《计算机图形学》实验 3 实验报告

实验人	刘沅昊	学号	15331220
学院	数据科学与计算机学院	年级专业	16 级软件工程(数字媒体技术)

实验内容：

Homework

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）。
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。
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)。

Bonus:

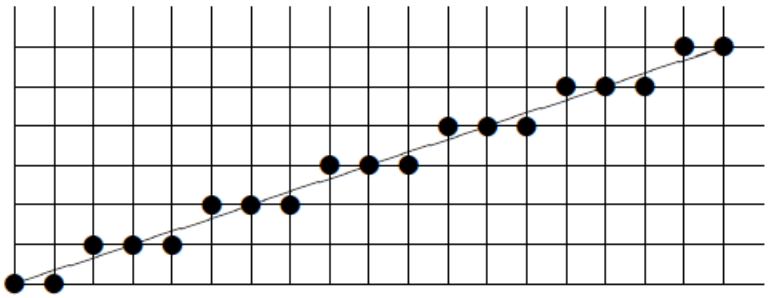
1. 使用三角形光栅转换算法，用和背景不同的颜色，填充你的三角形。

实验步骤：

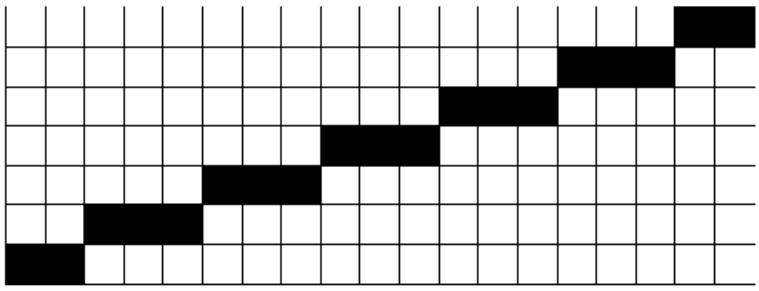
Basic：

1. Bresenham 直线算法：

   直线事实上是连续的，但是计算机精度有限，而且显示设备都是一个个的像素组成，所以显示的的时候不能真实显示连续的直线，都是用的离散化的像素点来近似表示一条直线。



(a).实际要求的直线及其近似点

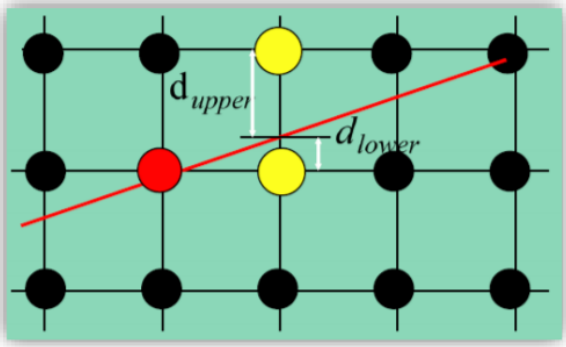


(b).离散化后用像素点表示的的直线

图表 1 《计算机图形学的概念与方法》柳朝阳

Bresenham 算法的思想就是从一个点出发到下一个点，哪个点离真实值近就选那个点。比如说下面已经确定了红色点是要显示的一个点，黄色点是下一个要选择的点，则选择上面黄色点还是下面黄色点的判别标准就用到了左式。

$$
\begin{aligned}
d_{upper} &= \bar{y}_i + 1 - y_{i+1} \\
&= \bar{y}_i + 1 - mx_{i+1} - B \\
d_{lower} &= y_{i+1} - \bar{y}_i \\
&= mx_{i+1} + B - \bar{y}_i
\end{aligned}
$$



