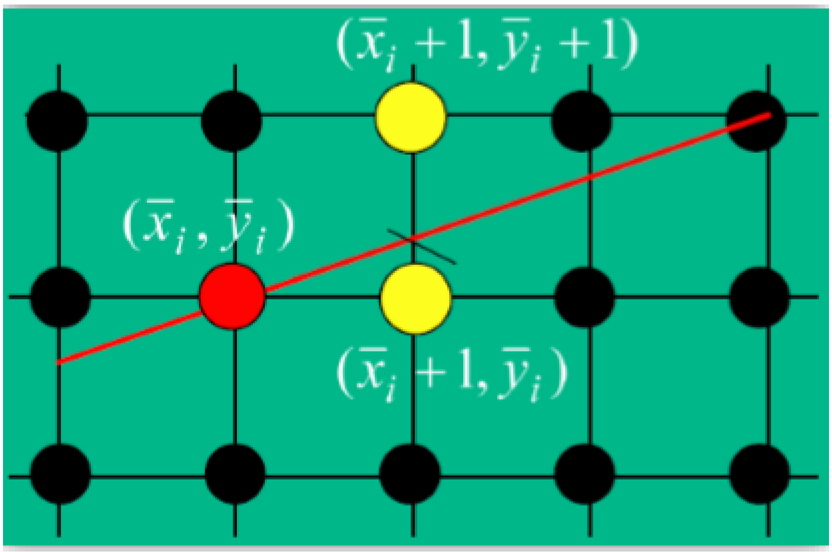
计算机图形学作业（二）

陈伟桐 16340034

Bresenham算法画直线

原理

首先，观察下图：

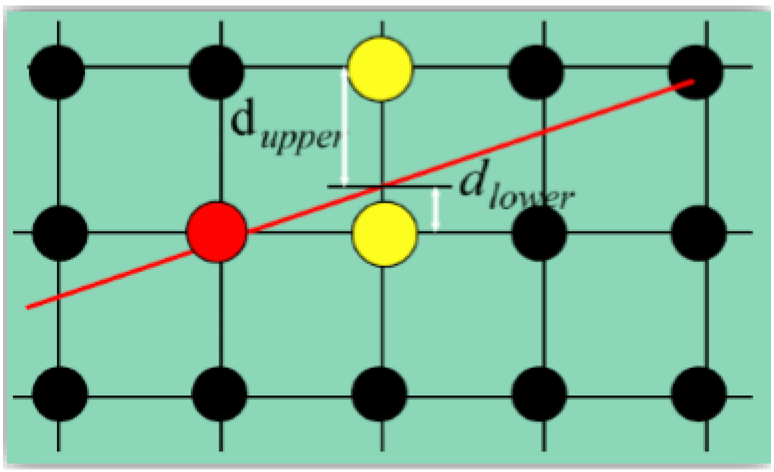


设一条直线为y=mx+B，那么上图图中的参数为：

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = mx_{i+1} + B = m(x_i + 1) + B$$

然后观察下图：



在图中，红色点为当前的点，我们要计算出下一个点是取高位的黄色点，还是低位的黄色点，就要比较这两个点谁距离直线最近，结合之前的图，可得d_{upper}和d_{lower}的大小 。

$$d_{upper} = \overline{y}_i + 1 - y_{i+1} = \overline{y}_i + 1 - mx_{i+1} - B$$
$$d_{lower} = y_{i+1} - \overline{y}_i = mx_{i+1} + B - \overline{y}_i$$

如果d_{lower}-d_{upper}>0，则取上方的黄色点，否则取下方的黄色点。 设定一个参数p_i，如下图：

$$\begin{aligned} p_i &= \Delta x \bullet (d_{lower} - d_{upper}) = 2\Delta y \bullet (x_i + 1) - 2\Delta x \bullet \overline{y}_i + (2B-1)\Delta x \\ &= 2\Delta y \bullet x_i - 2\Delta x \bullet \overline{y}_i + \boxed{(2B-1)\Delta x + 2\Delta y} \\ &= 2\Delta y \bullet x_i - 2\Delta x \bullet \overline{y}_i + c \end{aligned}$$

