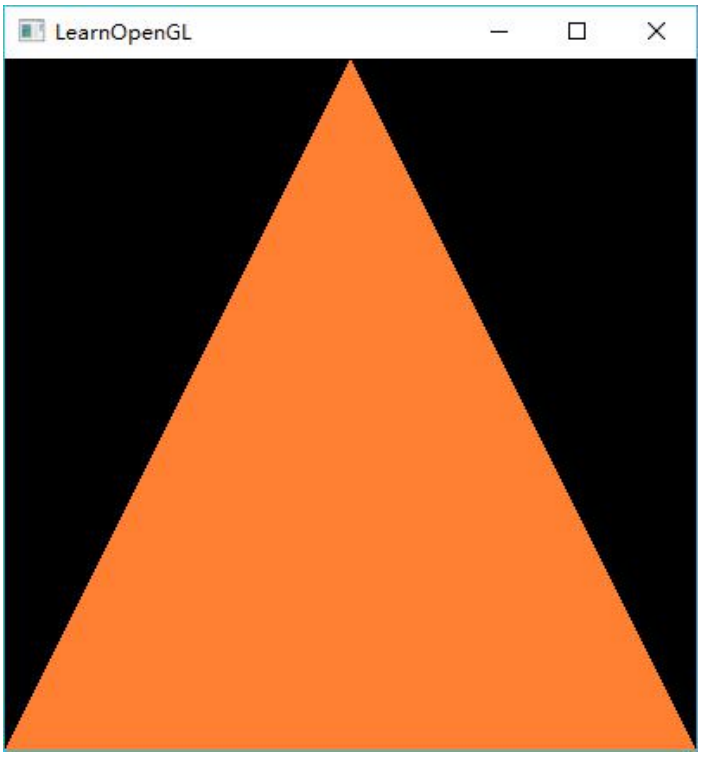
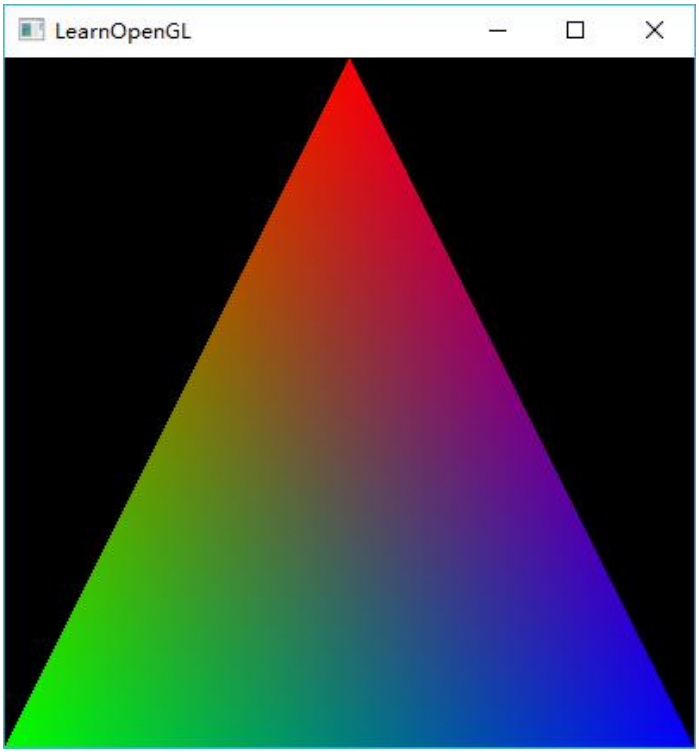
1. 使用 OpenGL(3.3 及以上)+GLFW 或 freeglut 画一个简单的三角形。