图表 2 判别标准

但是计算机进行除法运算效率比较低，所以要对上式进行改进。

$$d_{lower} - d_{upper} = m(x_i + 1) + B - \bar{y}_i - (\bar{y}_i + 1 - m(x_i + 1) - B)$$
$$= 2m(x_i + 1) - 2\bar{y}_i + 2B - 1$$

division operation

- **It has the same sign with**

$$p_i = \Delta x \bullet (d_{lower} - d_{upper}) = 2\Delta y \bullet (x_i + 1) - 2\Delta x \bullet \bar{y}_i + (2B - 1)\Delta x$$
$$= 2\Delta y \bullet x_i - 2\Delta x \bullet \bar{y}_i + (2B - 1)\Delta x + 2\Delta y$$
$$= 2\Delta y \bullet x_i - 2\Delta x \bullet \bar{y}_i + c$$

where

$$\Delta x = x_1 - x_0, \Delta y = y_1 - y_0, \quad m = \Delta y / \Delta x$$
$$c = (2B - 1)\Delta x + 2\Delta y$$

图表 3 变换过程

- If $p_i \succ 0$, then $(\bar{x}_i + 1, \bar{y}_i + 1)$ is selected

  If $p_i \prec 0$, then $(\bar{x}_i + 1, \bar{y}_i)$ is selected

  If $p_i = 0$, *arbitrary one*

- **Can we simplify the computation of p_i ?**

$$p_0 = 2\Delta y \bullet x_0 - 2\Delta x \bullet \bar{y}_0 + (2B - 1)\Delta x + 2\Delta y$$
$$= 2\Delta y \bullet x_0 - 2(\Delta y \bullet x_0 + B \bullet \Delta x) + (2B - 1)\Delta x + 2\Delta y$$
$$= 2\Delta y - \Delta x$$

$$y_{i+1} = mx_{i+1} + B$$

图表 4 变换过程

- As

$$p_{i+1} - p_i = (2\Delta y \bullet x_{i+1} - 2\Delta x \bullet \bar{y}_{i+1} + c) - (2\Delta y \bullet x_i - 2\Delta x \bullet \bar{y}_i + c)$$
$$= 2\Delta y - 2\Delta x(\bar{y}_{i+1} - \bar{y}_i)$$