经过计算可得：

$$p_0 = 2\Delta y - \Delta x$$

且有以下关系：

$$\begin{aligned} p_{i+1} - p_i &= (2\Delta y \bullet x_{i+1} - 2\Delta x \bullet \overline{y}_{i+1} + c) - (2\Delta y \bullet x_i - 2\Delta x \bullet \overline{y}_i + c) \\ &= 2\Delta y - 2\Delta x(\overline{y}_{i+1} - \overline{y}_i) \end{aligned}$$

If $p_i \le 0$ **then** $\overline{y}_{i+1} - \overline{y}_i = 0$ **therefore**

$$p_{i+1} = p_i + 2\Delta y$$

If $p_i > 0$ **then** $\overline{y}_{i+1} - \overline{y}_i = 1$ **therefore**

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x$$

算法

绘制斜率在[0,1]范围内的直线算法如下：

- **draw** (x_0, y_0)
- **Calculate** $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x, p_0 = 2\Delta y - \Delta x$
- **If** $p_i \leq 0$ **draw** $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y}_i)$

  and compute $p_{i+1} = p_i + 2\Delta y$
- **If** $p_i > 0$ **draw** $(x_{i+1}, \bar{y}_{i+1}) = (x_i + 1, \bar{y}_i + 1)$

  and compute $p_{i+1} = p_i + 2\Delta y - 2\Delta x$
- **Repeat the last two steps**

拓展

由于使用Bresenham算法只能绘制斜率在[0,1]范围内的直线，那么要绘制其它斜率范围的直线，就要进行坐标平移、对称变换。

首先，我在代码中输入三个正整数点坐标，由于在OpenGL中坐标范围是 -1.0f~1.0f，所以我定义了一个函数 `normalize`，负责把正整数转化为 -1.0f~1.0f 的浮点数。因为我的窗口大小是800*800，所以只需要写成以下形式：

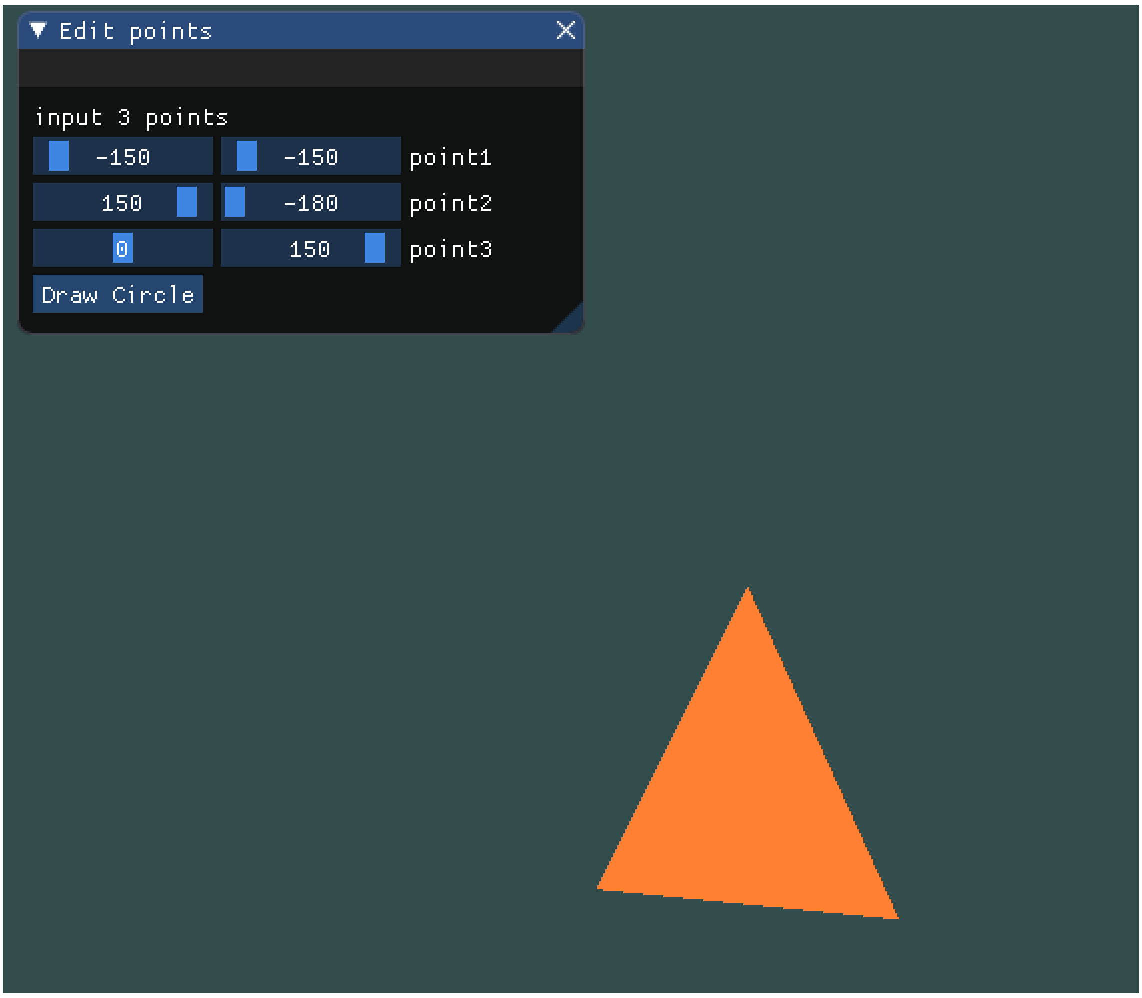
```
float normalize(int input) {
        float result = float(input) / 800;
        return result;
}
```

