实验 2：Triangles and Z-buffering

要求：

- **独立完成**
- **在截止时间之前提交**

---

1 总览

在上次实验中，虽然我们在屏幕上画出一个线框三角形，但这看起来并不是那么的有趣。所以这一次我们继续推进一步——在屏幕上画出一个实心三角形，换言之，栅格化一个三角形。上一次实验中，在视口变化之后，我们调用了函数 rasterize_wireframe(const Triangle& t)。但这一次，你需要自己填写并调用函数 rasterize_triangle(const Triangle& t)。

该函数的内部工作流程如下：

1. 创建三角形的 2 维 bounding box。
2. 遍历此 bounding box 内的所有像素（使用其**整数**索引）。然后，使用像素中心的屏幕空间坐标来检查中心点是否在三角形内。
3. 如果在内部，则将其位置处的**插值深度值** (interpolated depth value) 与深度缓冲区 (depth buffer) 中的相应值进行比较。
4. 如果当前点更靠近相机，请设置像素颜色并更新深度缓冲区

(depth buffer)。

你需要修改的函数如下：

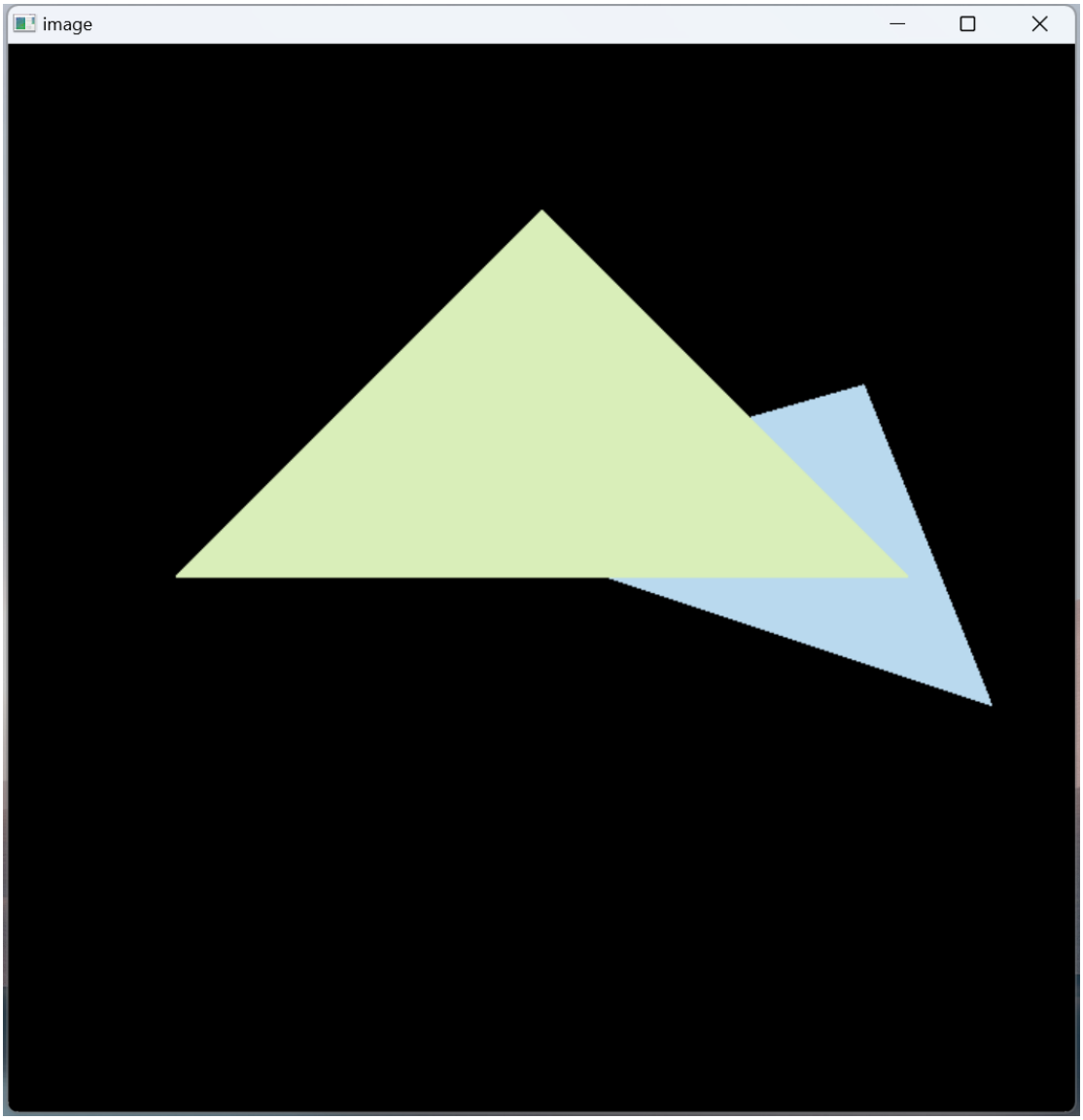
• rasterize_triangle(): 执行三角形栅格化算法

• static bool insideTriangle(): 测试点是否在三角形内。你可以修改此函数的定义，这意味着，你可以按照自己的方式更新返回类型或函数参数。

因为我们只知道三角形三个顶点处的深度值，所以对于三角形内部的像素，我们需要用**插值**的方法得到其深度值。我们已经为你处理好了这一部分，因为有关这方面的内容尚未在课程中涉及。插值的深度值被储存在变量 z_interpolated 中。

请注意我们是如何初始化 depth buffer 和注意 z values 的符号。为了方便同学们写代码，我们将 z 进行了反转，保证都是正数，并且越大表示离视点越远。

在此次实验中，你无需处理旋转变换，只需为模型变换返回一个单位矩阵。最后，我们提供了两个 hard-coded 三角形来测试你的实现，如果程序实现正确，你将看到如下所示的输出图像：



2 代码框架

下载本次实验的代码框架，你会注意到，在 main.cpp 下的 get_projection_matrix() 函数是空的。请复制粘贴你在**第一次实验**中的实现来填充该函数。

3 提高项

用 super-sampling 处理 Anti-aliasing：你可能会注意到，当我

们放大图像时，图像边缘会有锯齿感。我们可以用 super-sampling 来解决这个问题，即对每个像素进行 2 * 2 采样，并比较前后的结果(这里并不需要考虑像素与像素间的样本复用)。需要注意的点有，对于像素内的每一个样本都需要维护它自己的深度值，即每一个像素都需要维护一个 samplelist。最后，如果你实现正确的话，你得到的三角形不应该有不正常的黑边。