是参考learnopengl你好，三角形的。
  需要完成的步骤如下：
  - 初始化窗口：
    初始化GLFW，这是通过调用`glfwWindowHint`函数实现的。

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

    接着创建窗口，通过调用`glfwCreateWindow`函数，这个函数需要窗口的宽和高两个参数，还可以加入自定义的窗口名字。

```
GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
```

  - 绘制三角形：
    按照learnopengl官网上给的绘制原理（如下图），在绘制三角形的过程中，我用了以下几步：



    - 给OpenGL输入一些顶点数据：

```
float vertices[] = {
        -0.5f, -0.5f, 0.0f, // left
         0.5f, -0.5f, 0.0f, // right
         0.0f,  0.5f, 0.0f  // top
};
```

然后将这样的顶点数据存在一个缓冲对象中：

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

- 使用顶点着色器：
  我们需要用着色器语言GLSL编写顶点着色器，这样才能在以后的程序中使用它。

```
const char *vertexShaderSource = "#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"void main()\n"
"{\n"
"   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
"}\0";
```

定义之后需要编译着色器，需要用glCreateShader创建着色器对象，然后将需要创建的着色器类型给这个函数。最后将着色器源码附加到着色器对象上，编译。

```
int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
// check for shader compile errors
int success;
char infoLog[512];
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
```

- 片段着色器：
  操作过程和顶点着色器的构造类似，需要用到的代码如下：

```
const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"
```

```
"}\n\0";
int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
glCompileShader(fragmentShader);
// check for shader compile errors
glGetShaderiv(fragmentShader, GL_COMPILE_STATUS, &success);
```

- 着色器程序：
  着色器程序是将多个着色器合并之后并最终链接完成的版本。

```
int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
// check for linking errors
glGetProgramiv(shaderProgram, GL_LINK_STATUS, &success);
glDeleteShader(vertexShader);
glDeleteShader(fragmentShader);
```

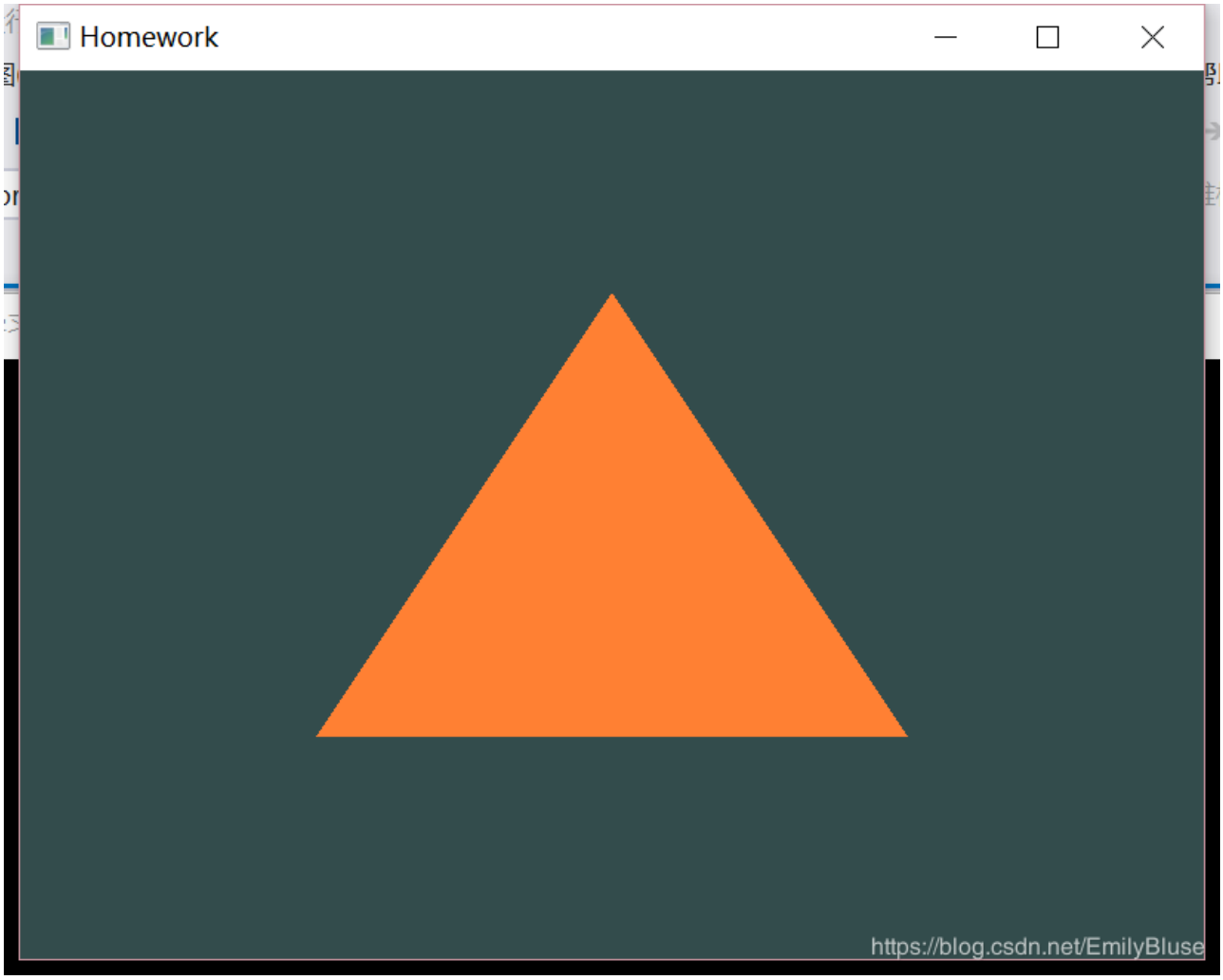
- 链接定点属性：
  这部分的目的是要我们手动的指定输入数据的哪一部分对应顶点着色器的哪一个顶点属性。
  这部分的代码如下：

