Homework3

此次作业需要掌握Brehensam Algorithm来绘制直线和圆，从而更好地理解光栅化（Rasterization）的原理。此外还需要修改ImGUI界面以更换显示内容。

绘制三角形

绘制三角形使用的是Bresenham's line algorithm（直线算法），它是用来描绘由两点所决定的直线的算法。这个算法只会用到较为快速的整数加法、减法（或移位），不用进行较慢的浮点乘除。

如从一点(x0, y0)到(x1, y1)，它们组成的直线为：

$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

则对每一点x，其直线上对应y值为：

$$\frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

因为斜率可预先计算，这可以减少许多运算次数。此外，每次计算像素点时，需要计算其与直线间的误差，即每一点x相对的像素点之y值和在直线上实际y值得差距。每当x的值增加1，误差的值就会增加m，当误差的值超出0.5，直线就会比较靠近下一个点，因此y的值便加1，误差减1（复原）。

为了得到一个一般化的算法，需要扩展算法使其在在反方向、斜率绝对值大于一等情况下都能够正常工作。如：斜率为负时，误差超出0.5则y改为减1（而非加1）；斜率绝对值大于1时，可在绘画中交换x和y（反函数的原理）。

最后，可以看到上面所使用的表达式中还是存在浮点数的运算，可以对算法中所有的分数表达式都乘deltax（即x1 - x0），就可用整数来表示它们。最终的伪代码程序：

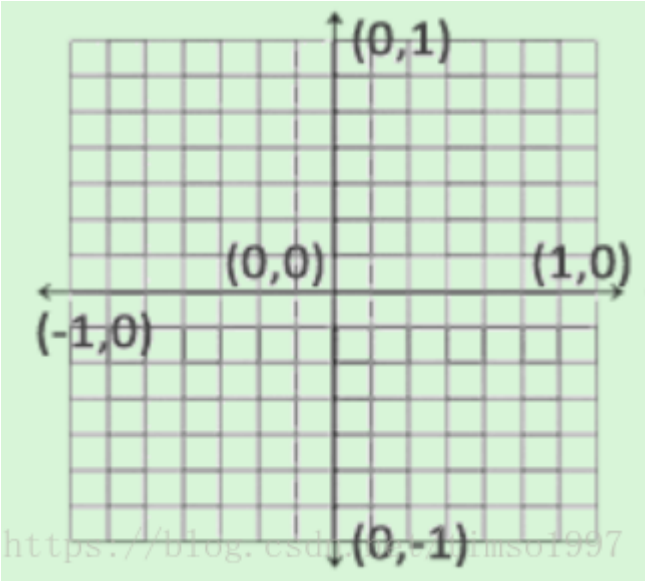
```
function line(x0, x1, y0, y1)
    boolean steep := abs(y1 - y0) > abs(x1 - x0)
    if steep then
        swap(x0, y0)
        swap(x1, y1)
    if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)
    int deltax := x1 - x0
    int deltay := abs(y1 - y0)
    int error := deltax / 2
    int ystep
    int y := y0
    if y0 < y1 then ystep := 1 else ystep := -1
    for x from x0 to x1
        if steep then plot(y,x) else plot(x,y)
        error := error - deltay
```

```
        if error < 0 then
            y := y + ystep
            error := error + deltax
```

