# Homework 3 - Draw line

## Basic:

**1.使用 Bresenham 算法(只使用 integer arithmetic)画一个三角形边框：input 为三个 2D 点；output 三条直线（要求图元只能用 GL_POINTS，不能使用其他，比如 GL_LINES 等）。**

环境配置与着色器的编写与上一次作业相同，在此不在赘述。

关于 Bresenham 算法画直线的原理,在本次报告中不再说明,如需要,请前往我的博客。

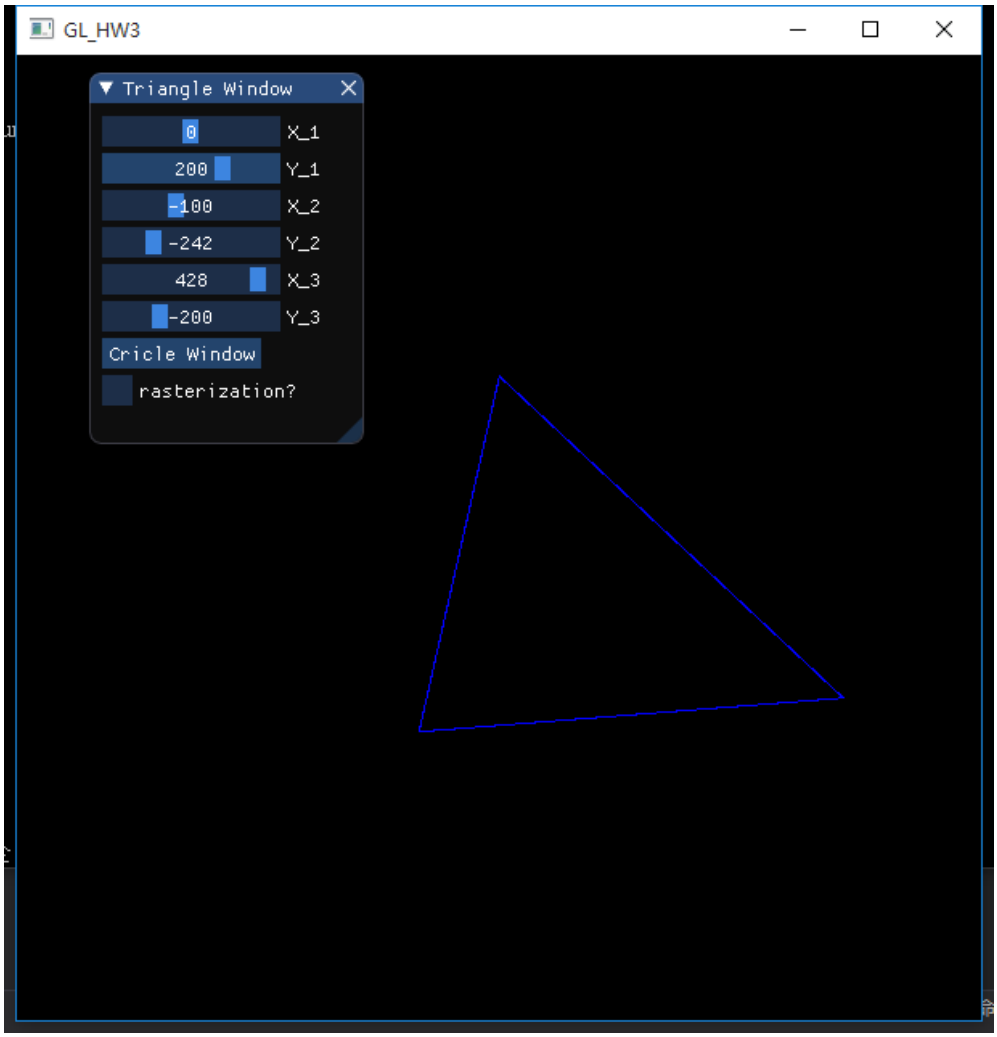
**Bresenham 算法**画直线的实现步骤如下：

1. 画出（$x_0$，$y_0$）
2. 计算$\triangle x$，$\triangle y$，$2\triangle y$，$2\triangle y - 2\triangle x$，$p_0 = 2\triangle y - \triangle x$
3. 如果 $p_i \leqslant 0$，画出（$x_{i+1}$，$y_{i+1}$）=（$x_i + 1$，$y_i$），同时计算 $p_{i+1} = p_i + 2\triangle y$
4. 如果 $p_i > 0$，画出（$x_{i+1}$，$y_{i+1}$）=（$x_i + 1$，$y_i + 1$），并计算 $p_i+1 = p_i + 2\triangle y - 2\triangle x$
5. 重复第 3 步和第 4 步，直到画完直线。

本次使用 Bresenham 算法画三角形的步骤如下：

- 确定三角形的三个顶点，然后对于每条边，使用 Bresenham 算法进行绘制。
- Bresenham 函数，使用 bresenham 算法画线段，但只适合绘制斜率范围为[0,1]的线段。