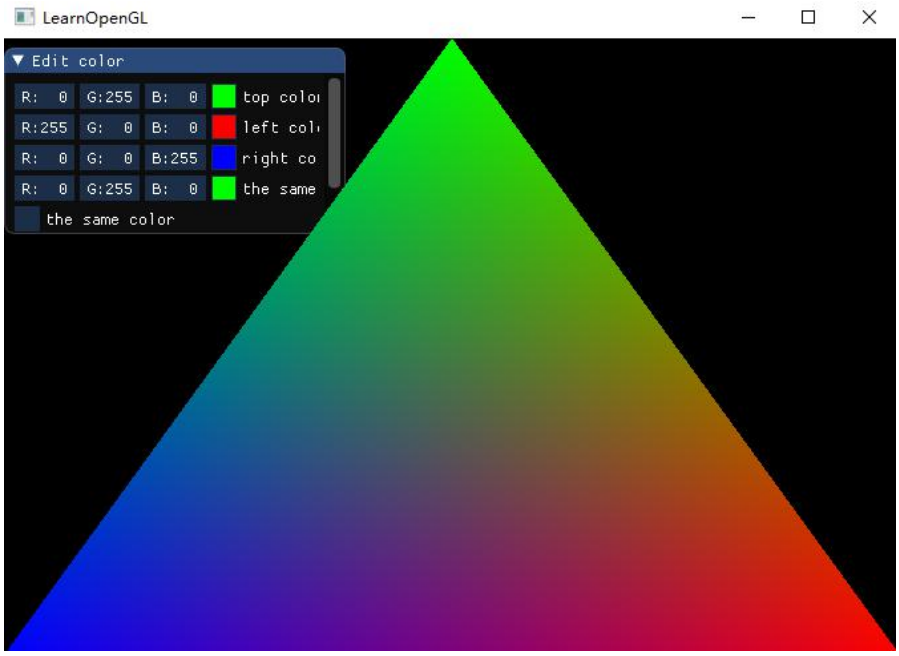
2. 对三角形的三个顶点分别改为红绿蓝，像下面这样。并解释为什么会出现这样的结果。



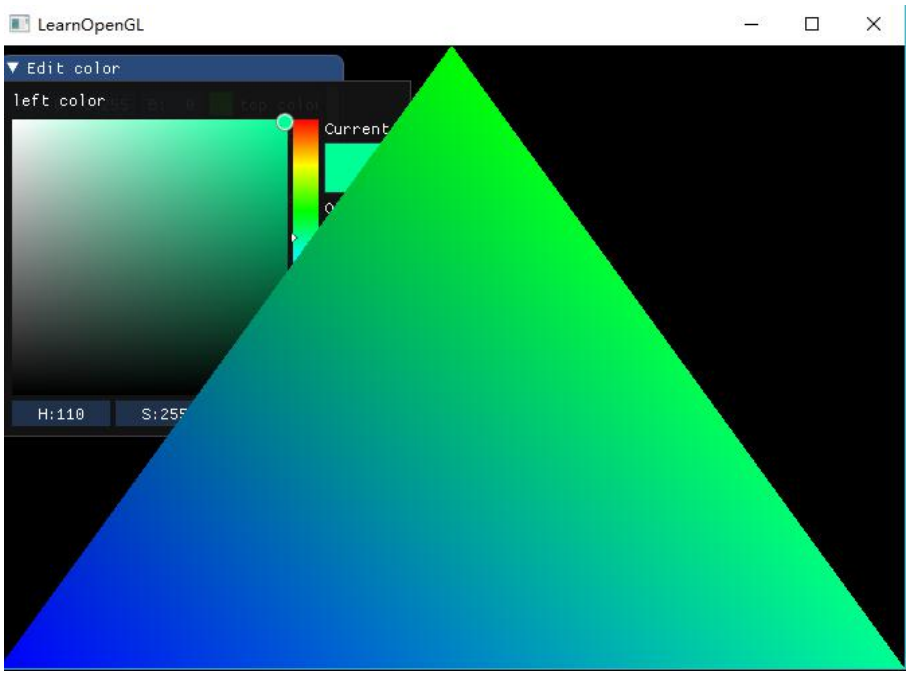
解释：这是在片段着色器中进行的所谓片段插值后导致的结果。当渲染一个三角形时，光栅化阶段通常会造成比原指定顶点更多的片段。光栅会根据每个片段在三角形形状上所处相对位置决定这些片段的位置。基于这些位置，它会插值所有片段着色器的输入变量。我们有 3 个顶点，和相应的 3 个颜色，从这个三角形的像素来看它可能包含 50000 左右的片段，片段着色器为这些像素进行插值颜色。

