### 电计2003 邓人玮 20201071080 CG第二次作业报告

#### 任务1：阅读理解如下算法1、算法2，说明2是如何改进1的？

**答：**

**算法一是把三角形包围盒中的所有点都单独拿出来计算是否在三角形内部，而每次计算都要把当前点(x,y)带入三个三角形的边直线方程进行计算，然后得到对应的三个值e1,e2,e3。这三个值的计算过程完全没有利用上一次计算的结果。**

**而算法二是把三角形包围盒以扫描线为单位，只需每个扫描线的起始点(xmin,y)计算一次e1,e2,e3，下一个点（右侧一个单位,x+=1）对应的ei+1和当前点对应的ei只相差ai（第i个边的系数），ei+1=ei+ai。利用这一点可以大幅度简化运算，提高效率。**

#### 任务2：编程实现以上两个算法，并至少绘制5个不同的填充三角形。

##### 开发和运行环境说明

语言：ISO C++17

IDE：Visual Studio 2022

环境：Eigon:x64-windows 3.4.0；OpenCV:x64-windows 4.5.5

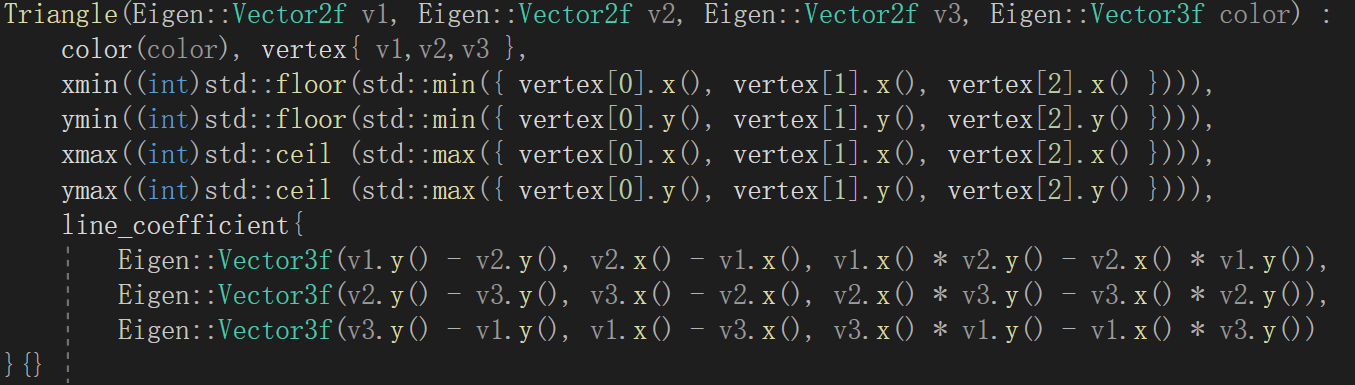
\*使用vcpkg打包集成到项目中

·Eigon方便管理2维数据和三维数据

·OpenCV用于把图像显示出来

##### 算法设计思想和实现技术

首先定义一个三角形类Triangle(见**Triangle.h**)，构造函数要求三个顶点的坐标和三角形颜色，并且在构造函数中就计算出包围盒的4个值，同时还计算出三条边每一条边的直线方程，每条边对应保存三个系数。

注意在计算每一条边的直线方程时，除了要注意把公式(a=y1 - y2, b=x2-x1, c= x1\*y2 - x2\*y1)对应上之外，每条线还必须得确定方向头尾衔接（假设三个点是v1v2v3），比如v1v2，v2v3，v3v1，这样才能确保后续判定的正确性（顺时针方向：三个值都是负值表示在内；逆时针方向：三个值都是正值表示在内）。

另外再实现一个函数float edge\_equation(const size\_t idx, const float x, const float y)const 用于返回把点坐标带入对应直线方程计算的值。

由于需要生成图像，还得需要一个光栅化器，于是定义光栅化器Rasterizer类(见**Rasterizer.h, Rasterizer.cpp**)，该类可以生成一个帧缓存frame\_buf用来创建cv::Mat对象image，并最终通过cv::imshow来输出图像。在该类中通过对frame\_buf的值作修改即可达到对对应像素修改颜色的效果。