- drawLine 函数，画任意线段的处理。设 m 为线段斜率，主要考虑线段与 x 轴垂直，线段与 y 轴垂直（为加快计算，单独考虑），|m| <= 1，|m| > 1 四种情况。垂直 x 轴的线段，只需固定像素的 x 值，增加 y 值即可。对于垂直 y 轴的线段，则是固定 y 值，递增 x 值即可。当|m|<=1 时，x 轴递增，计算 y 的值，此时分为两种情况讨论：0 <= m <= 1 时，可直接使用 bresenham 函数处理，线段方向为左下至右上；-1 <= m < 0 时，此时线段方向为从左上至右下，为了能使用 bresenham 函数，不妨先计算线段关于 x 轴的对称线段，即将 y 方向上所有的处理取反。得到对称线段的像素点坐标后，将对称线段的所有像素点在 y 轴上取反，则得到原线段的像素点坐标。当|m|>1 时，y 轴递增，计算 x 的值。实际为将|m| <= 1 情况中 x 与 y 的身份调换即可。
- createVertices 函数，将原本整数的坐标值（x,y）转换为 OpenGL 能渲染的坐标值。包括属性扩展以及坐标值归为（-1,1）
- 对于三角形三个顶点的输入，为简便对用户输入判断的处理，采用 ImGui 的 SliderInt 函数来控制输入，使得可以改变三角形的形状。

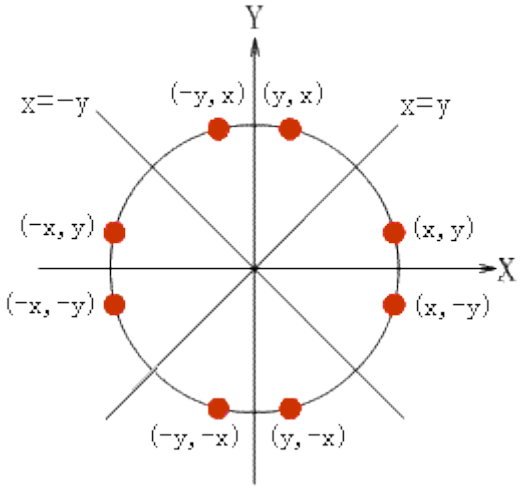
实现效果如下：




2. 使用 Bresenham 算法(只使用 integer arithmetic)画一个圆：input 为一个 2D 点(圆心)、一个 integer 半径；output 为一个圆。

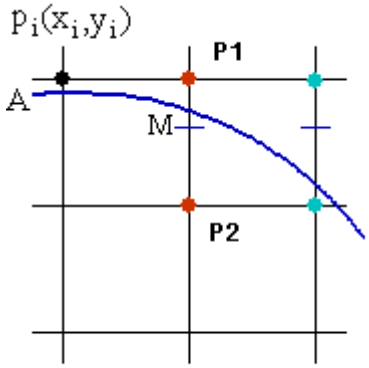
**实现原理：**

首先需明白，圆具有八分对称性，如下图：



因此，只需画出八分之一的圆弧，就可以利用对称性得到整个圆。

设圆心为$(x_0, y_0)$，半径为 R 的圆，其从（$x_0, y_0 + R$）顺时针旋转八分之一圆弧。假定某点 $P_i(x_i, y_i)$是前一个已确定的像素点，那么 Pi 的下一个点只可能是正右方的 P1 或右下方的 P2 两者之一，如下图：




