Assignment 0: Introduction to OpenGL

> Task 1、什么是OpenGL？OpenGL与计算机图形学的关系是什么？
> Task 2、完成了 着色器 章节之后，请修改顶点着色器让三角形上下颠倒。
> Task 3、完成了 纹理 章节之后，尝试用不同的纹理环绕方式，设定一个从 0.0f 到 2.0f 范围内的（而不是原来的 0.0f 到 1.0f ）纹理坐标。试试看能不能在箱子的角落放置4个笑脸。记得一定要试试其它的环绕方式。简述原因并贴上结果。
> Task 4、完成了 坐标系统 章节之后，对GLM的projection函数中的FoV和aspect-ratio参数进行实验。看能否搞懂它们是如何影响透视平截头体的。
> Task 5、请按照顺序将跟着教程实现的运行结果贴出来，要求将运行出来的窗口的标题改成自己的学号。(Tip: glfwCreateWindow函数)

姓名：伍斌
学号：17343124

Task 1、什么是OpenGL？OpenGL与计算机图形学的关系是什么？

1. Open Graphics Library，开放式图形库，是用于渲染2D、3D矢量图形的跨语言、跨平台的应用程序编程接口（API），包含了一系列可以操作图形、图像的函数。不过OpenGL本身并不是一个API，它仅仅是一个由Khronos组织制定并维护的规范(Specification)，它严格规定了每个函数该如何执行，以及它们的输出值，内部具体每个函数是如何实现由OpenGL库的开发者自行决定。
2. 计算机图形学是研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法。OpenGL是包含了一系列可以操作图形、图像的函数的API。计算机图形学是算法，指导利用OpenGL来实现各种功能。