类中实现了下列两个函数分别对应两个算法，2是优化的算法2。这两个算法会计算出光栅化传入的三角形需要的所有的点

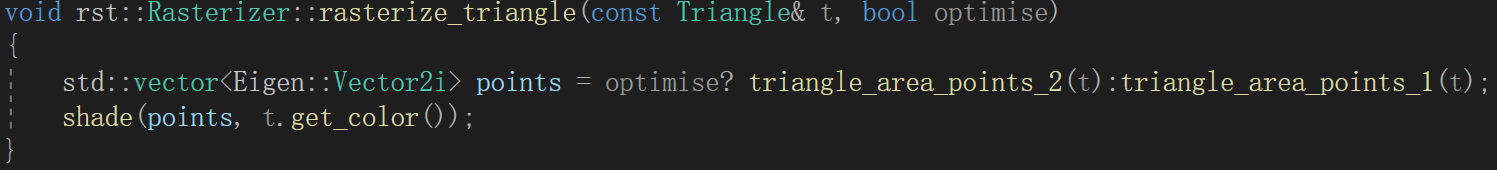
static std::vector<Eigen::Vector2i> triangle\_area\_points\_1(const Triangle& t);

static std::vector<Eigen::Vector2i> triangle\_area\_points\_2(const Triangle& t);

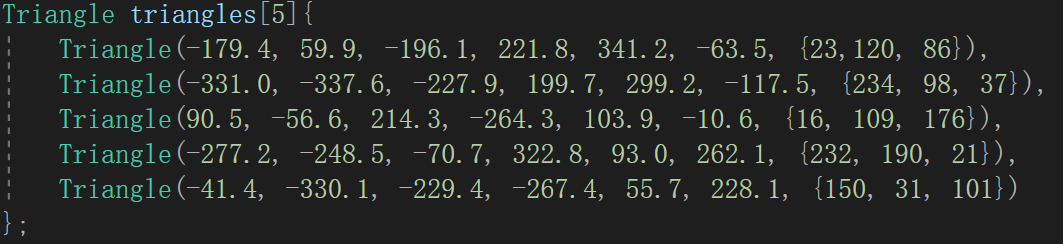
利用其返回的点列表传入以下函数即可修改frame\_buf的对应值

void shade(const std::vector<Vector2i>& points, const Vector3f& color);

都实现好之后光栅化三角形的函数rasterize\_triangle的实现如下



在主函数中(见**main.cpp**)，首先定义画布：宽700，高700，并初始化一个光栅化器的对象r。

随机生成在(-350,-350)到(350,350)之间的5个三角形的顶点坐标，以及对应的随机颜色，初始化Triangle数组（长度为5）。初始化列表如下：

然后分别计时执行两种算法，并显示图片以及输出算法执行时间。

##### 程序的运行结果与分析

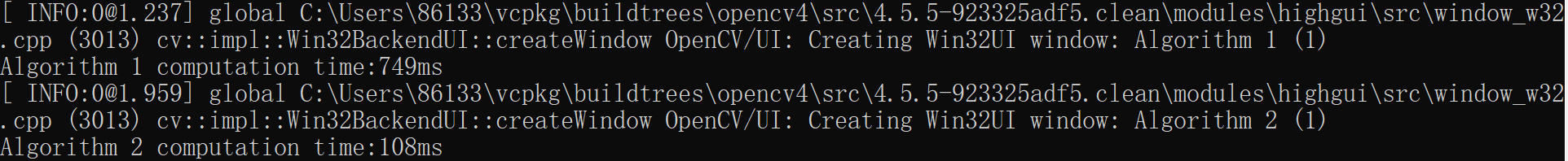
算法一：



算法二：



运行时间：



性能分析:

由生成的图片可见，二者生成的图像都正确；

由运行时间可见优化后的算法执行效率已经达到了原算法效率的7倍多，非常显著。

问题及改进：

可以发现图像中斜边有明显锯齿，可以通过超采样反走样技术来稍加改进。