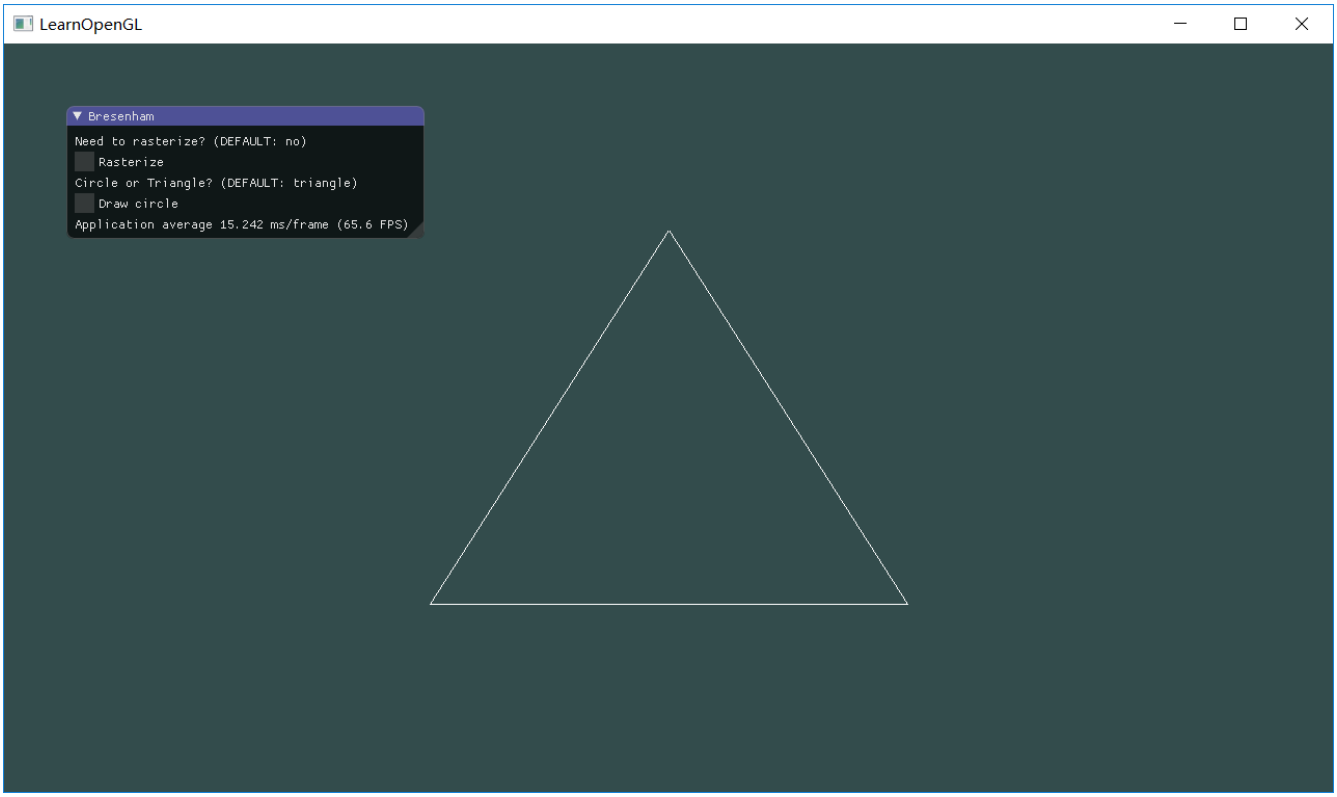
具体实现中，我将坐标数据抽象成一个struct，并添加几个Bresenham算法相关的函数，如对三角形的绘制：

```
struct Point {
    int x;
    int y;
    Point(int x_, int y_) {
        x = x_;
        y = y_;
    }
};
vector<Point> drawTriangle(Point, Point, Point);
```