然后，对于给出的两个点，我们要绘制这两点的直线，首先计算出它们的斜率，若斜率不存在，则固定横坐标，循环纵坐标画出一个个点；若斜率在[0,1]范围，则以Bresenham算法绘制；否则，按以下方法绘制直线：

1. 找出这两点谁的横坐标较小，然后把横坐标较小的点平移到原点，横坐标较大的点平移到对应位置。
2. 对横坐标较大的点做一系列变换（关于直线y=x或y=0等对称），使得两点斜率在[0,1]之内
3. 使用Bresenham算法绘制出直线，然后再把这条直线作步骤2相反的一系列变换即可。

结果

画出三角形并光栅化后，结果如下图所示：



使用光栅化算法填充三角形

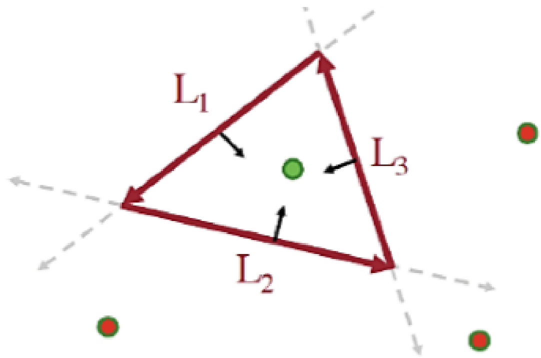
光栅化算法有三种：

1. Edge-walking
2. Edge-equation
3. Barycentric-coordinate based

我使用的算法是Edge-equation

算法伪代码

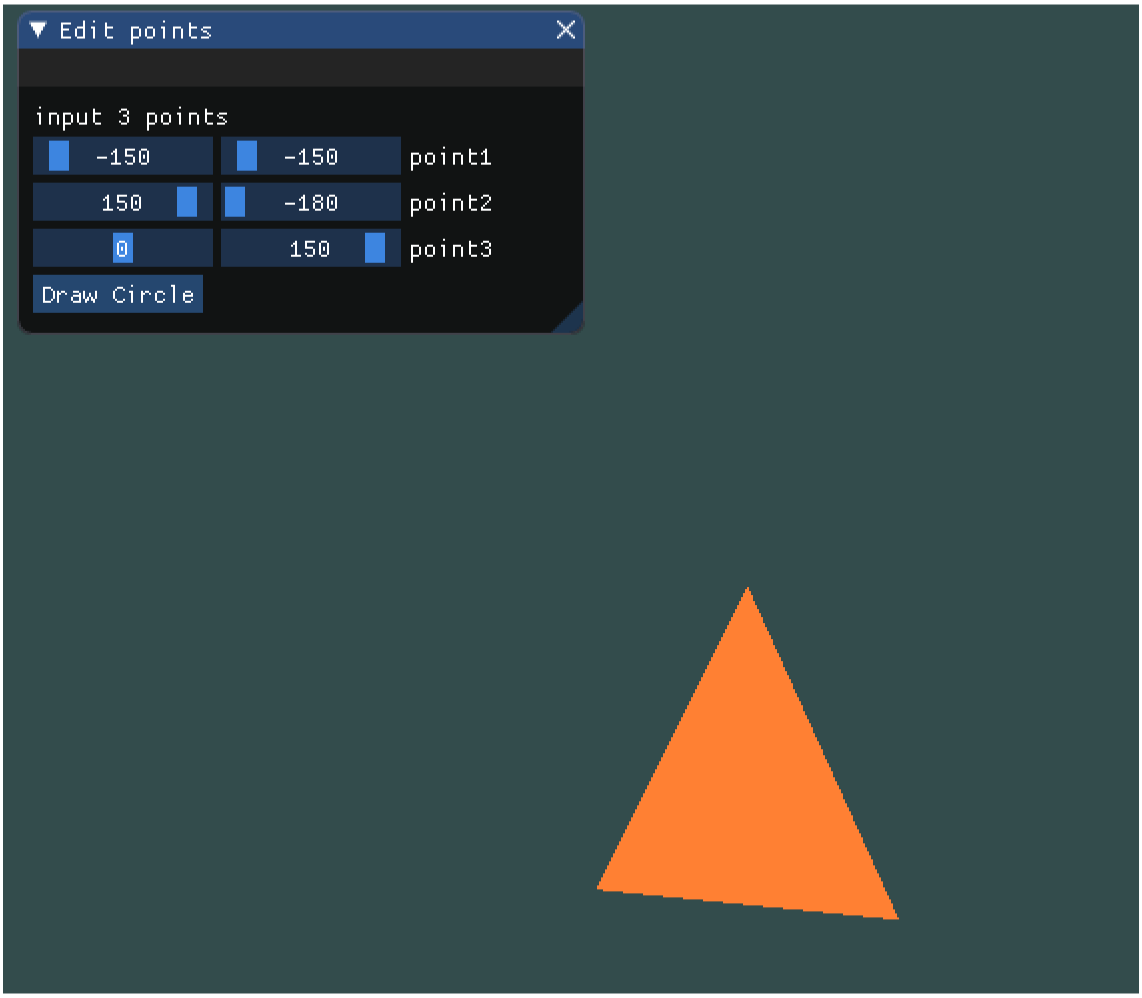
```
void edge_equations(vertices T[3])
{
  bbox b = bound(T);
  foreach pixel(x, y) in b {
    inside = true;
    foreach edge line Li of Tri {
      if (Li.A*x+Li.B*y+Li.C < 0) {
        inside = false;
      }
    }
    if (inside) {
      set_pixel(x, y);
    }
  }
}
```



算法解释

