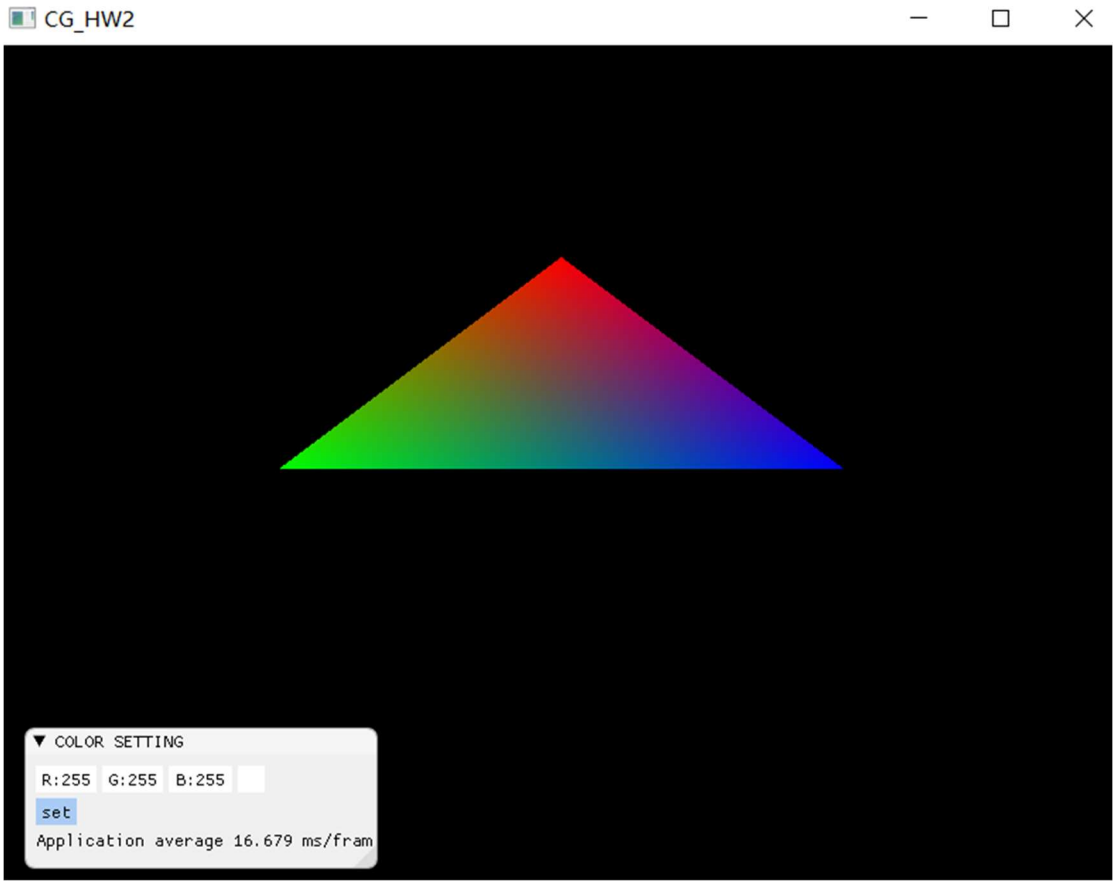
BASIC：

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

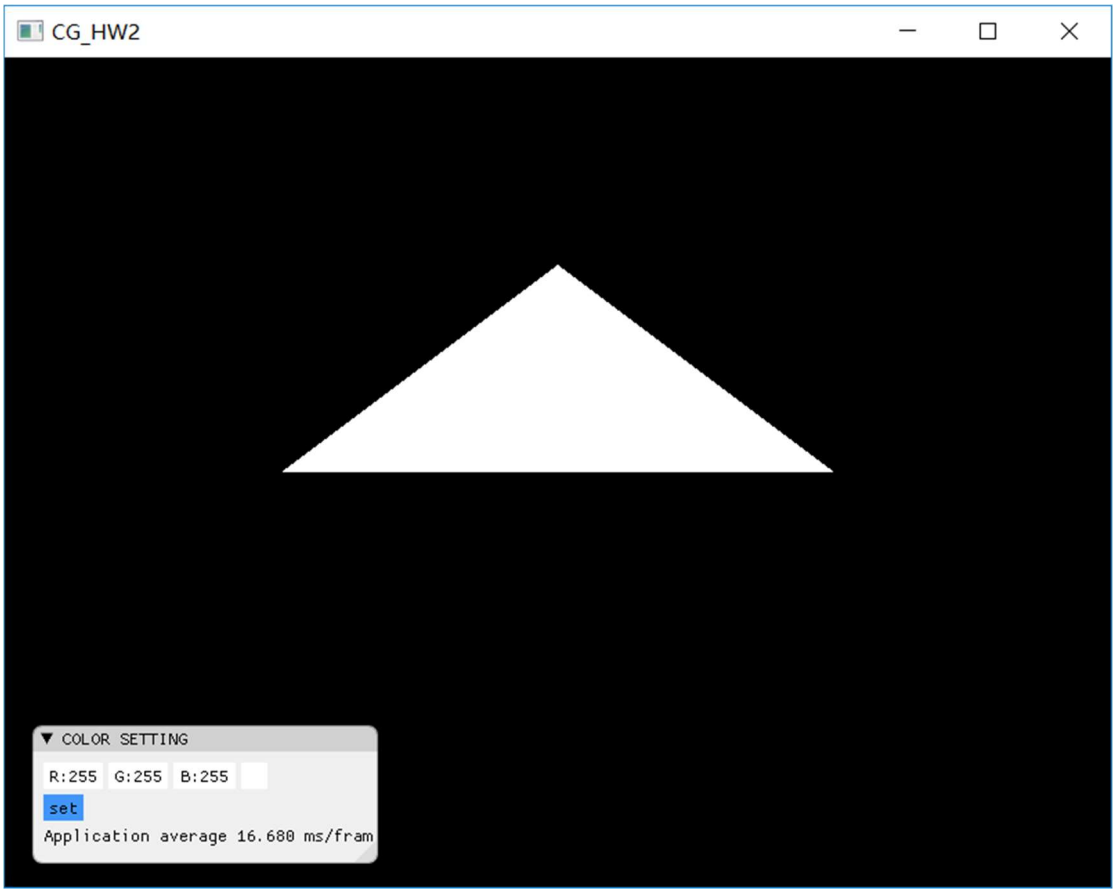


2. 对三角形的三个顶点分别改为红绿蓝，像下面这样。并解释为什么会出现这样的结果。

   这是因为在片段着色器中会进行的片段插值。当渲染一个三角形时，光栅化阶段会生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，插值所有片段着色器的输入变量。