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
glEnableVertexAttribArray(0);
```

- 渲染：
  在一个循环中，只要窗口没有被关闭，我们就要一直的渲染。这部分的代码如下：

```
while (!glfwWindowShouldClose(window)) {
        processInput(window);

        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glUseProgram(shaderProgram);
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 3);
        glfwSwapBuffers(window);
        glfwPollEvents();
}
```

这样我们的三角形就画好了，结果如下：



- 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。
  这部分是在上一个三角形的基础上，增加对着色器的使用得到的。 在这里我们需要修改两个着色器的定义：

```
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"
"{\n"
"   gl_Position = vec4(aPos, 1.0);\n"
"   ourColor = aColor;\n"
"}\0";

const char *fragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"
"in vec3 ourColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(ourColor, 1.0f);\n"
"}\n\0";
```

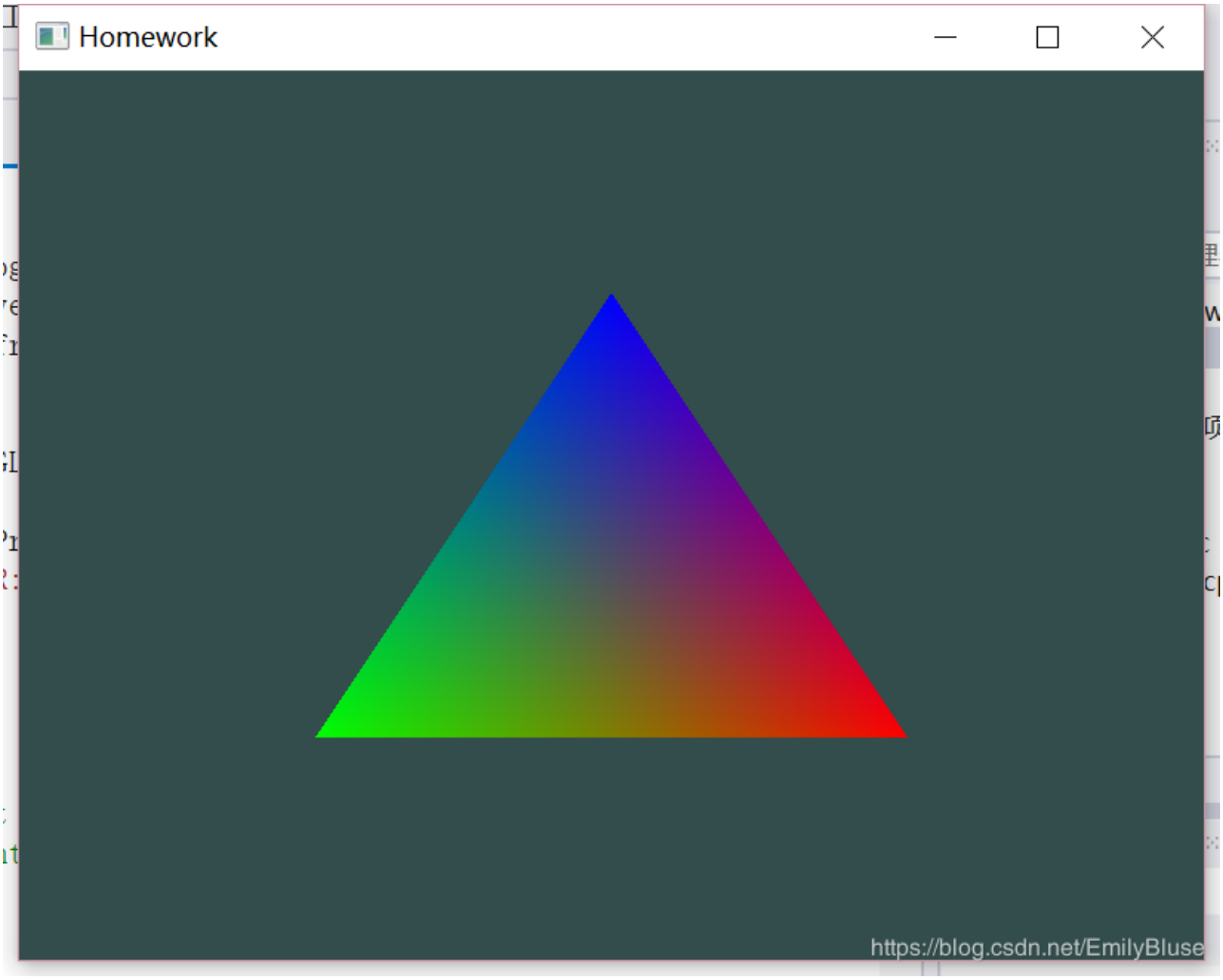
此外为了让三个顶点有颜色，对之前定义好的顶点坐标的数据也需要有改动：

```
float vertices[] = {
    //position           color
         0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,
        -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,
         0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f
};
```

在前三列是之前的顶点坐标数据，后三列是每隔顶点对应的颜色数据。
由于我们在VBO中的缓存多了颜色的数据，所以我们在绑定颜色这个参数的时候，要计算偏移步长，代码如下：

```
// color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3
* sizeof(float)));
glEnableVertexAttribArray(1);
```

在渲染的时候用的是和上面一样的代码。
这样得到的三角形结果如下：



问题：为什么会出现这样的结果？

> - 答：当我们只给出了三个顶点的颜色值的时候，片段着色器会进行片段插值的操作。当渲染一个三角形的时候，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上的相对位置决定这些片段的位置。
> 当有了这些位置的值之后，就会开始插值。这样在两个顶点之间的颜色，就是这两个顶点颜色的线性组合的结果。所以当三个顶点是红、绿、蓝的时候，就会出现这样的结果了。

- 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。

  要加入GUI，需要加入ImGui。在ImGui官网上下载源码，加入项目的头文件和源文件中，就可以使用了。
  我们在使用ImGui的时候，需要在`main.cpp`中加入ImGui的头文件：