则有如下判别式：

$$d = (x - x_0)^2 + (y - y_0)^2 - R^2$$

当 d = 0 时，表示点在圆上，当 d > 0 时，表示点在圆外，当 d < 0 时，表示点在园内。

经过处理（过程参考博客），可以得出如下递推式：

$$d_0 = 3 - 2R$$

当 $d_i$ < 0 时：

$$d_{i+1} = d_i + 2x_i + 3$$

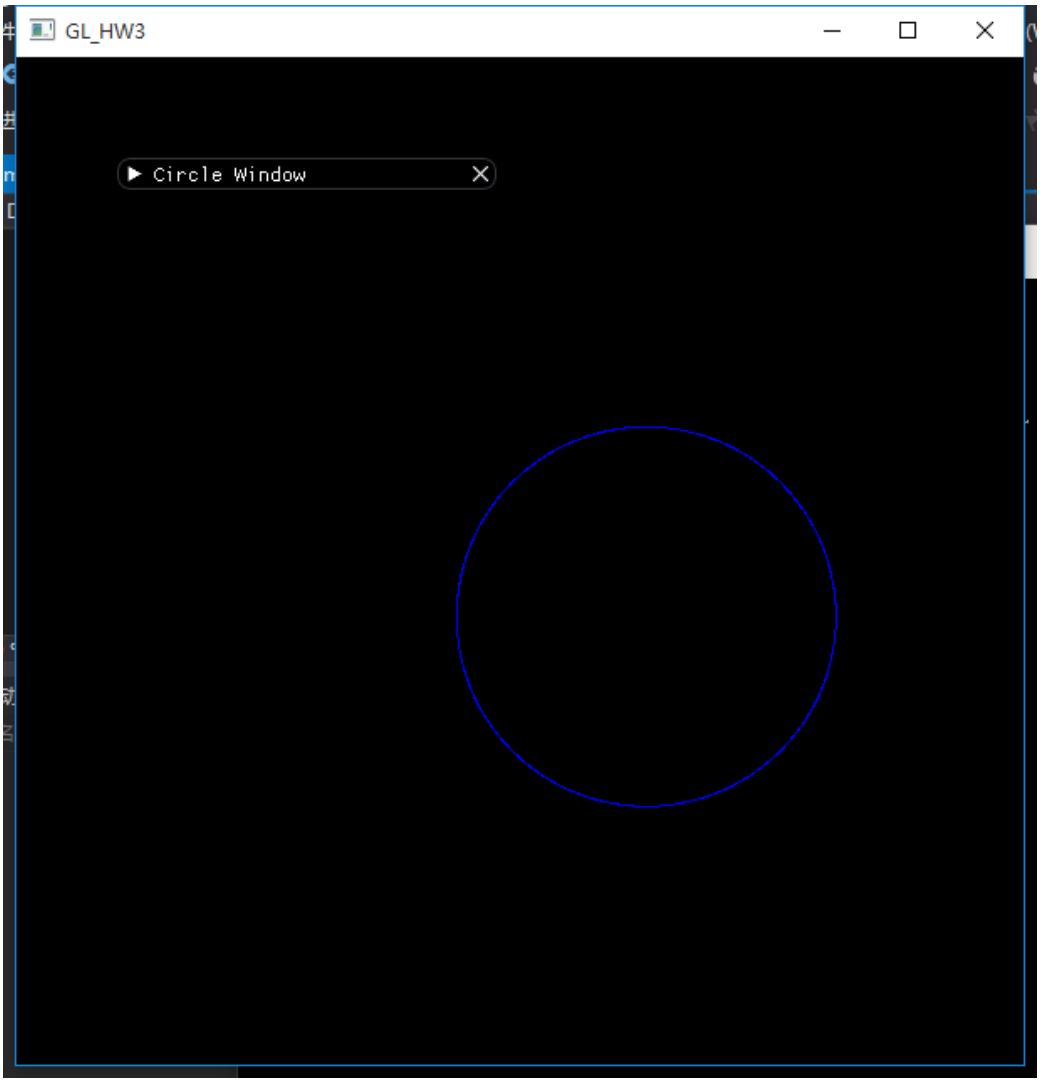
当 $d_i$ > 0 时：

$$d_{i+1} = d_i + 2(x_i - y_i) + 5$$

**实现步骤如下：**

- drawCircle 函数，根据圆心坐标和半径，按照实现原理中所述，求出八分之一圆弧。
- getRejectionPoints 函数，根据八分之一对称性质，补充圆上其他点。

