**计算机图形学实验报告**

学号：16340054

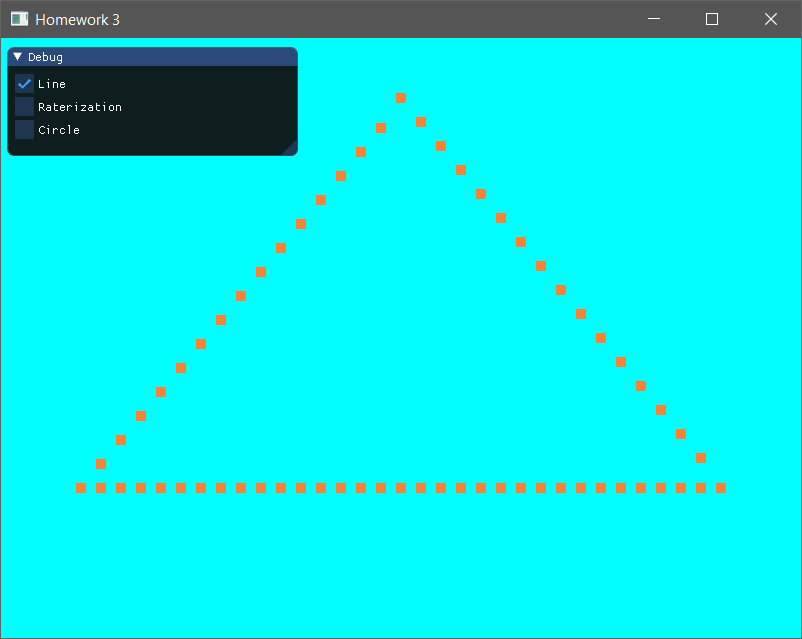
姓名：戴馨乐

学院：数据科学与计算机学院

作业：Homework3

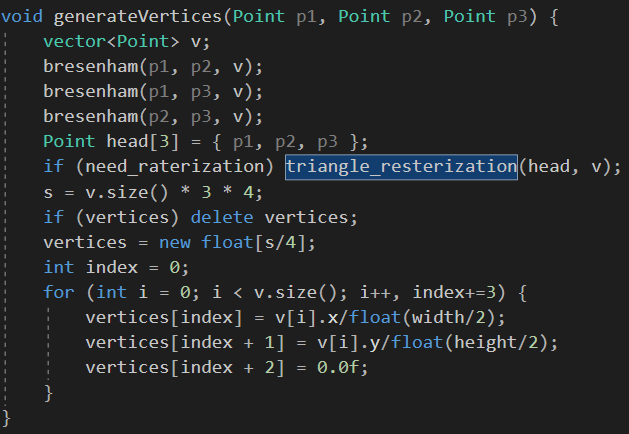
Basic：

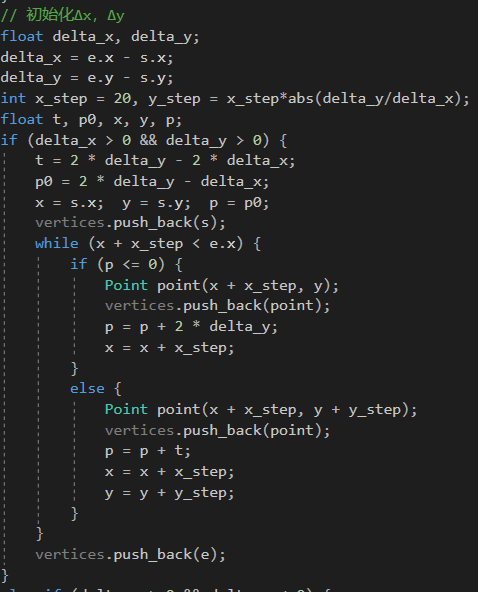
1. 使用Bresenham算法画一个三角形边框
   1. 运行截图：



* 1. 核心代码：

将3个点两两组合，通过Bresenham算法来分别画出3条直线

Bresenham画直线（部分）



* 1. 实现思路：

**Bresenham算法描述：**

对于一条从到的直线，初始化如下变量：

定义一个x变化的步长和一个y变化的步长，那么：

初始化x为，y为，p为：

不断进行如下过程：

若：

选取的下一个点为

更新x为