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

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

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

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

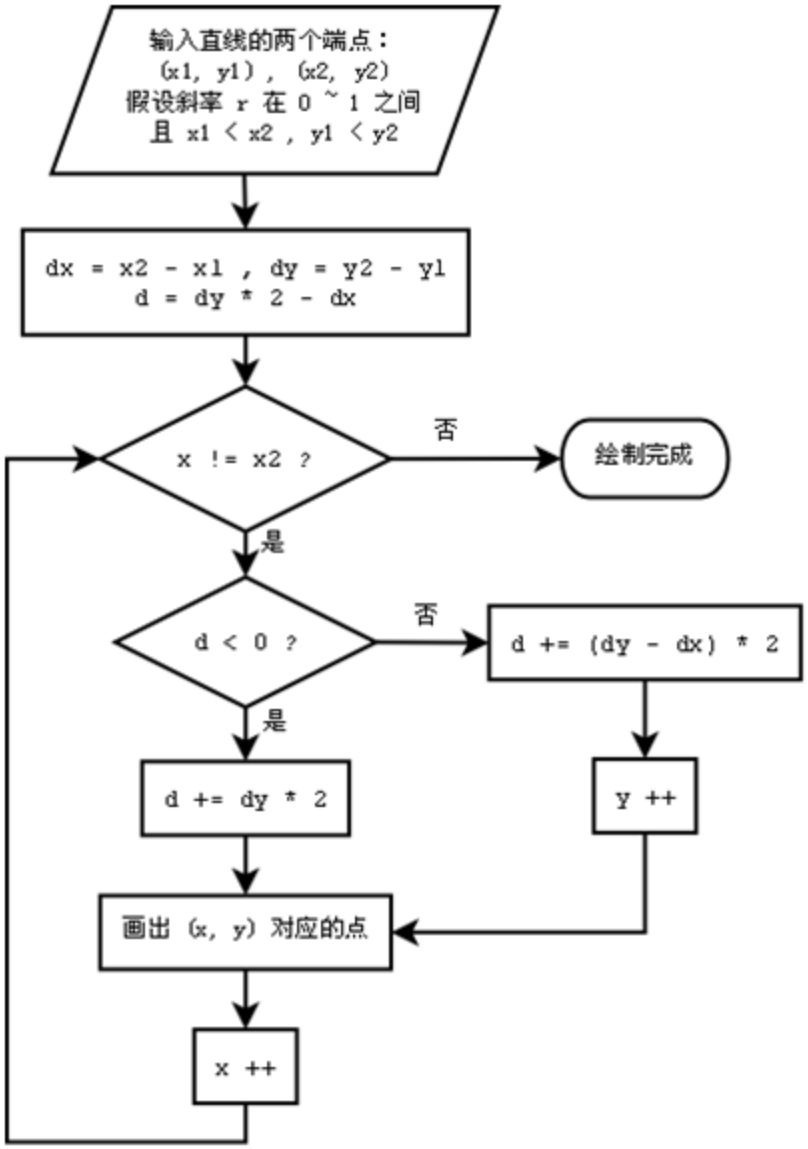
图表 5 变换过程

- **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** $p_i \le 0$ **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**