得到一系列点的数据后，还需要注意其与opengl坐标之间的转换，因为opengl视口的坐标是-1到1的：

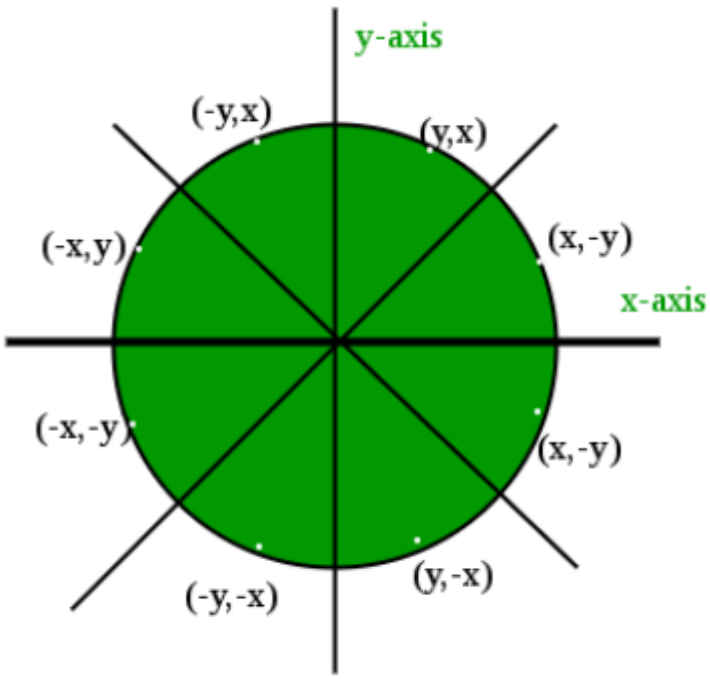


绘制结果：



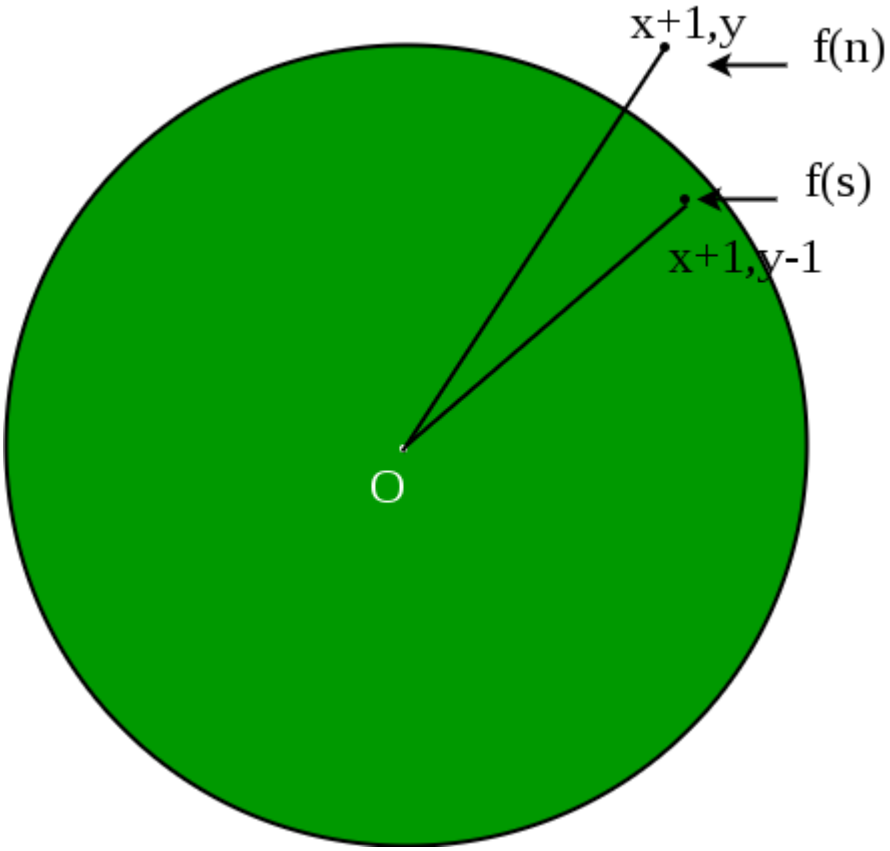
绘制圆形

绘制圆形同样使用的是Bresenham算法，首先假设将圆分为八个区域：



For a pixel (x,y) all possible pixels in 8 octants.

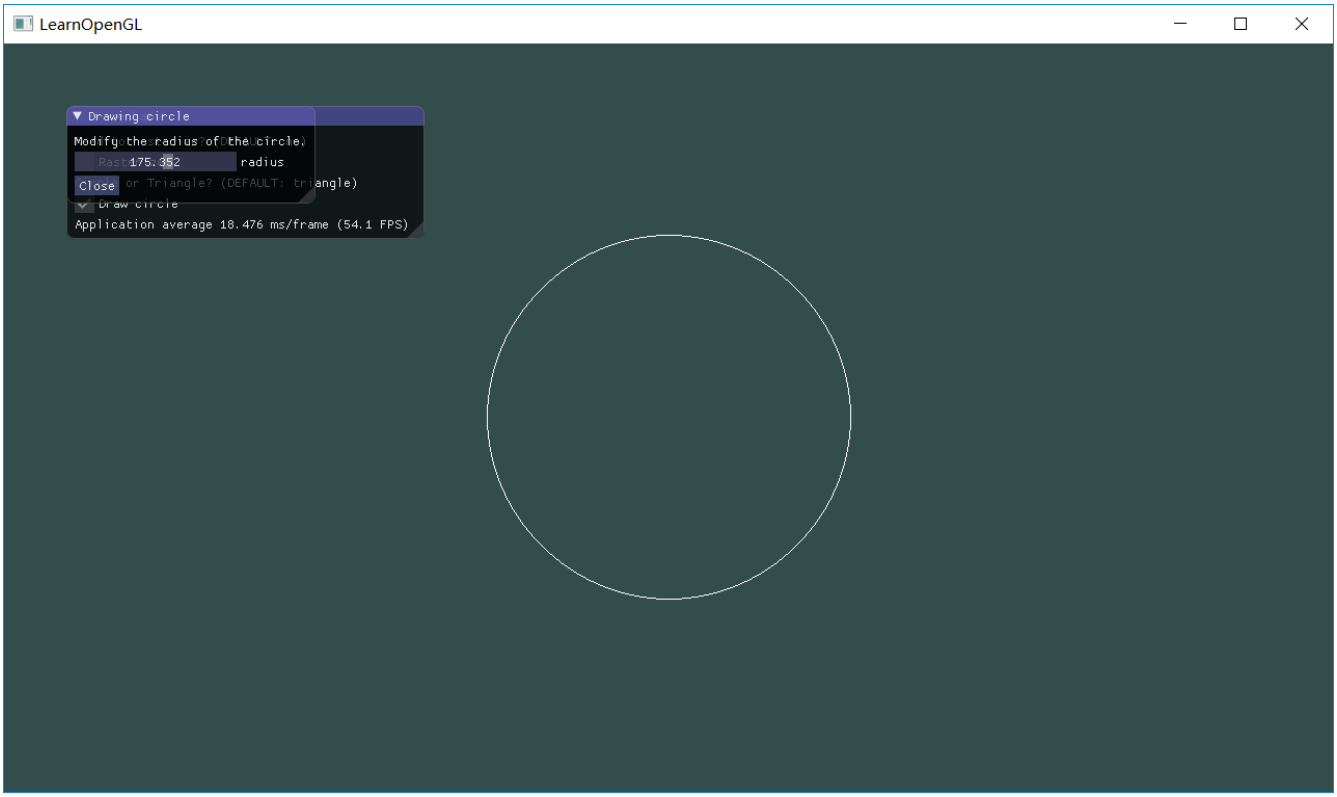
对于第一象限上45°这个区域来说，一点(x, y)的下一个要描绘的点只可能是(x+1, y)或(x+1, y-1)这需要看谁距离圆弧更近。其他区域可以由这种情况推广：