  ```
  #include "imgui.h"
  #include "imgui_impl_glfw_gl3.h"
  ```

  GUI的加入需要分一下几步：

  - 对GUI初始化：
    使用如下代码对GUI进行初始化：

    ```
    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO(); (void)io;
    io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard;  // Enable
    Keyboard Controls
    ImGui_ImplGlfwGL3_Init(window, true);
    ImGui::StyleColorsDark();
    ```

  - 在渲染中加入GUI的一些控件：
    在这里和改变颜色需要的控件是两个CheckBox和三个ColorEdit3。CheckBox的使用需要绑定Bool值，ColorEdit的使用需要用到ImVec4这个数据类型。我这里的写法是这样的：

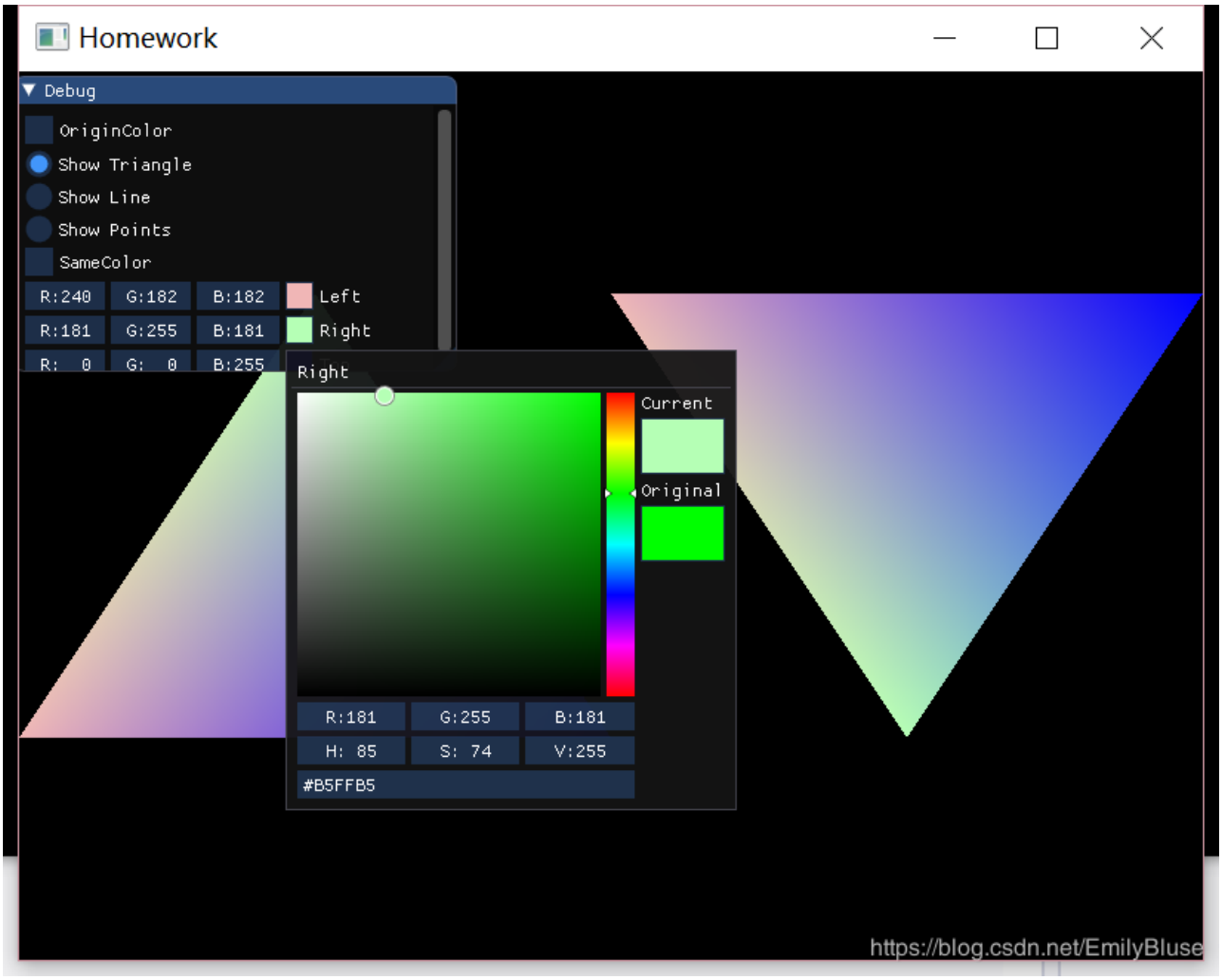
    ```
    // Triangle info
    ImVec4 tri1 = ImVec4(1.0f, 0.0f, 0.0f, 1.00f);
    ImVec4 tri2 = ImVec4(0.0f, 1.0f, 0.0f, 1.00f);
    ImVec4 tri3 = ImVec4(0.0f, 0.0f, 1.0f, 1.00f);

    bool sameColor = false;
    bool originColor = true;
    ...
    //在循环中...
    ImGui_ImplGlfwGL3_NewFrame();
    ImGui::SetWindowSize(ImVec2(300, 200));
    ```