更新p为

若：

选取的下一个点为

更新x为

更新y为

更新p为

直到

**实现思路：**

可以看到，上面的算法仅仅考虑了斜率为0到的直线，对于其他直线，后来我查到可以通过直线变换来得到。但是当时实现的时候没有这样做，而是做了一个分类讨论。

首先确认起点和终点。规则是：选择x坐标小的为起点，大的为终点；若两点的x坐标一样，则选择y坐标小的为起点，大的为终点，假如一样，则这两点是同一点，不需要画直线。

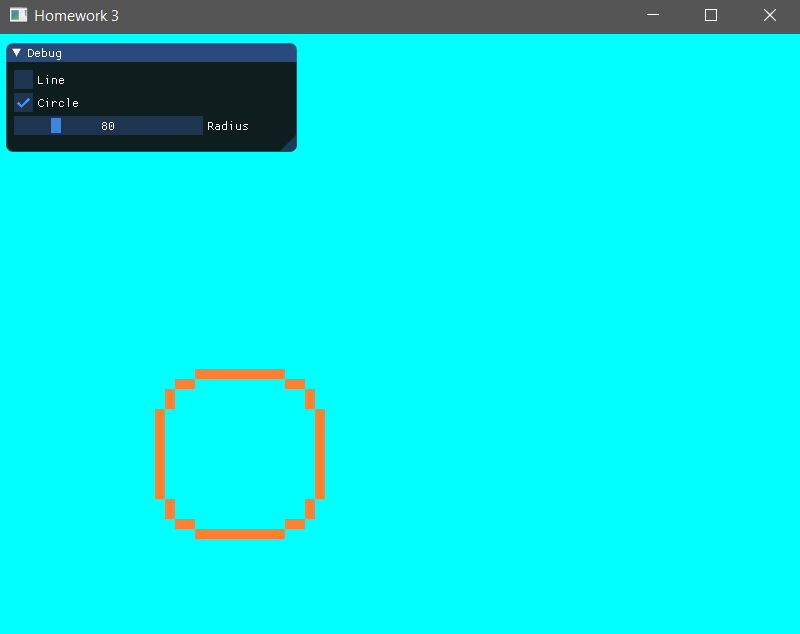
然后将直线根据和来区分出4类直线：

* + 1. 斜角为到
    2. 斜角为到
    3. 斜角为
    4. 斜率不存在（垂直）

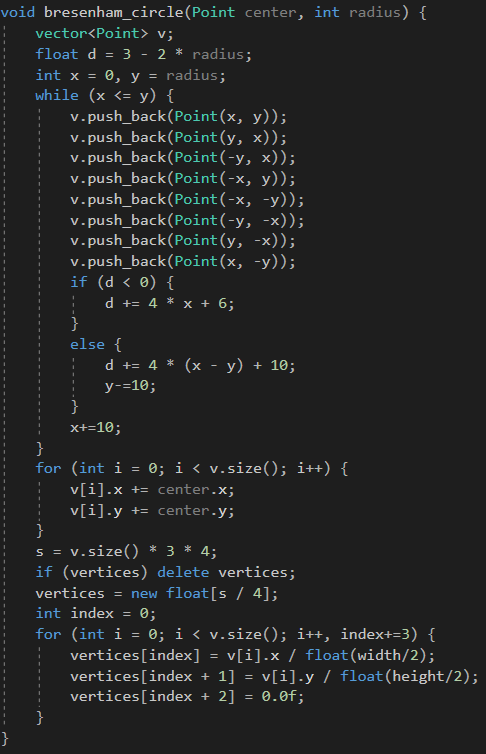
其中，第二种直线需要重新推导一下，稍微有些不一样，但是还是类似的。

可以看到代码是蛮累赘的，假如通过直线变换将会简洁很多。