实现效果如下：



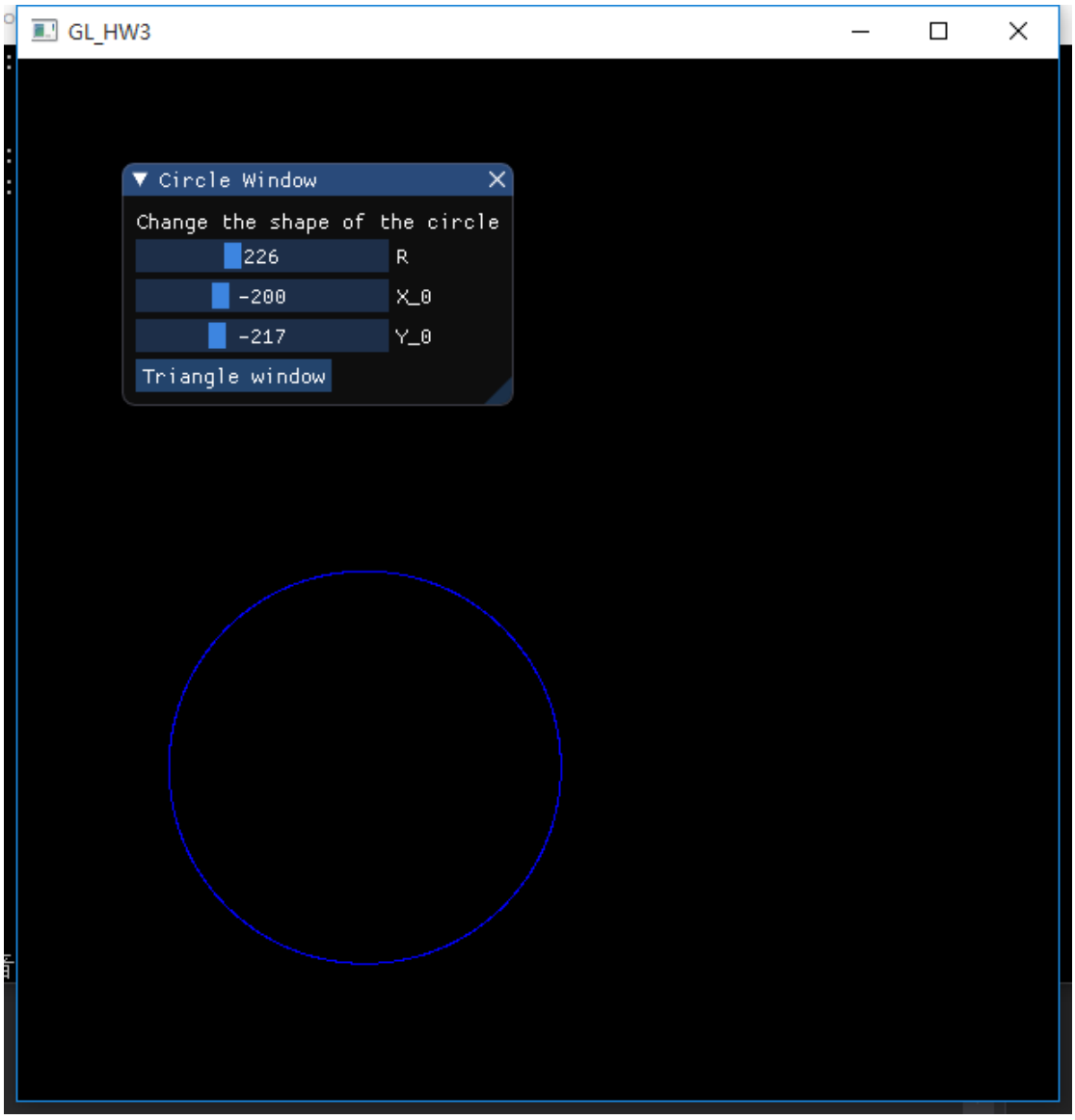

3. 在 GUI 在添加菜单栏，可以选择是三角形边框还是圆，以及能调整圆的大小(圆心固定即可)。

实现步骤：

添家 ImGui 组件，使用 SliderInt 函数控制圆的半径以及圆心坐标。

```
//画圆
if (drawing_triangle == false) {
    ImGui::Begin("Circle Window", &drawing_triangle);
    ImGui::Text("Change the shape of the circle");
    ImGui::SliderInt("R", &R, 5, 600);
    ImGui::SliderInt("X_0", &x0, -550, 550);
    ImGui::SliderInt("Y_0", &y0, -550, 550);
    if (ImGui::Button("Triangle window")) {
        drawing_triangle = true;
    }
    ImGui::End();
    drawCircle(x0, y0, R, graphic);
}
```

**实现效果：**




## 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。**

采用 Edge Equations 算法进行三角形的填充。

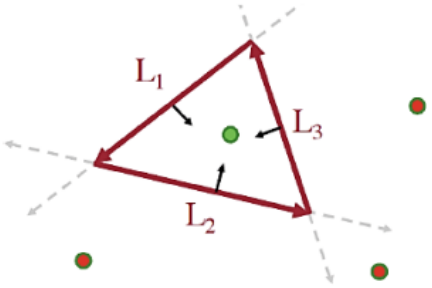
**算法步骤：**

- 计算三角形三条边的一般表达式（Ax + By + C = 0）；
- 计算三角形的外接矩阵。
- 将三条边“中心化”，即使得三角形中任意一点带入 3 条表达式，均有 Ax + By + C 的结果都大于 0。具体为将决定线段的两个点外的第三个点（这里采用三角形的另外一个顶点）带入到直线方程，如结果小于 0，则将参数 A、B、C 都乘以-1。
- 遍历外接矩阵中的每个点，用 3 条边的方程判断该点在不在三角形中，若在，则绘制该点。