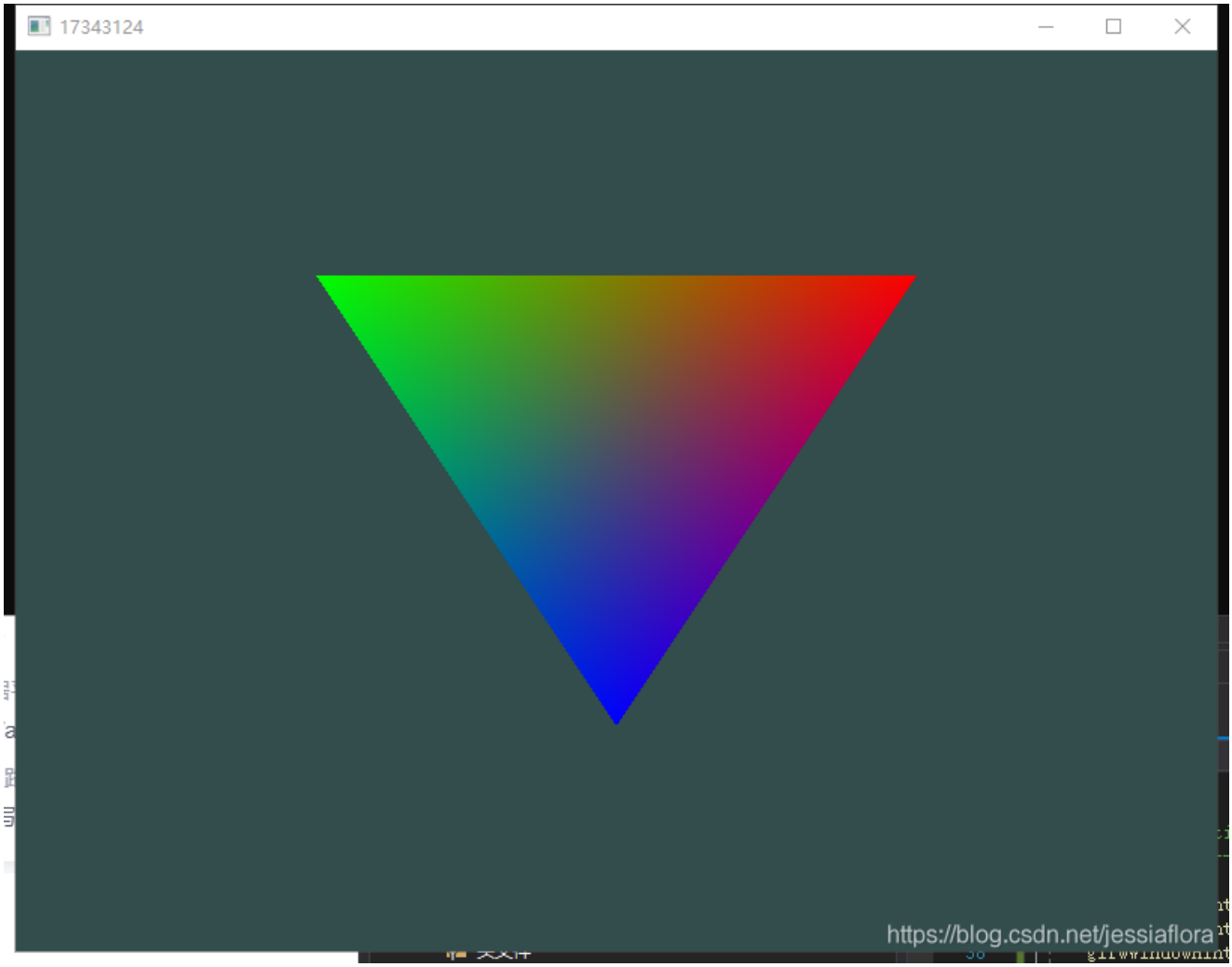
Task 2、完成了 着色器 章节之后，请修改顶点着色器让三角形上下颠倒。

直接把顶点着色器里的y坐标乘上一个负一：

```
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aColor;
out vec3 Color;
out vec3 posi;
void main() {
```

```
    gl_Position = vec4(aPos.x, -aPos.y, aPos.z, 1.0); // 上下颠倒
    posi = aPos;
    Color = aColor;
}
```

得到：



Task 3、完成了 纹理 章节之后，尝试用不同的纹理环绕方式，设定一个从 0.0f 到 2.0f 范围内的（而不是原来的 0.0f 到 1.0f ）纹理坐标。试试看能不能在箱子的角落放置4个笑脸。记得一定要试试其它的环绕方式。简述原因并贴上结果。

原因：
当把纹理坐标设置在范围外OpenGL会根据选择的环绕方式进行处理，例如GL_REPEAT是重复纹理图像；GL_MIRRORED_REPEAT 镜像放置重复图片，GL_CLAMP_TO_EDGE超出的部分会重复纹理坐标的边缘，GL_CLAMP_TO_BORDER超出的坐标为用户指定的边缘颜色。因此采用GL_REPEAT的方式就可以产生四个笑脸的结果。

结果：

Task 4、完成了 坐标系统 章节之后，对GLM的projection函数中的FoV和aspect-ratio参数进行实验。看能否搞懂它们是如何影响透视平截头体的。