1. 使用Bresenham算法画一个圆
   1. 运行截图：



* 1. 核心代码：



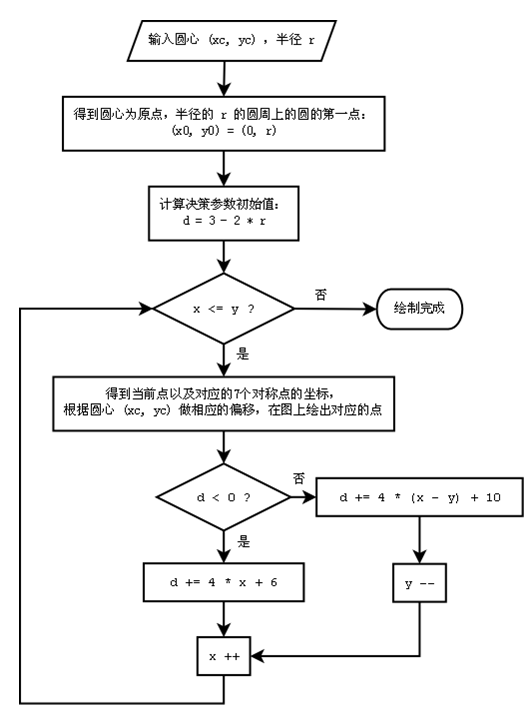
* 1. 实现思路：

**算法描述：**

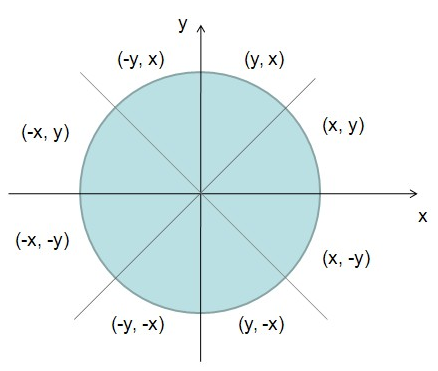
Bresenham算法画圆的算法可以通过下面这张流程图可以很清楚的展示出来：

假设：

圆心为，半径为



其中7个相对于圆心的对称点是如下7个：



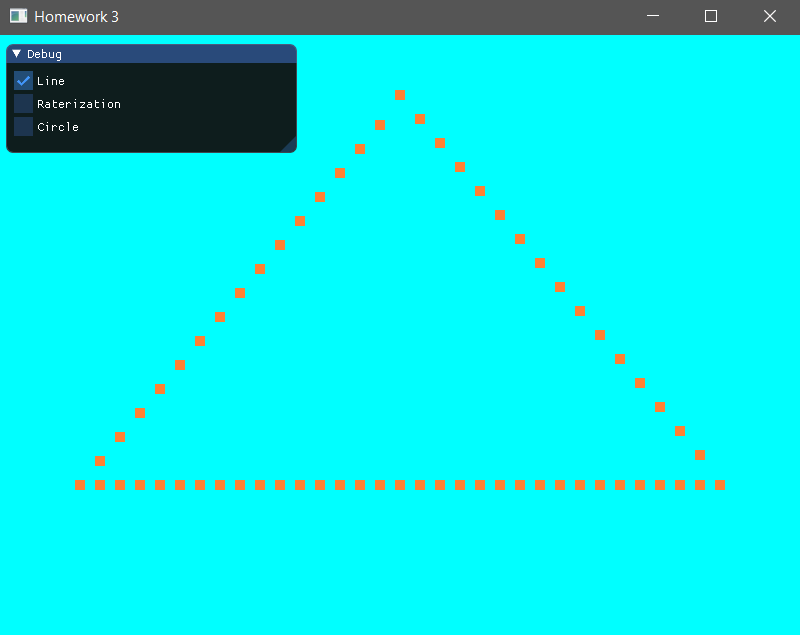
为什么需要对称点？这是因为圆的对称性，所以我们只需要计算的圆周，其余的部分都可以通过对称得到。

**实现思路：**

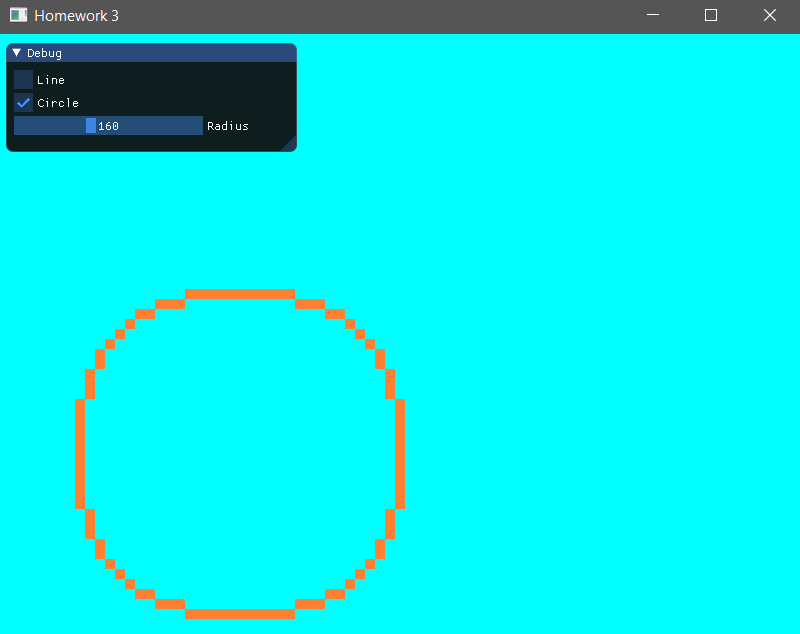
首先，为了简化对称点的计算，我们可以假设圆心为，按照算法流程图来求出圆周上的所有点