图表 6 Bresenham 画线算法流程



图表 7 Bresenham **画线算法流程图**[1]

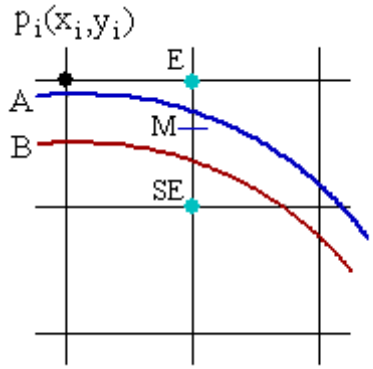
可以看到，算法其实只考虑了斜率在 0～1 之间的直线，也就是与 x 轴夹角在 0 度到 45 度的直线。只要解决了这类直线的画法，其它角度的直线的绘制全部可以通过简单的坐标变换来实现。

2. Bresenham 画圆算法：

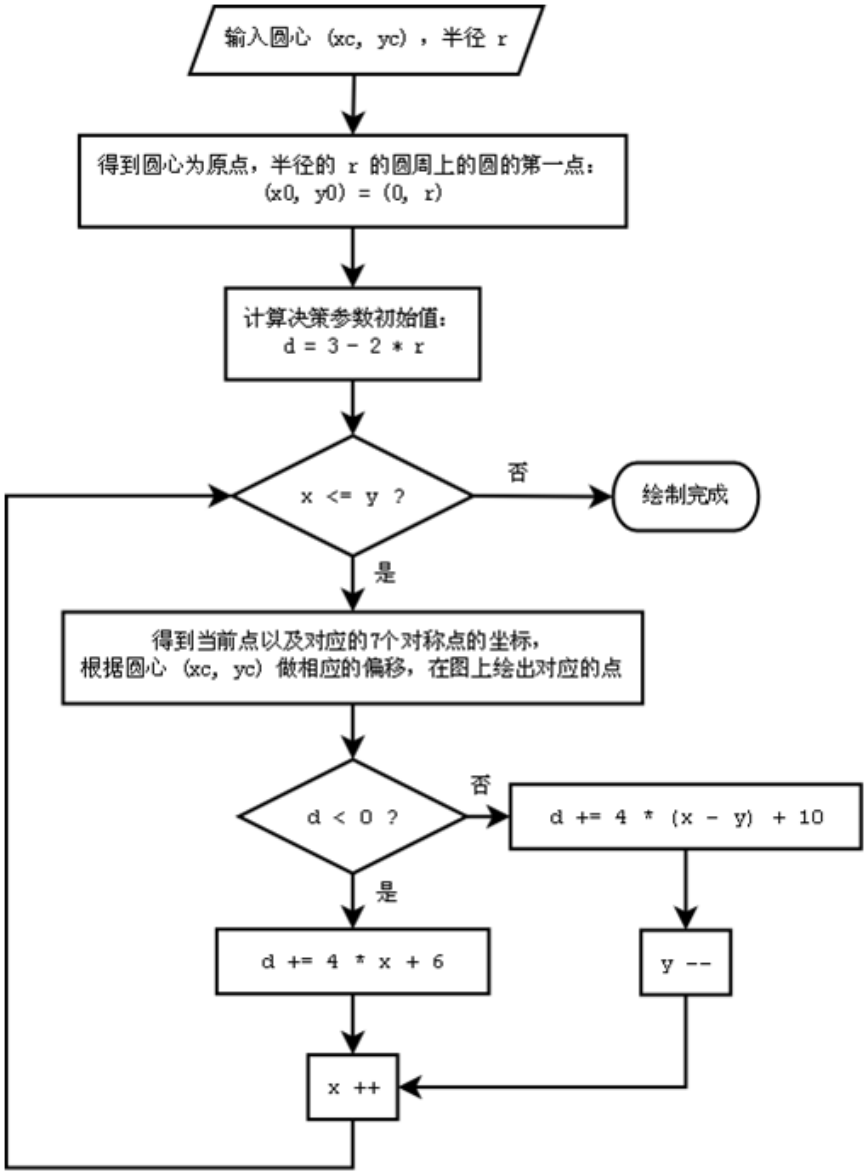
Bresenham 画圆算法又称中点画圆算法，与 Bresenham 直线算法一样，其基本的方法是利用判别变量来判断选择最近的像素点，判别变量的数值仅仅用一些加、减和移位运算就可以计算出来。为了简便起见，考虑一个圆 心在坐标原点的圆，而且只计算八分圆周上的点，其余圆周上的点利用对称性就可得到。