参数	描述
FoV	field of view 视野
aspect-ratio	宽高比

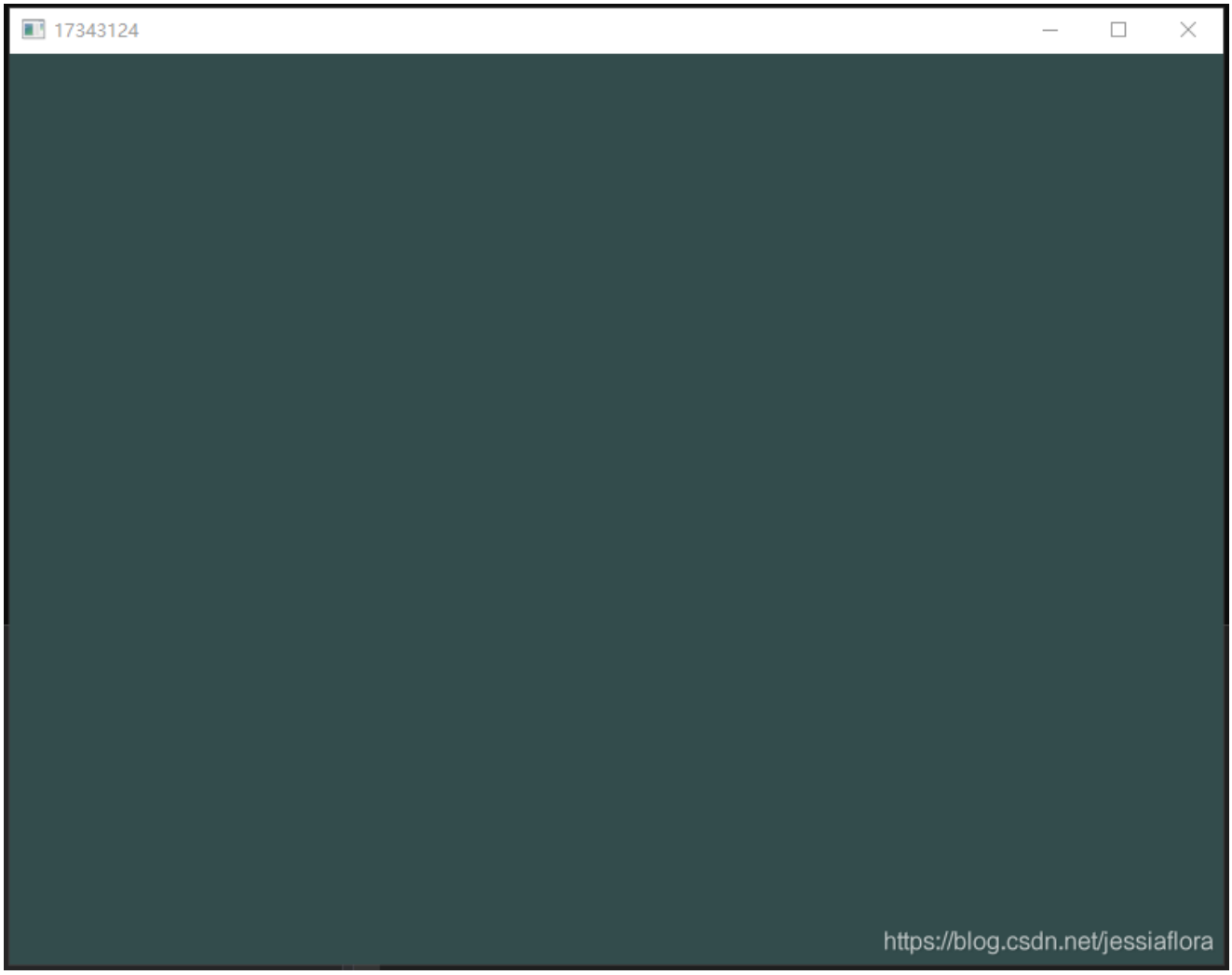
FoV类比于现实生活中镜头的焦段，也就是镜头的视野。镜头是焦段越小，视野越广，而焦段越长，视野越小。FoV角度的变化代表我们摄像机镜头焦段的变化。

aspect-ratio宽高比，决定了视野剪裁。因为渲染的内容最终要呈现到屏幕，这个范围外的内容将会被剪裁掉，即超出这个范围的将不可见。

Task 5、请按照顺序将跟着教程实现的运行结果贴出来，要求将运行出来的窗口的标题改成自己的学号。(Tip: glfwCreateWindow函数)