1. 根据三角形的三个点，计算三条边的一般式方程Ax+By+C=0，其中A=Y2-Y1，B=X1-X2，C=X2*Y1-X1*Y2。
2. 将三条边“中心化”，取三角形的中点，即((X1+X2+X3)/3, (Y1+Y2+Y3)/3)，分别代入三个直线方程，若最终结果小于0，则把该方程的三个参数A、B、C分别乘以-1。
3. 计算三角形外接矩阵，该矩阵的两条边分别与X轴Y轴平行。即在三角形的三个顶点中，找到最小的x值leftX和最大的x值rightX，那么外接矩阵的横边就从leftX开始到rightX结束，同理竖边从bottomY开始到topY结束。
4. 遍历外接矩阵的每一个点，每个点都代入三个直线方程，若结果都大于0，则该点在三角形内，画出该点。

结果



Bresenham算法画圆

原理

我是用八分法画圆，因为圆上的一点，可以有另外在圆上的7个对称点，我们只需要画八分之一的圆。如下图：

我们只需要画第一象限的上半部分的八分之一圆。画图时，计算出的点作对称变换就可得到其余7个对称点，最后就能获得一整个圆的所有点。用GL_POINTS即可画出圆。

算法

我们假设圆心在原点（若不在原点，只需要作简单平移变换即可），并且主要画第一象限的上半部分的八分之一圆，其余部分用对称方法解决。

1. 取初始值x=0, y=radius, endX=radius/(float)sqrt(2), d=1.25-radius
2. 如果d<=0，令(x+1, y)取代(x,y)，并且d=d+2*x+2；否则令(x+1, y-1)取代(x,y)，并且d=d+2*(x-y)+5。
3. 画出当前(x,y)点和其余7个对称点。
4. 当x<=endX，重复步骤2~3。

结果

画出的圆如下所示：