```
ImGui::Checkbox("OriginColor", &originColor);
if (!originColor) {
        ImGui::Checkbox("SameColor", &sameColor);
        if (!sameColor) {
                ImGui::ColorEdit3("Left", (float*)&tri1);
                ImGui::ColorEdit3("Right", (float*)&tri2);
                ImGui::ColorEdit3("Top", (float*)&tri3);
        }
        else {
                ImGui::ColorEdit3("All Vertices", (float*)&tri1);
                tri2 = tri1;
                tri3 = tri1;
        }
}
```

- 在三角形的渲染中也需要改变一下渲染的方法，如下：

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(triangleIndices),
triangleIndices, GL_STATIC_DRAW);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
ImGui::Render();
ImGui_ImplGlfwGL3_RenderDrawData(ImGui::GetDrawData());
glfwSwapBuffers(window);
```

这样我们就在之前代码的基础上又加入了GUI，详细使用结果，请见同文件夹下的视频。这里只放图片如下：



Bouns

- 绘制其他的图元，除了三角形，还有点、线等。
  【注】：这里的代码和加GUI的部分放在了一起。
  为了加入线，我们同样也需要用两个顶点决定一个线。我们可以利用三角形中的顶点值，加入直线对应两点的下标说明，构造出直线。

  - 在原来代码中加入对顶点位置的说明：

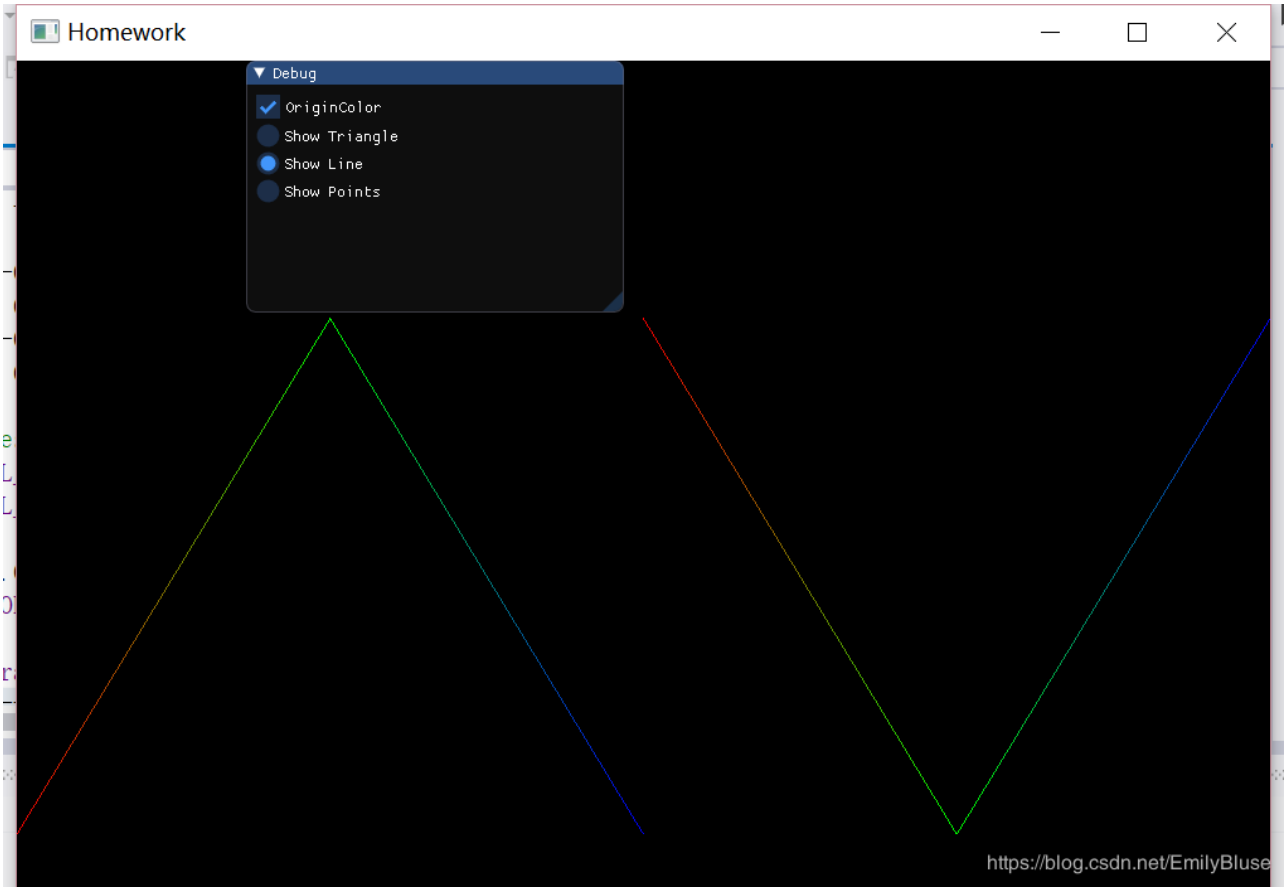
```
float vertices[] = {
                -1.0f, -0.5f, 0.0f, tri1.x, tri1.y, tri1.z,
                -0.5f,  0.5f, 0.0f, tri2.x, tri2.y, tri2.z,
                0.0f, -0.5f, 0.0f, tri3.x, tri3.y, tri3.z,
                0.0f,  0.5f, 0.0f, tri1.x, tri1.y, tri1.z,
                0.5f, -0.5f, 0.0f, tri2.x, tri2.y, tri2.z,
                1.0f,  0.5f, 0.0f, tri3.x, tri3.y, tri3.z,
};

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

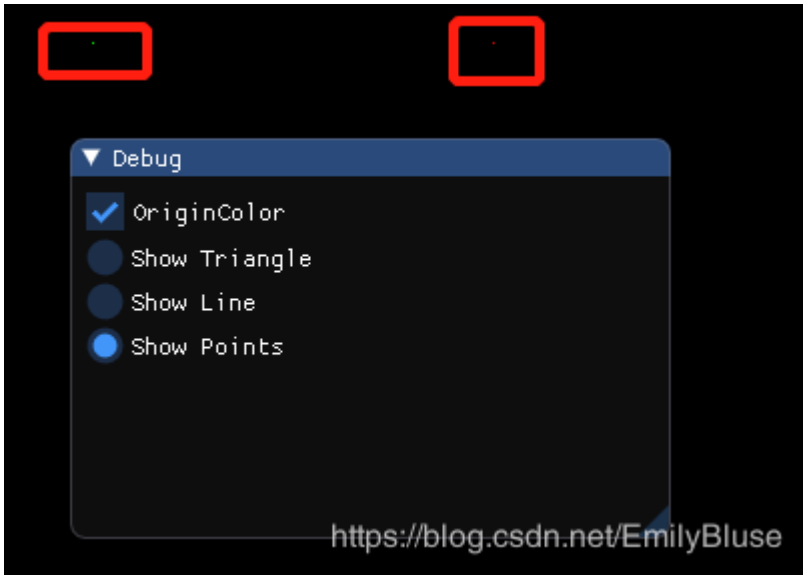
这样的点也可以用在我们渲染画出点的过程中。

- 在渲染时，加入GUI的控制。使用RadioButton，当选中显示直线的时候，我们渲染直线；当选中显示点的时候，我们渲染点。由于我们一共有8个直线的顶点，所以在渲染的时候，需要给`glDrawElements`8这个参数：

```
...
//[渲染中...]
ImGui::RadioButton("Show Triangle", &radioMark, 0);
ImGui::RadioButton("Show Line", &radioMark, 1);
ImGui::RadioButton("Show Points", &radioMark, 2);