1. **你好，窗口**

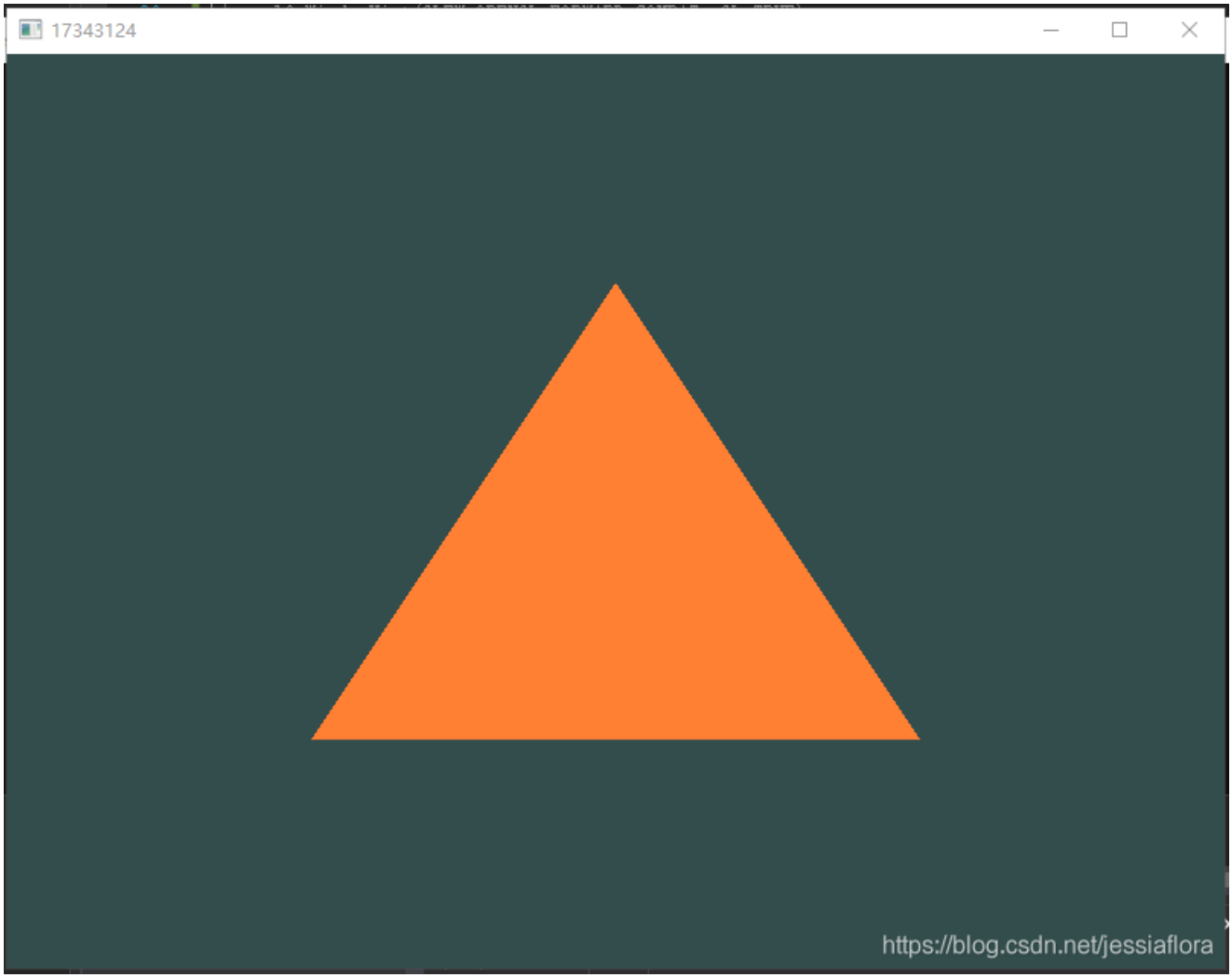
```
//将实现的窗口的标题改成自己的学号
GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "17343124", nullptr,
nullptr);
//修改颜色为墨绿色
glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
```