然后，将坐标系的原点移动到。将每一个点的x坐标加上，y坐标加上

1. 添加菜单栏，可以选择是三角形边框还是圆，以及能调整圆的大小
   1. 运行截图：
      1. 选择直线还是圆

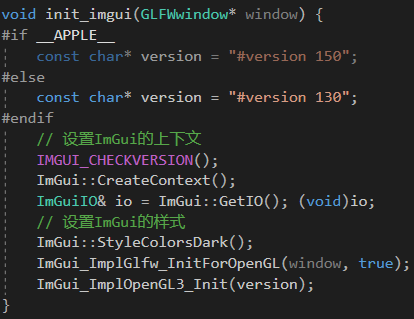


* + 1. 调整圆的半径的大小



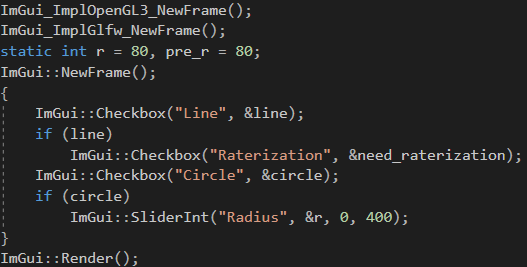
* 1. 核心代码及实现思路：

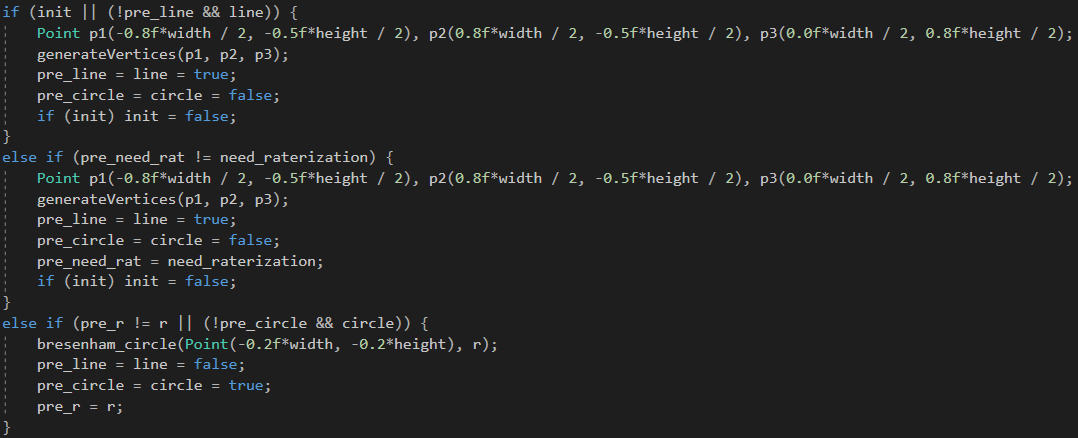
**初始化ImGui：**



**ImGui控件的设置和渲染：**

这里我们需要的是3个Checkbox和一个Slide来调整圆的半径。其中，是否光栅化的Checkbox由line是否选中来决定是否显示，Slide由circle是否选中来决定是否显示