if (radioMark == 0) {
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(triangleIndices),
triangleIndices, GL_STATIC_DRAW);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
} else if (radioMark == 1) {
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(lineIndices),
lineIndices, GL_STATIC_DRAW);
        glDrawElements(GL_LINES, 8, GL_UNSIGNED_INT, 0);
} else if (radioMark == 2) {
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
        glDrawArrays(GL_POINTS, 0, 6);
}
```

得到的结果如下：





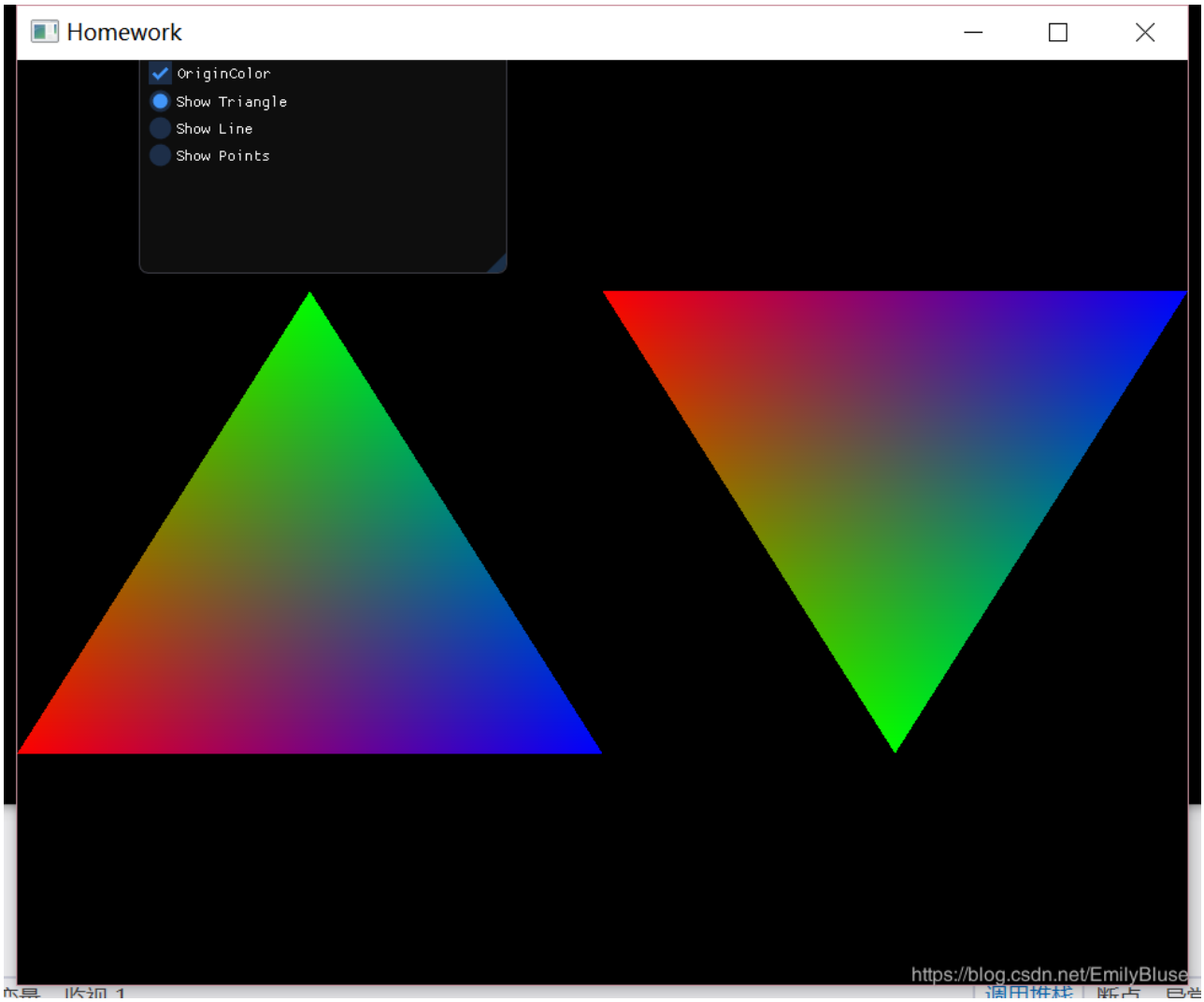
- 使用EBO(Element Buffer Object)绘制多个三角形。
  【注】：我们在上面代码的基础上加入第二个三角形。
  在我们定义的点中，其实已经有两个三角形的顶点值了。所以我们也只需要再定义这些点对应的下标就好了：

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

在渲染的时候，我们将这样的下标元素加入缓存中即可：

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(triangleIndices),
triangleIndices, GL_STATIC_DRAW);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

这样就可以在渲染的时候得到两个三角形：



【完整的结果展示请看视频】