2. **你好，三角形**

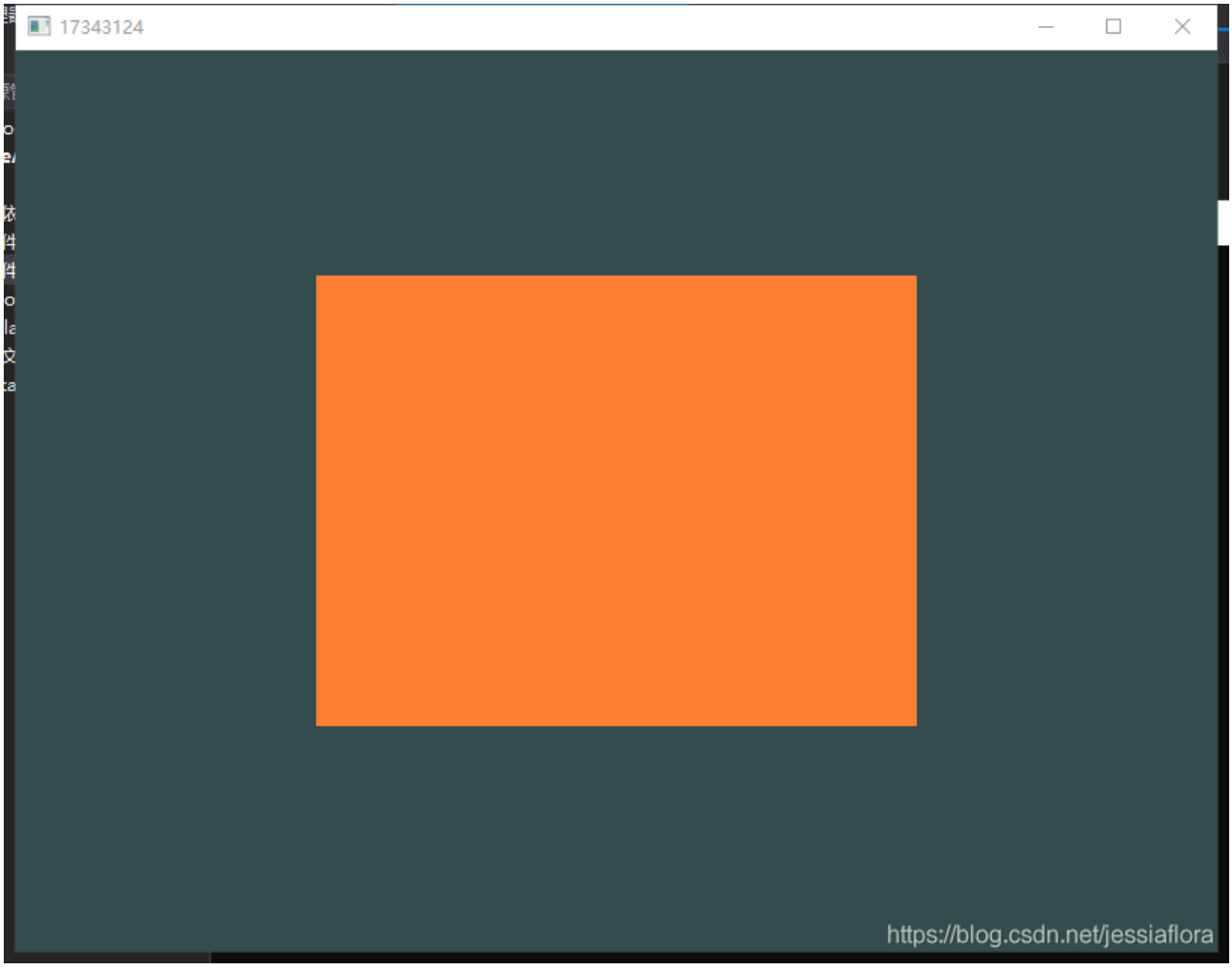
- 三角形:

```
GLfloat vertices[] = {
    0.5f,  0.5f, 0.0f,  // Top Right
    0.5f, -0.5f, 0.0f,  // Bottom Right
    -0.5f, -0.5f, 0.0f,  // Bottom Left
};
```