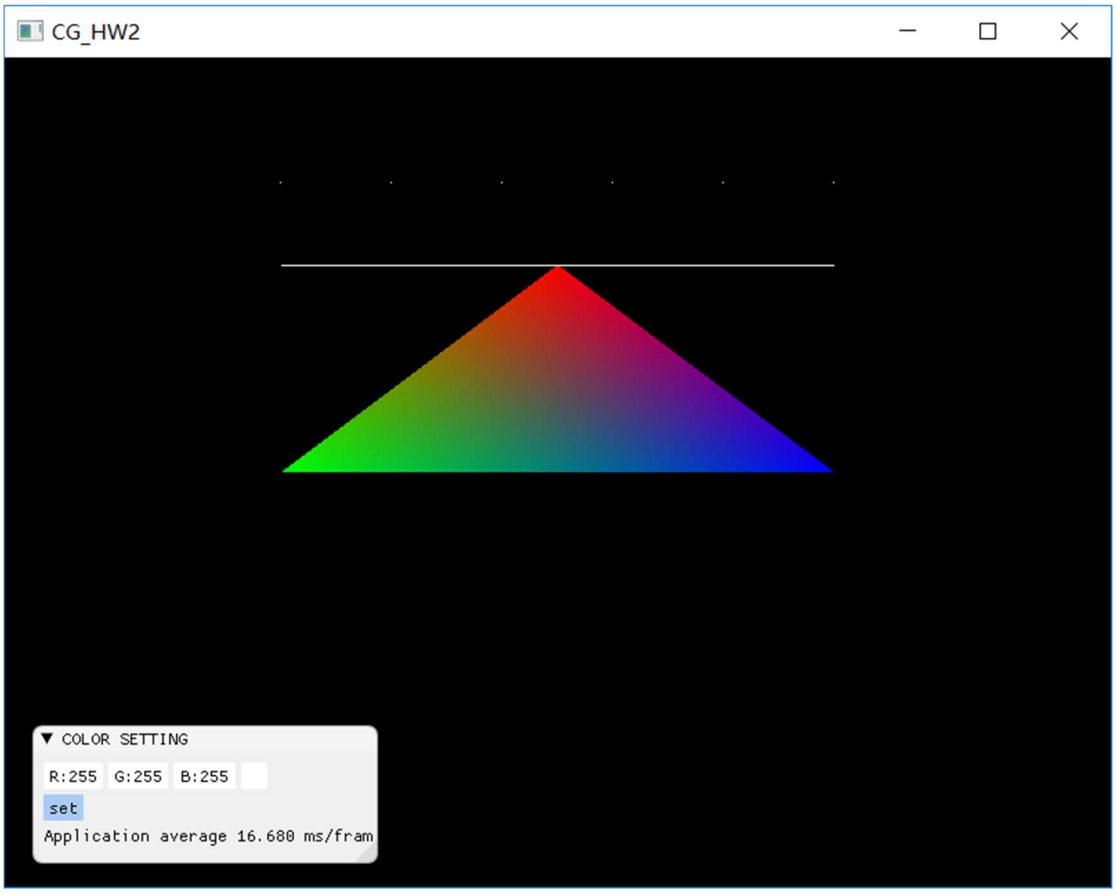
   即是光栅化的时候，根据我们提供的 3 个顶点来确定三角形边和内部的像素点。因为只有 3 个顶点有颜色输入变量，OpenGL 会将其他像素点进行按与 3 个顶点的距离等比例插值作为输入，所以结果与调色板类似。