伪代码如下：

## Edge Equations

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




**具体实现步骤：**

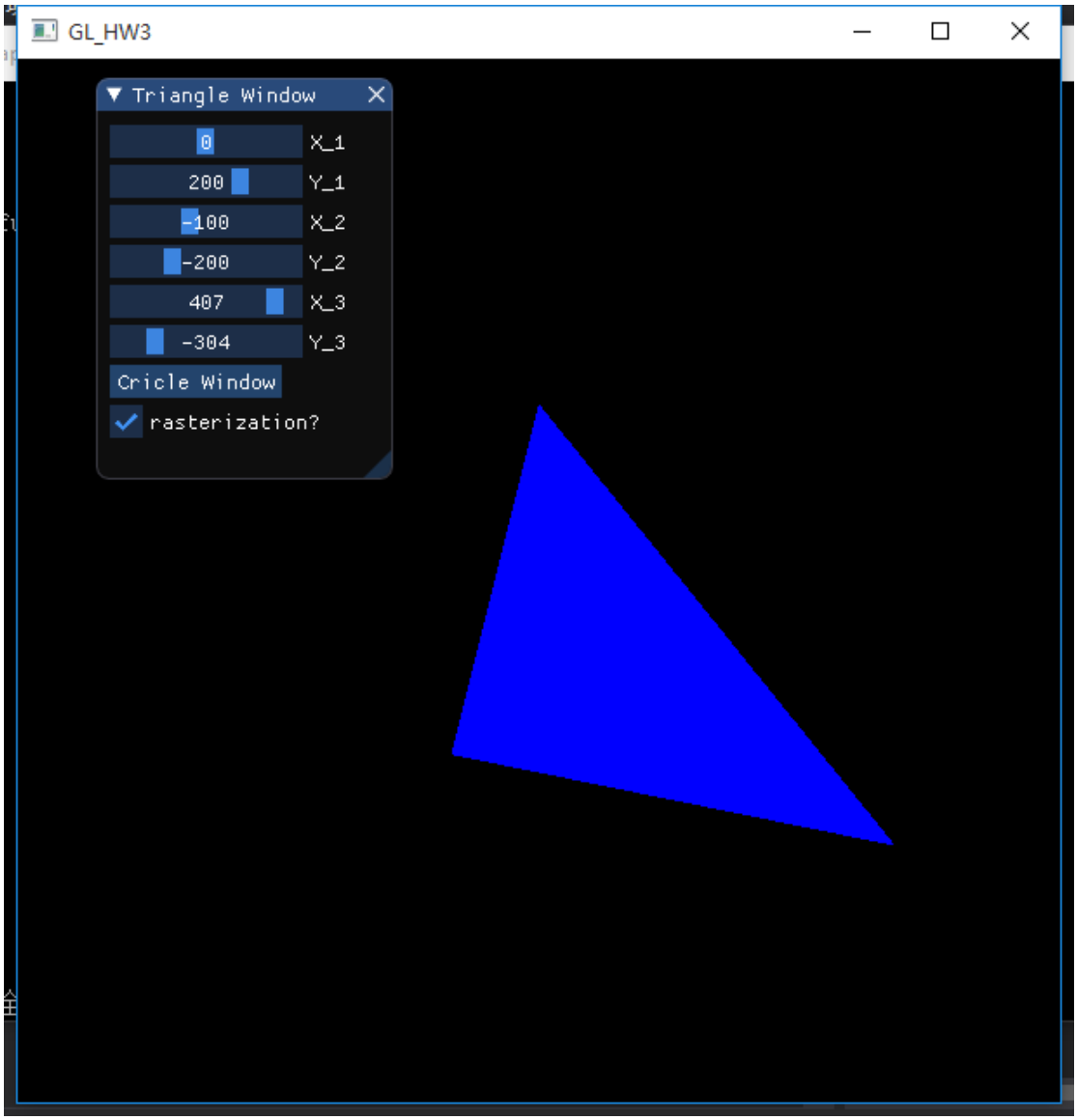
- rasterizeTriangle 函数，根据输入的三角形三个顶点坐标，首先，找出三角形的外接矩阵；然后，调用 lineExpression 函数，求出三条边的一般表达式；接下来，将三条边“中心化”；最后，遍历外接矩阵中的每个点，用 3 条边的方程判断该点在不在三角形中。
- lineExpression 函数求出每条边的一般表达式。实现如下：

```
/*
* 根据两个端点，求出线段的一般表达式（Ax + By + C = 0）
* A = y2 - y1;  B = x1 - x2;  C = x2 * y1 - x1 * y2
* x1,y1,x2,y2:两个端点的坐标
*/
vector<int> lineExpression(int x1, int y1, int x2, int y2) {
    //line expression:Ax + By + C = 0
    vector<int> res;
    res.push_back(y2 - y1);
    res.push_back(x1 - x2);
    res.push_back(x2 * y1 - x1 * y2);

    return res;
}
```

- 在 drawTriangle 函数中增加一个 bool 参数，如果该参数值为真，则在 drawTriangle 函数中进行三角形填充，否则，不填充。
- 设置 GUI，通过 Checkbox 函数来控制上述 bool 参数的真假，从而判定是否进行填充。
- 在 GUI 中增加一个 button，用于在画三角形和画圆之间进行切换。

**实现效果如下：**




**说明：**

演示视频见 doc 文件夹中的演示视频.mp4