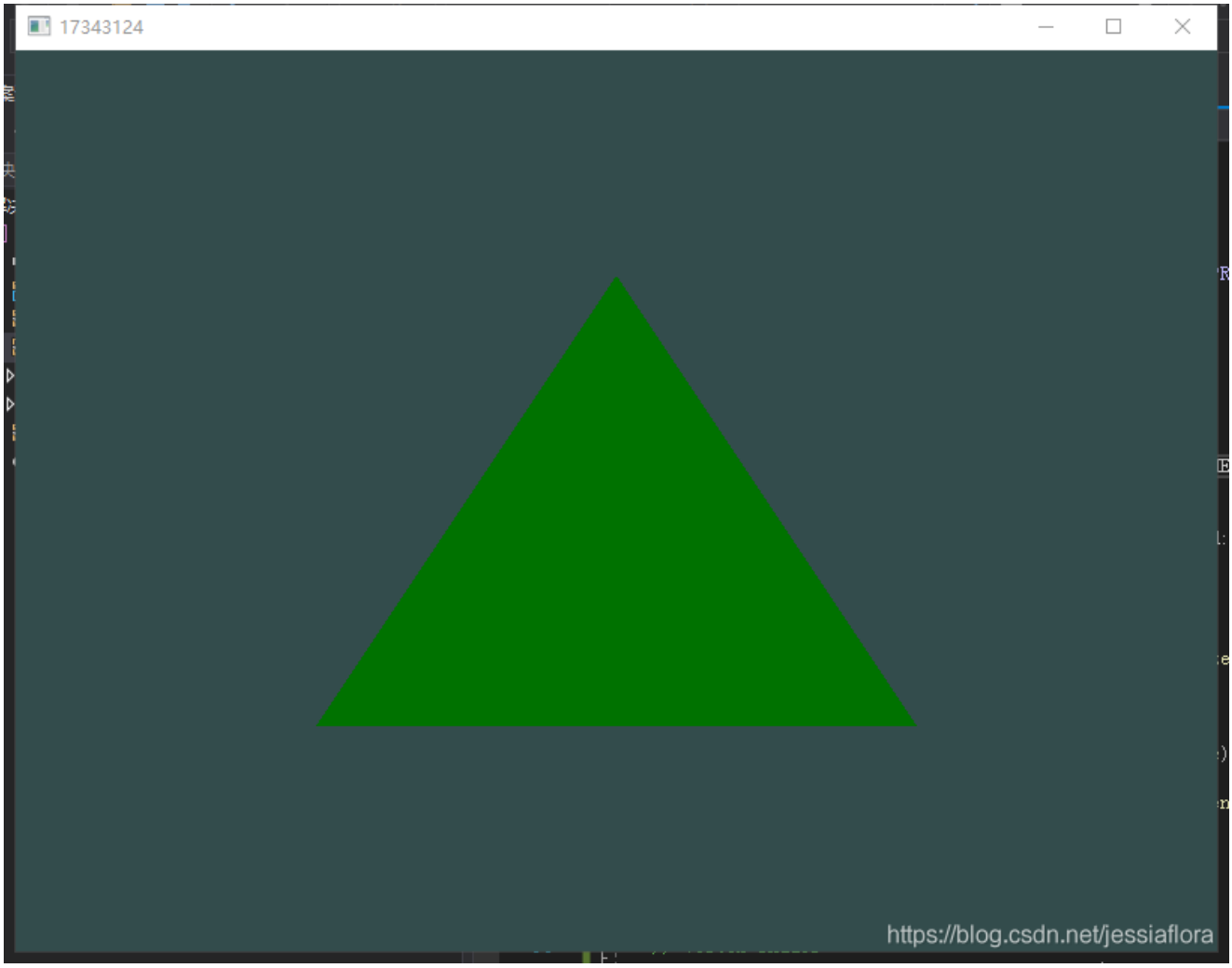


· 矩形

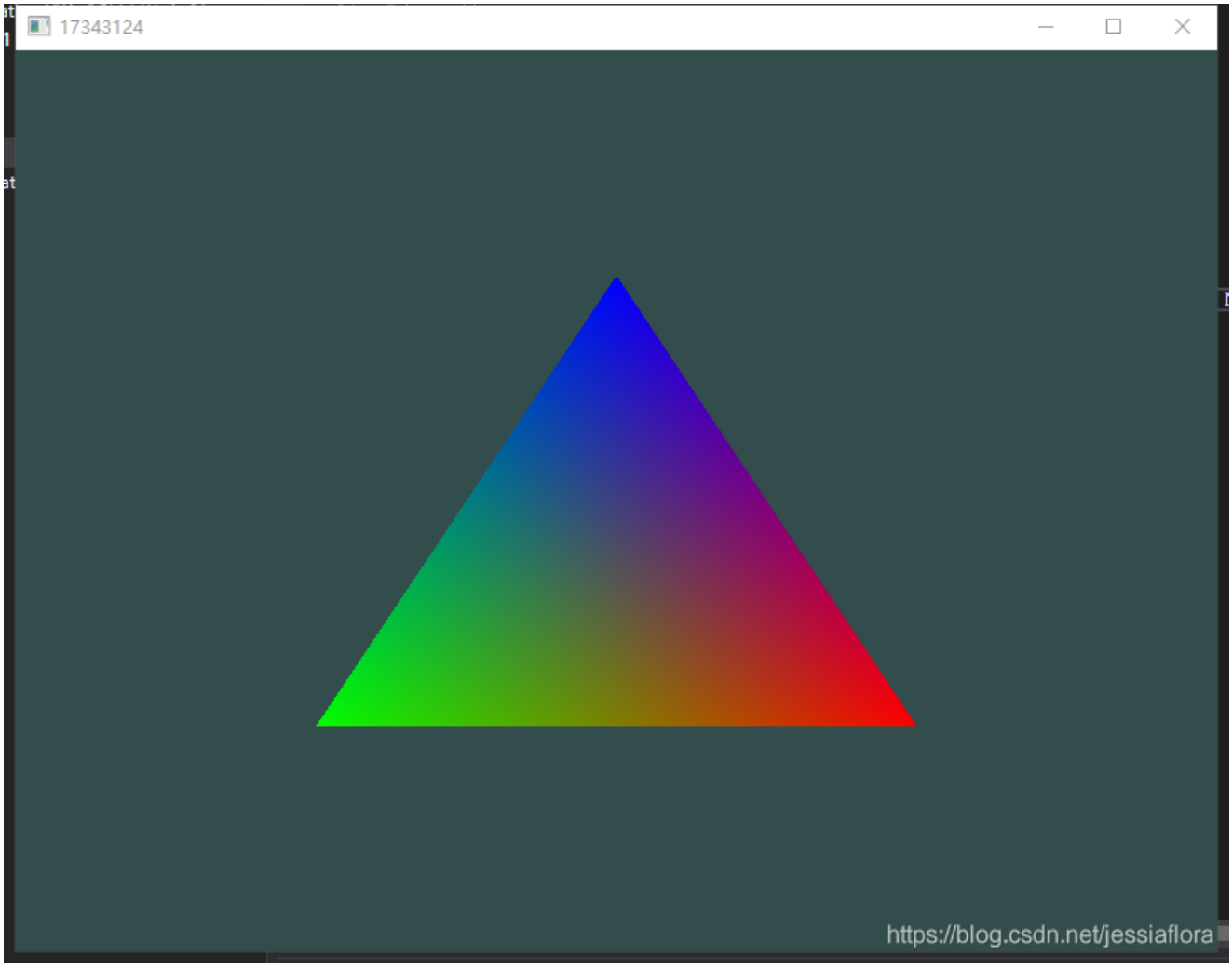
```
GLfloat vertices[] = {
    0.5f,  0.5f, 0.0f,  // Top Right
    0.5f, -0.5f, 0.0f,  // Bottom Right
    -0.5f, -0.5f, 0.0f,  // Bottom Left
    -0.5f,  0.5f, 0.0f  // Top Left
};
```