对于画圆算法，主要思路就是取可选点之间的中点，如下图的 M:

[1] https://www.cnblogs.com/wlzy/p/8695226.html

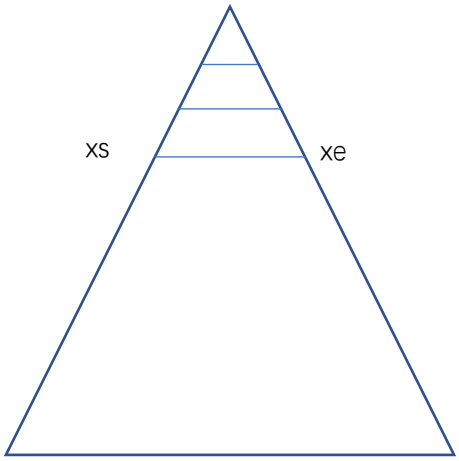


然后判断这个点是在圆内还是在圆外，据此判断所选取的点是 E 还是 SE。然后通过迭代画出 1/8 个圆，根据八分法，就可以画出整个圆了。



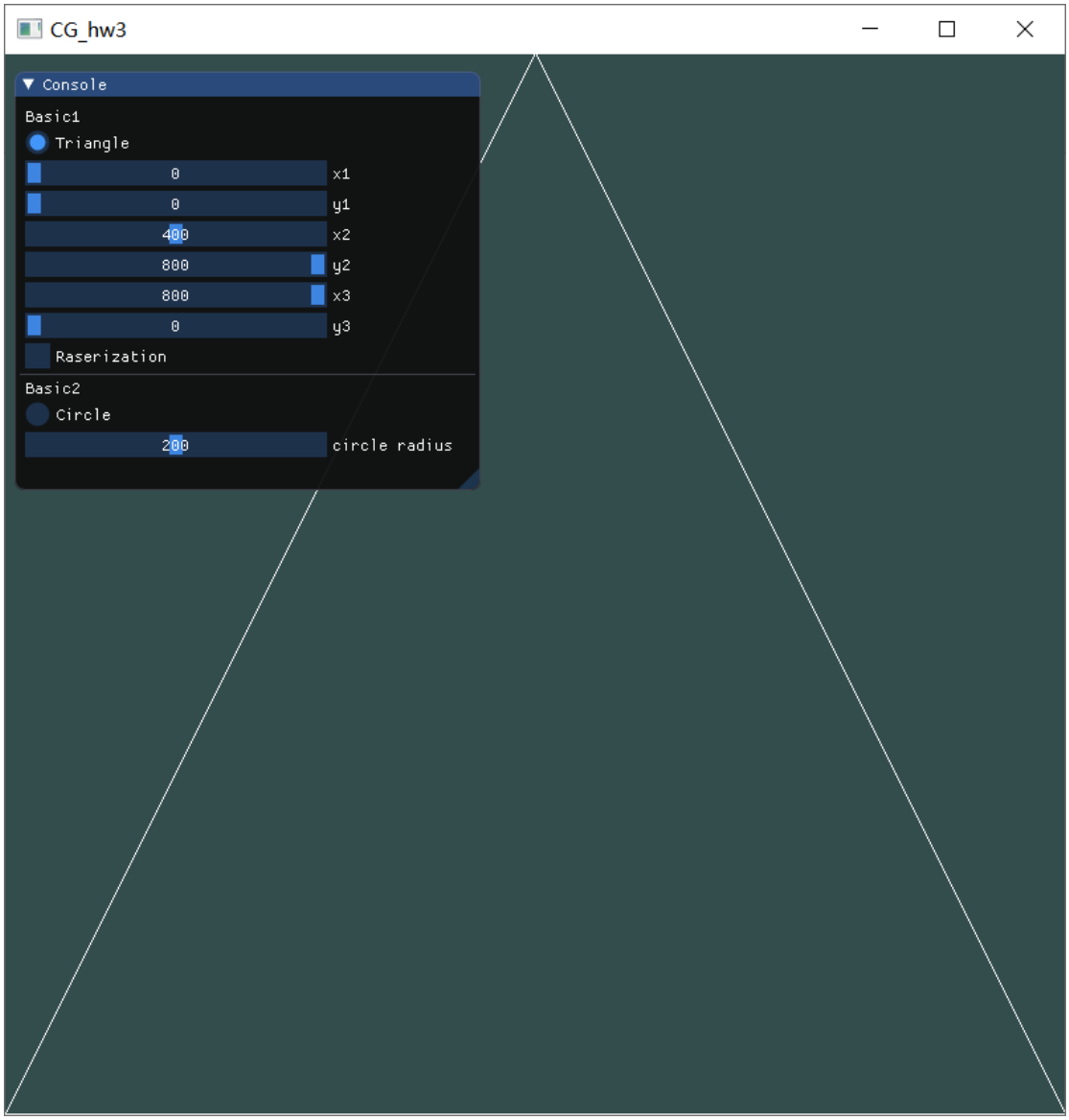
3. 扫描线光栅化算法：

前面实现了画线算法，三角形光栅化就是一行行的画线段填充三角形。

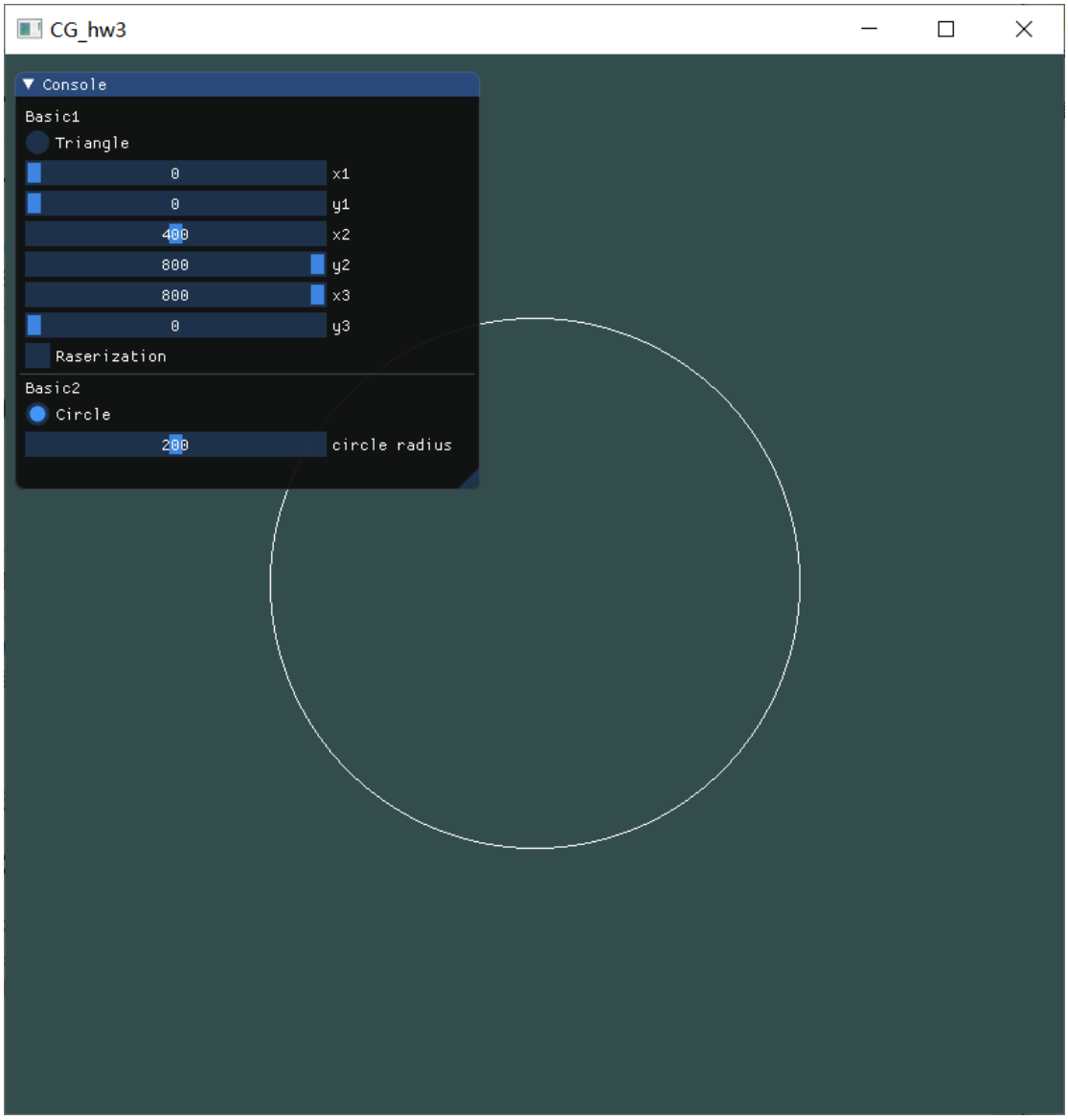


我们需要做的就是求得一条水平直线与三角形两条边的交点，然后在这两个交点(xs,y),(xe,y)之间画线段。这样从上往下扫描就完成了三角形光栅化。

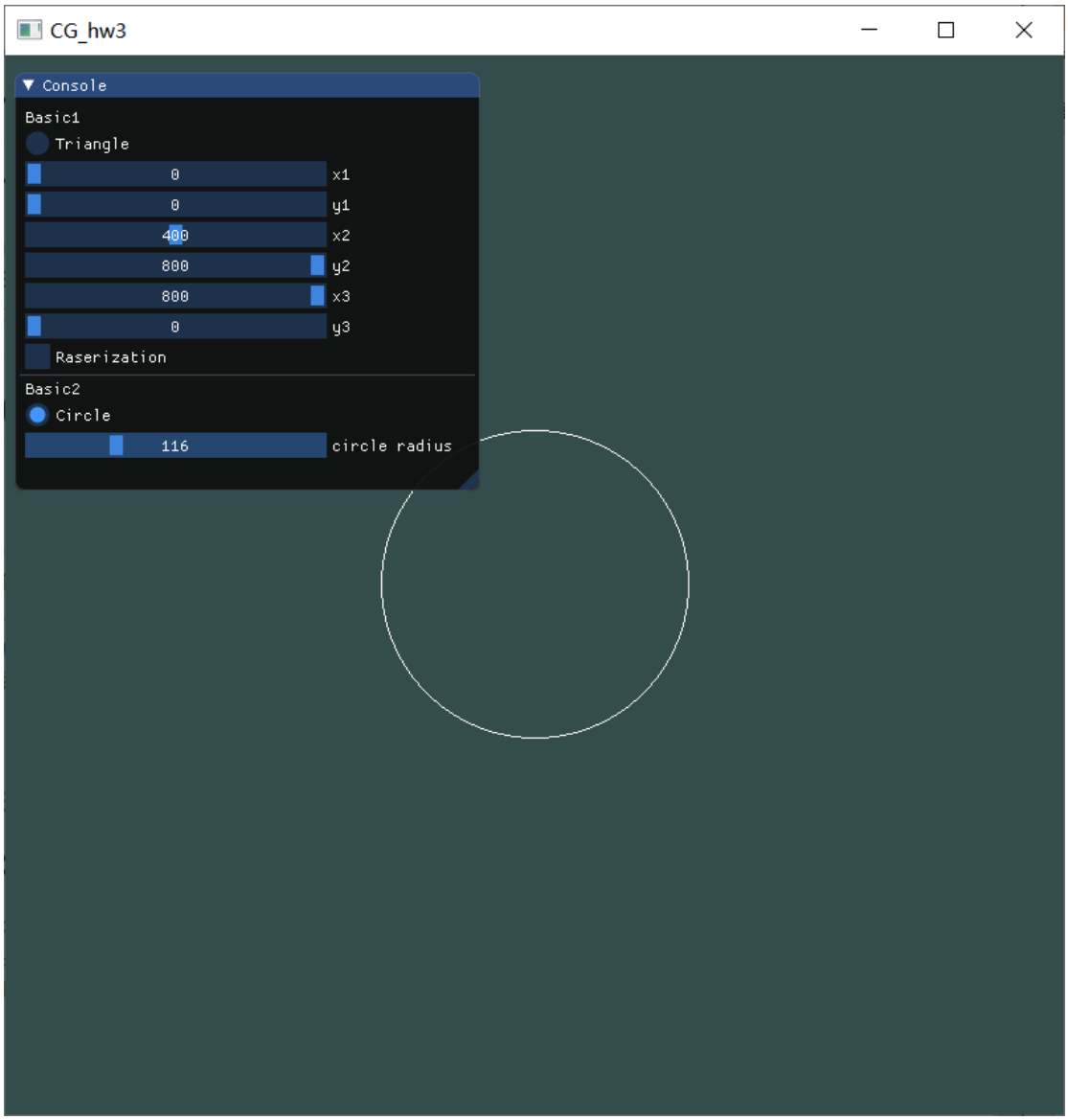
实验结果：