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。

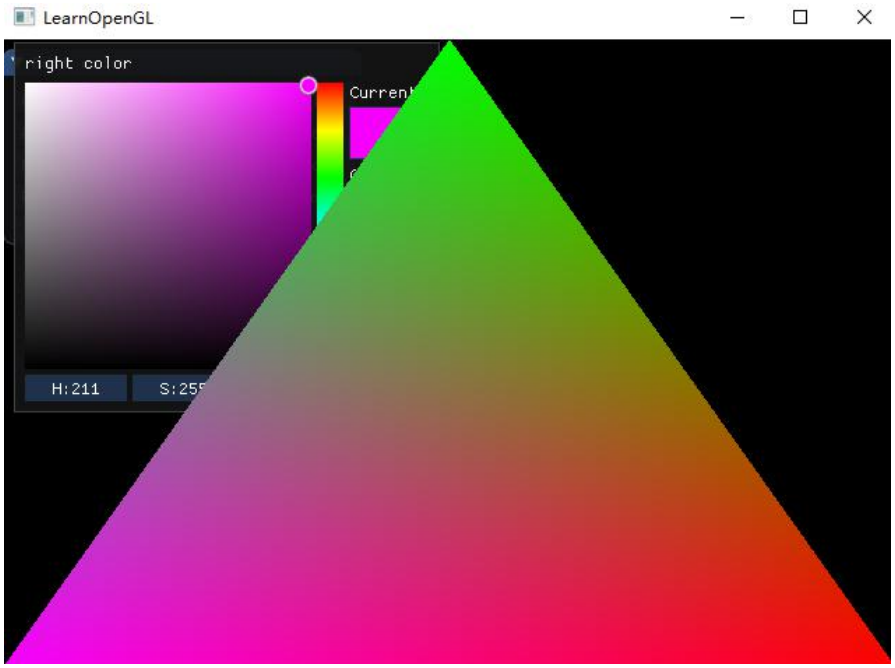
出现菜单栏



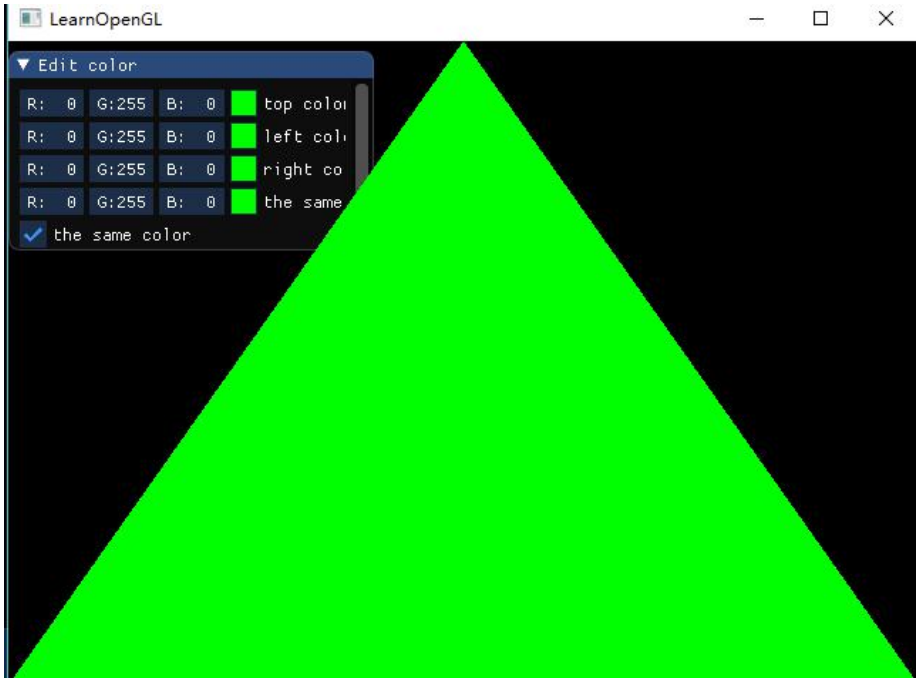
改变左顶点颜色



改变右顶点颜色



三个顶点颜色相同



实现思路：

1. vertexShaderSource：顶点着色器
2. vertexShaderSource：片段着色器
3. glfwInit（）：初始化 GLFW
4. glfwWindowHint（）：使用 glfwWindowHint 配置 GLFW
5. glfwCreateWindow（）：创建窗口对象
6. glfwMakeContextCurrent（）：通知 GLFW 将窗口的上下文设置为当前线程的主上下文
7. gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)：初始化 GLAD，GLAD 用来管理 OpenGL 的函数指针，在调用任何 OpenGL 的函数之前需要初始化 GLAD