3. **着色器**




```
GLfloat vertices[] = {
    // Positions      // Colors
    0.5f, -0.5f, 0.0f,  1.0f, 0.5f, 0.0f,  // Bottom Right
    -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.5f,  // Bottom Left
    0.0f,  0.5f, 0.0f,  0.5f, 0.0f, 1.0f  // Top
 };
```



4. 纹理

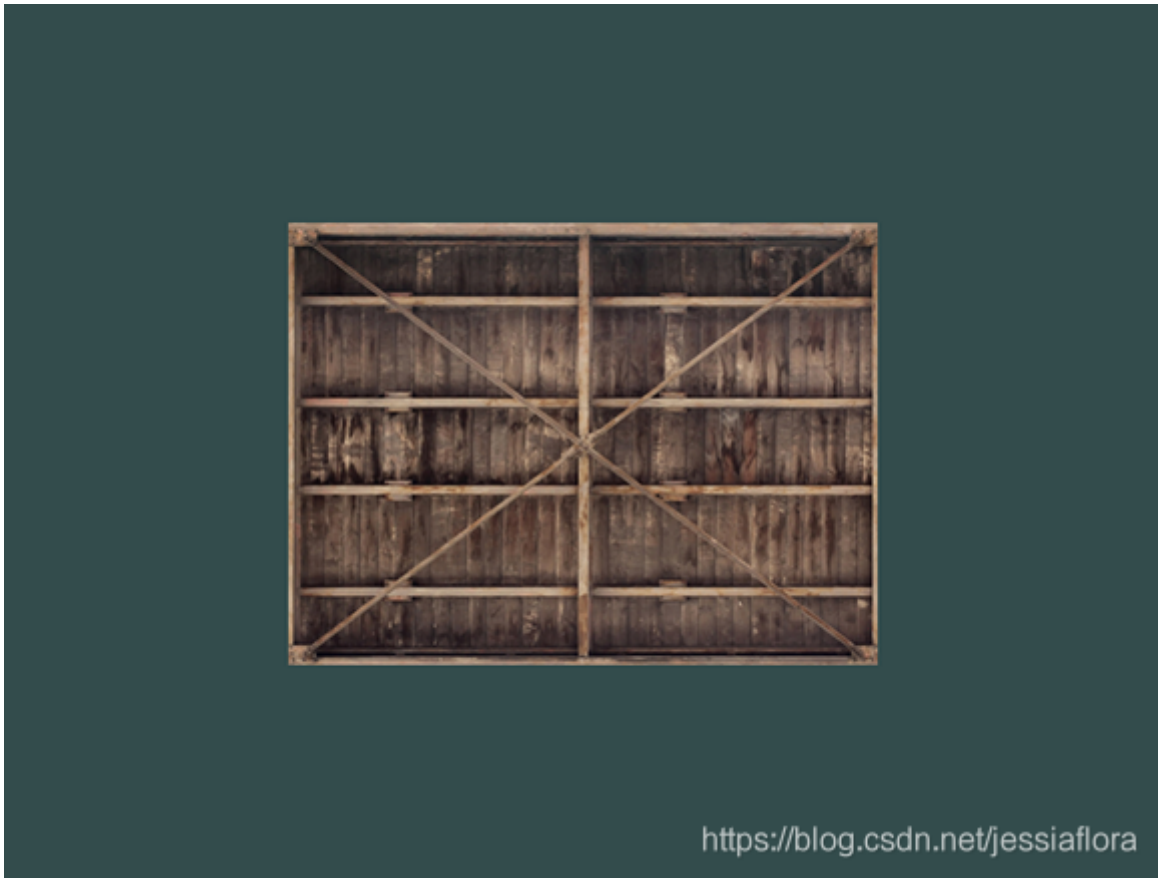
先加载纹理图：

```
        // 贴图
unsigned int TexBuffer;
glGenTextures(1, &TexBuffer);
glBindTexture(GL_TEXTURE_2D, TexBuffer);
        // 加载图片
int width, height, nrChannels;
unsigned char *data = stbi_load("Textures/test.jpg", &width, &height, &nrChannel
if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSI
        glGenerateMipmap(GL_TEXTURE_2D); // 生成MipMap即是多级渐远纹理
}
else {
        cout << "Load Texture failed." << endl;
}
stbi_image_free(data); // 释放内存
```