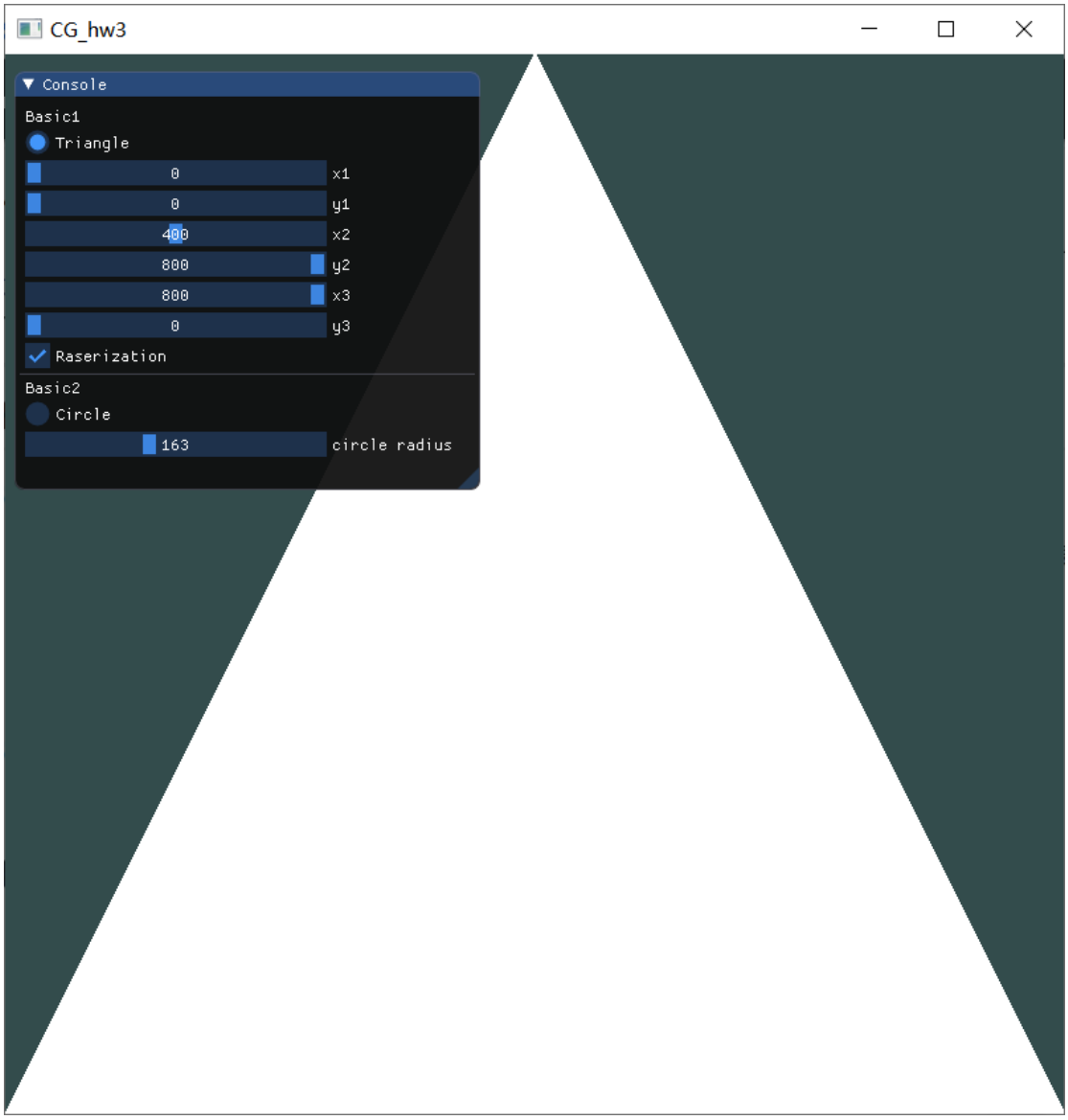
图表 8 初始界面


CG_hw3
Console
Basic1
Triangle
0
x1
0
y1
400
x2
800
y2
800
x3
0
y3
Raserization
Basic2
Circle
200
circle radius

图表 9 选择画圆按钮



图表 10 通过控件调整圆的半径



图表 11 选中光栅化选项，三角形被填充

```
126         if (choose == 0) {
127             if (if_raserization) {
128                 DrawTriangle(triangle_point[0], triangle_point[1], triangle_point[2],
129                     triangle_point[3], triangle_point[4], triangle_point[5]);
130             } else {
131                 Bresenham_line(triangle_point[0], triangle_point[1], triangle_point[2], triangle_point[3]);
132                 Bresenham_line(triangle_point[0], triangle_point[1], triangle_point[4], triangle_point[5]);
133                 Bresenham_line(triangle_point[2], triangle_point[3], triangle_point[4], triangle_point[5]);
134             }
135         }
136         if (choose == 1) {
137             Bresenham_circle(now_screen_width / 2, now_screen_height / 2, circle_radius);
138         }
139 
140         glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
141         glClear(GL_COLOR_BUFFER_BIT);
142 
143         glBindVertexArray(VAO);
144 
145         // 渲染
146         glDrawArrays(GL_POINTS, 0, points_to_draw_index / 3);
147         ImGui::Render();
148         ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
149         glfwSwapBuffers(window);
150     }
```

图表 12 没有使用 GL_LINES 等绘制