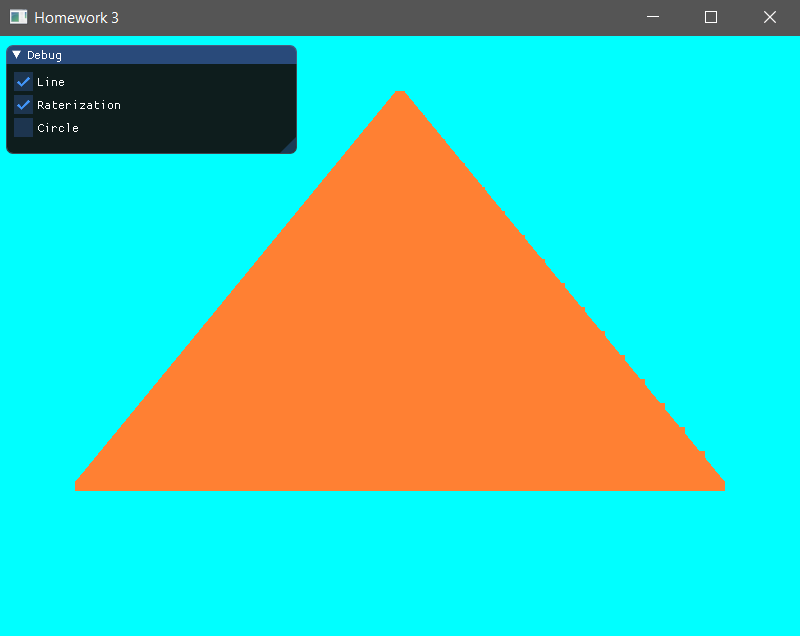


CheckBox选择决定的逻辑：

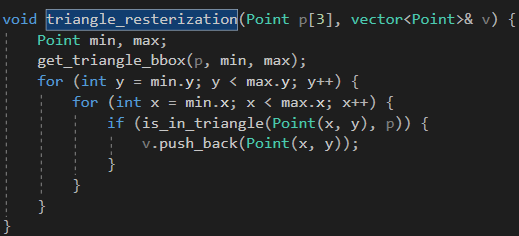
其中init是表示是否是第一次打开，假如是，则画的是Bresenham画直线画出来的三角形边框。另外，pre\_line和pre\_circle是为了记录上一次line和circle的选择值，来判断选项是否改变，从而避免了每次渲染都需要重新计算点，避免不必要的运算。need\_raterization和pre\_need\_rat同理。

Bonus：

1. 使用三角形光栅转换算法，用和背景不同的颜色，填充你的三角形
   1. 运行截图：

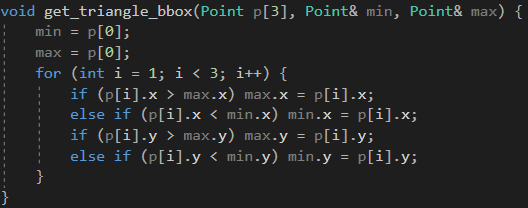


* 1. 核心代码：

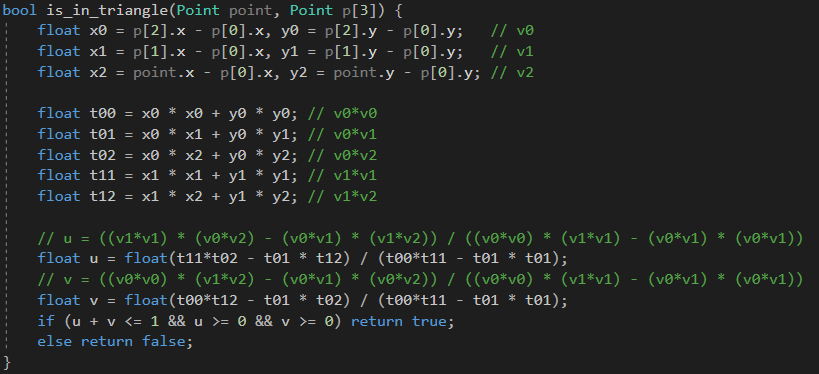
三角形光栅转换算法：

辅助函数：

求出三角形的包围盒



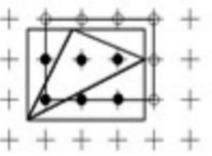
判断一个点是否在三角形内



* 1. 实现思路：

三角形光栅算法思路如下：

首先，对于一个三角形，求解其包围盒



可以理解为一个可以包含了整个三角形的最小矩形。求解包围盒的目的是避免扫描整个图片，仅仅扫描尽量少的点

然后遍历包围盒内的每一个点：

若该点在三角形内，则填充该点

否则，继续