在渲染循环中加上这一句绑定纹理：

```
        glBindTexture(GL_TEXTURE_2D, TexBuffer);
```

结果：



修改片元着色器，把顶点颜色和贴图颜色相乘：

```
        FragColor = texture(ourTexture, TexCoord) * Color;
```

结果：


```
        // === 贴图 ===
stbi_set_flip_vertically_on_load(true); // 反转Y
int width, height, nrChannels;
unsigned char *data;
        // 第一张图
unsigned int TexBuffer1;
glGenTextures(1, &TexBuffer1);
glActiveTexture(GL_TEXTURE0); // 激活0号单元
glBindTexture(GL_TEXTURE_2D, TexBuffer1);
data = stbi_load("Textures/container.jpg", &width, &height, &nrChannels, 0);
if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSI
        glGenerateMipmap(GL_TEXTURE_2D);
}
else {
        cout << "Load Texture failed." << endl;
}
stbi_image_free(data);
        // 第二张图
unsigned int TexBuffer2;
glGenTextures(1, &TexBuffer2);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, TexBuffer2);
data = stbi_load("Textures/awesomeface.png", &width, &height, &nrChannels, 0);
if (data) {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UN
```

```
        glGenerateMipmap(GL_TEXTURE_2D);
}
else {
        cout << "Load Texture failed." << endl;
}
stbi_image_free(data);
```

在渲染循环前通过设置unifrom设置两张贴图的纹理单元ID，需要先use。

```
/*指定纹理*/
myShader.use();
myShader.setInt("texture1", 0);
myShader.setInt("texture2", 1);
```

在渲染循环中两个纹理都需要激活绑定。

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, TexBuffer1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, TexBuffer2);
```

5. 坐标系统

   在顶点着色器中传出裁剪空间的坐标。根据公式我们还需要三个矩阵：

```
glm::mat4 model;
model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glm::mat4 view;
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
glm::mat4 projection;
projection = glm::perspective(glm::radians(45.0f), 800.0f / 600.0f, 0.1f, 100.0f
```

然后在顶点着色器里面写一下uniform，相乘（注意顺序）：

```
#version 330 core

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aColor;
layout(location = 2) in vec2 aTexCoord;

out vec3 Color;
out vec2 TexCoord;

void main() {
  gl_Position = projection * view * model * vec4(aPos, 1.0);
  Color = aColor;
  TexCoord = aTexCoord;
}
```

在渲染循环中传入三个矩阵：

```
/*Uniform*/
myShader.setMat4("model", model);
myShader.setMat4("view", view);
myShader.setMat4("projection", projection);
```

结果：