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。（选择颜色值后按 SET 按钮即可生效）

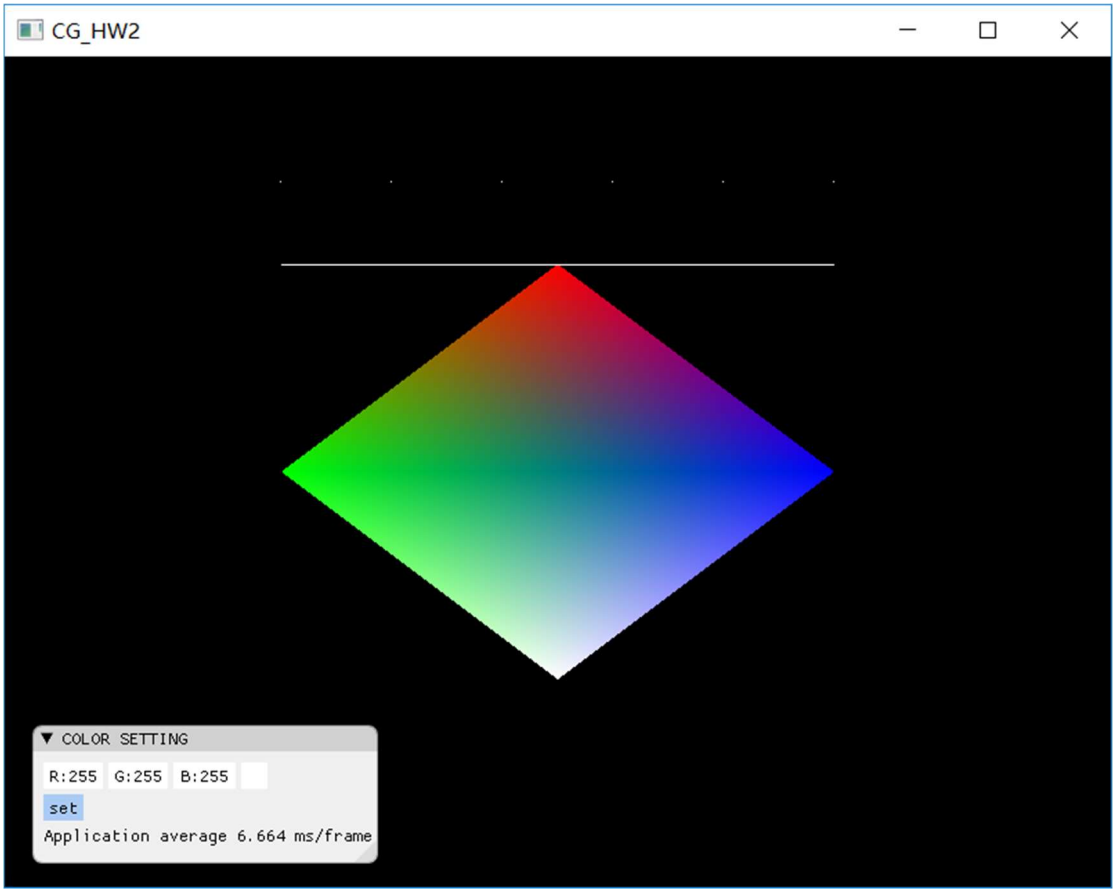


Bonus

1. 绘制其他的图元，除了三角形，还有点、线等。
   (三角形上面一条白色直线，直线上面有五个白点)



2. 使用 EBO(Element Buffer Object)绘制多个三角形。



代码思路解析

附代码文件 main.cpp

1. 两个着色器代码

   顶点着色器主要对传入的顶点数据预处理，传递位置和颜色值，片段着色器则对颜色值作最后的处理。

```
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, 1.0f);\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";
```

2. 顶点数据设置

   四个顶点画两个三角形，两点画直线，五个离散点。

```
// set up vertex data (and buffer(s)) and configure vertex attributes
// ------------------------------------------------------------------
float vertices[] = {
    // 位置              // 颜色
    //triangle
     0.5f,   0.0f,  0.0f,   0.0f,  0.0f,  1.0f,
    -0.5f,   0.0f,  0.0f,   0.0f,  1.0f,  0.0f,
     0.0f,   0.5f,  0.0f,   1.0f,  0.0f,  0.0f,
     0.0f,   -0.5f,  0.0f,   1.0f,  1.0f,  1.0f,

    -0.5f,  0.5f,0.0f,   1.0f,  1.0f,  1.0f,
     0.5f,  0.5f,0.0f,   1.0f,  1.0f,  1.0f,

     -0.5f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f,
     -0.3f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f,
     -0.1f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f,
     0.1f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f,
     0.3f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f,
     0.5f,  0.7f,0.0f,   1.0f,  1.0f,  1.0f
};

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

3. Imgui 控件

   当用户选择颜色控件并点击 SET 按钮时，更新顶点数组内的颜色值，并重新绑定输送数据到显卡。

```
//imGui
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();

{
    ImGui::Begin("COLOR SETTING");
    ImGui::ColorEdit3("", (float*)&color, 1);

    //Change the color
    if (ImGui::Button("set"))
    {
        for (int i = 0; i < 3; i++) {
            vertices[i * 6 + 3] = color.x;
            vertices[i * 6 + 4] = color.y;
            vertices[i * 6 + 5] = color.z;
        }
    }

    ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate, ImGui::GetIO().Framerate);
    ImGui::End();
}
```

4. 渲染核心部分

```
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

glUseProgram(shaderProgram);
glBindVertexArray(VAO);
//VBO
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
//EBO
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
//position
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
//color
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);

//glDrawArrays(GL_TRIANGLES, 0, 3);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
glDrawArrays(GL_LINES, 4, 2);
glDrawArrays(GL_POINTS, 6, 6);

//unbind
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);

//render imGui
ImGui::Render();
ImGui_Impl0penGL3_RenderDrawData(ImGui::GetDrawData());
```