8. glViewport（）：设置窗口维度

9. 创建并绑定 ImGui:

```
IMGUI_CHECKVERSION();
    CreateContext();
    ImGuiIO& io = GetIO(); (void)io;
    StyleColorsDark();
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init(glsl_version);
```

10. 创建 ImGui:

```
    ImGui_ImplOpenGL3_NewFrame();
        ImGui_ImplGlfw_NewFrame();
        NewFrame();
        Begin("Edit color");

        ColorEdit3("top color", (float*)&top_color);
        ColorEdit3("left color", (float*)&left_color);
        ColorEdit3("right color", (float*)&right_color);
        ColorEdit3("the same color", (float*)&same_color);

        Checkbox("the same color", &the_same_color);
        Checkbox("Draw triangle without rendering",
&draw_trangle_without_render);
```

Checkbox("Basic draw triangle", &draw_trangle);

End();

11. 渲染窗口颜色：

Render();

int view_width, view_height;

glfwMakeContextCurrent(window);

glfwGetFramebufferSize(window, &view_width, &view_height);

glViewport(0, 0, view_width, view_height);

glClear(GL_COLOR_BUFFER_BIT);//清空屏幕的颜色缓冲

ImGui_ImplOpenGL3_RenderDrawData(GetDrawData());

12. 渲染三角形

glEnableVertexAttribArray(1);

glBindVertexArray(VAO);

glDrawArrays(GL_TRIANGLES, 0, 3);

12.glfwSwapBuffers（）：*双缓冲。前缓冲保存着最终输出的图像，它会在屏幕上显示；而所有的的渲染指令都会在后缓冲上绘制。*

*13.*释放 VAO、VBO 资源

glDeleteVertexArrays(1, &VAO);

glDeleteBuffers(1, &VBO);

14. 释放 ImGui 资源

ImGui_ImplOpenGL3_Shutdown();

ImGui_ImplGlfw_Shutdown();

DestroyContext();

15. 释放所有申请的 glfw 资源

glfwTerminate();

16. glGenBuffers（）：使用 glGenBuffers 函数和一个缓冲 ID 生成一个 VBO 对象：

17. glBufferData（）：把之前定义的顶点数据复制到缓冲的内存中

18. glBindBuffer（）：把新创建的缓冲绑定到 GL_ARRAY_BUFFER 目标上

19. glVertexAttribPointer（）：解析定点数据

20. glUseProgram（）：激活程序对象

21.定点 3D 坐标：　　float vertices[] = {

-1.0f, -1.0f, 0.0f,right_color.x, right_color.y, right_color.z,

1.0f, -1.0f, 0.0f, left_color.x, left_color.y, left_color.z,

0.0f,　1.0f, 0.0f, top_color.x, top_color.y, top_color.z　　};

21. glCreateShader（）：创建着色器对象，用 ID 引用

22. glShaderSource（）：着色器源码附加到着色器对象上

23. glCompileShader（）：编译

24. 测试编译是否成功

glGetShaderiv(framentShader, GL_COMPILE_STATUS, &success);

if (!success)

{

```
        glGetShaderInfoLog(framentShader, 512, NULL, infoLog);
        cout << "ERROR::SHADER::FRAGMENT::COMPILATION_FAILED\n"
<< infoLog << endl;
    }
```

25. glCreateProgram（）：创建着色器程序对象

26. 把着色器附加到程序对象上，用 glLinkProgram 链接

```
 glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, framentShader);
glLinkProgram(shaderProgram);
```

27.删除着色器对象

```
glDeleteShader(vertexShader);
glDeleteShader(framentShader);
```