这里使用一个参数d来判定选择哪个点，若d>0则选择(x+1, y-1)，否则选择(x+1, y)。d的选择需要借助判别函数：`F(x, y) = x^2 + y^2 - R^2`，这个函数判定点是否在圆上。假设(x+1, y)和(x+1, y-1)分别为P1和P2，而M是P1和P2的中点，则M的坐标为(x+1, y–0.5)。通过判别函数，若M在圆内，则选择P1，否则选择P2。代入判别函数进行推导最终可以得到参数d的表示：

```
d = F (x + 1, y - 0.5) = (x + 1)^2 + (y - 0.5)^2 - R^2
//若d小于0
d’ = F (x + 2, y - 0.5) = (x + 2)^2 + (y - 0.5)^2 - R^2
d’ = d + 2x + 3
//若d大于0
d’ = F (x + 2, y - 1.5) = (x + 2)^2 + (y - 1.5)^2 - R^2
d’ = d + 2(x - y) + 5
```

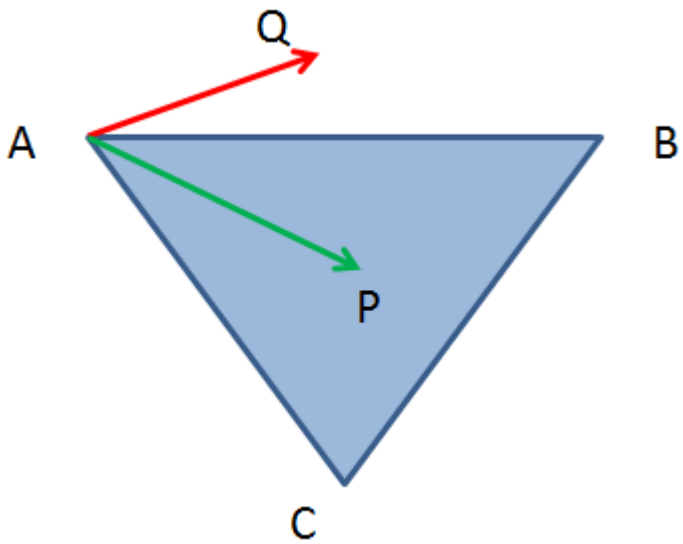
同样经过坐标转换后绘制出结果：



三角形光栅化填充颜色

为了填充三角形内部，必须要判定一个点是否在三角形的内部，我这里采用的是同向法。

假设点P在三角形内，会有这样的规律：当选定线段AB时，点C位于AB的右侧，同理选定BC时，点A位于BC的右侧，最后选定CA时，点B位于CA的右侧，所以当选择某一条边时，我们只需验证点P与该边所对的点在同一侧即可。如图：



可以先求出三条边的方程，然后利用同向法进行判断，求出辅助u、v、w：

```
u = fa(x,y)*fa(a1,a2)
v = fb(x,y)*fb(b1,b2)
w = fc(x,y)*fc(c1,c2)
```

判定条件为没有一个为负数，则在三角形内（或边上）。绘制结果：