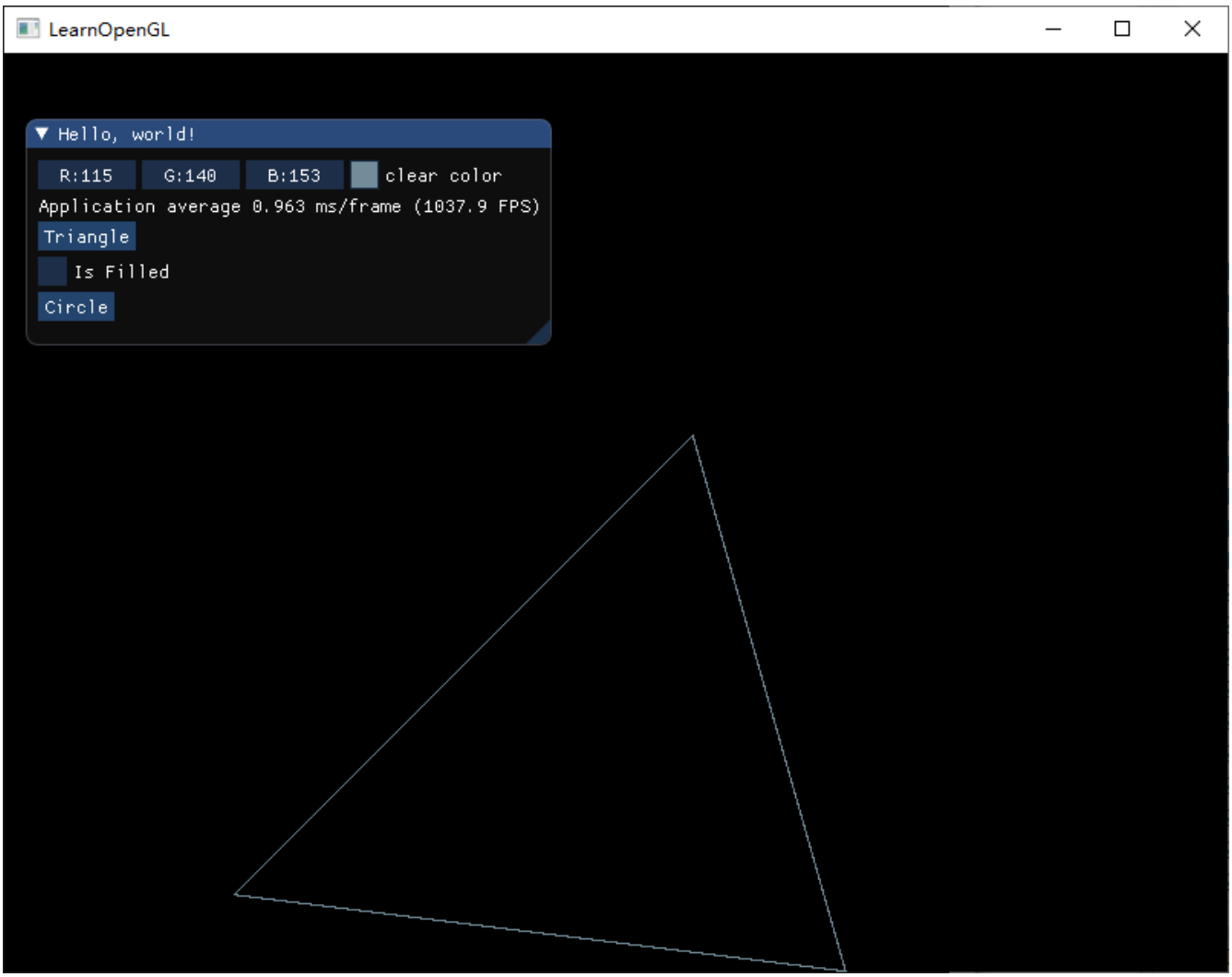
Homework 3

16340282 袁之浩

Basic

1. 绘制三角形边框

实验截图

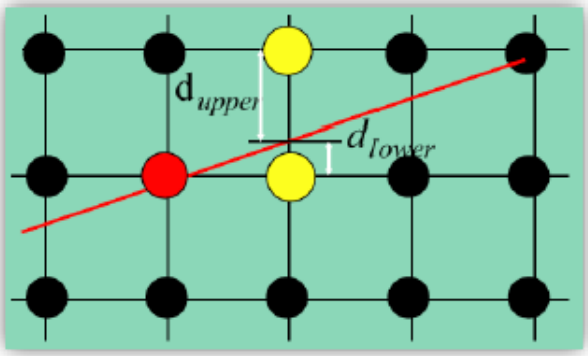


算法实现

首先考虑最基本的情况，假设输入直线的两个端点为p1(x1, y1) 和p2 (x2, y2),直线的斜率在0-1之间，且x1<x2，y1<y2。

- The distances are respectively

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



显然：如果 $d_{lower} - d_{upper} > 0$ 则应取右上方的点；如果 $d_{lower} - d_{upper} < 0$ 则应取右边的点； $d_{lower} - d_{upper} = 0$ 可任取，如取右边点。

- **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**

现在考虑其他情况，如果x1>x2，则交换p1和p2的坐标，如果y1>y2，则计算与直线关于x轴对称的直线坐标，如果斜率大于1，则交换x和y的坐标。在计算出每一个点的位置后，再通过逆变换得出原直线坐标点的位置。

```
bool isFlipXY = abs(p1.x - p2.x) < abs(p1.y - p2.y);
//扩展3,反转xy轴
if (isFlipXY) {
    flipXY(p1);
    flipXY(p2);
}

//扩展1,保证x是递增
```

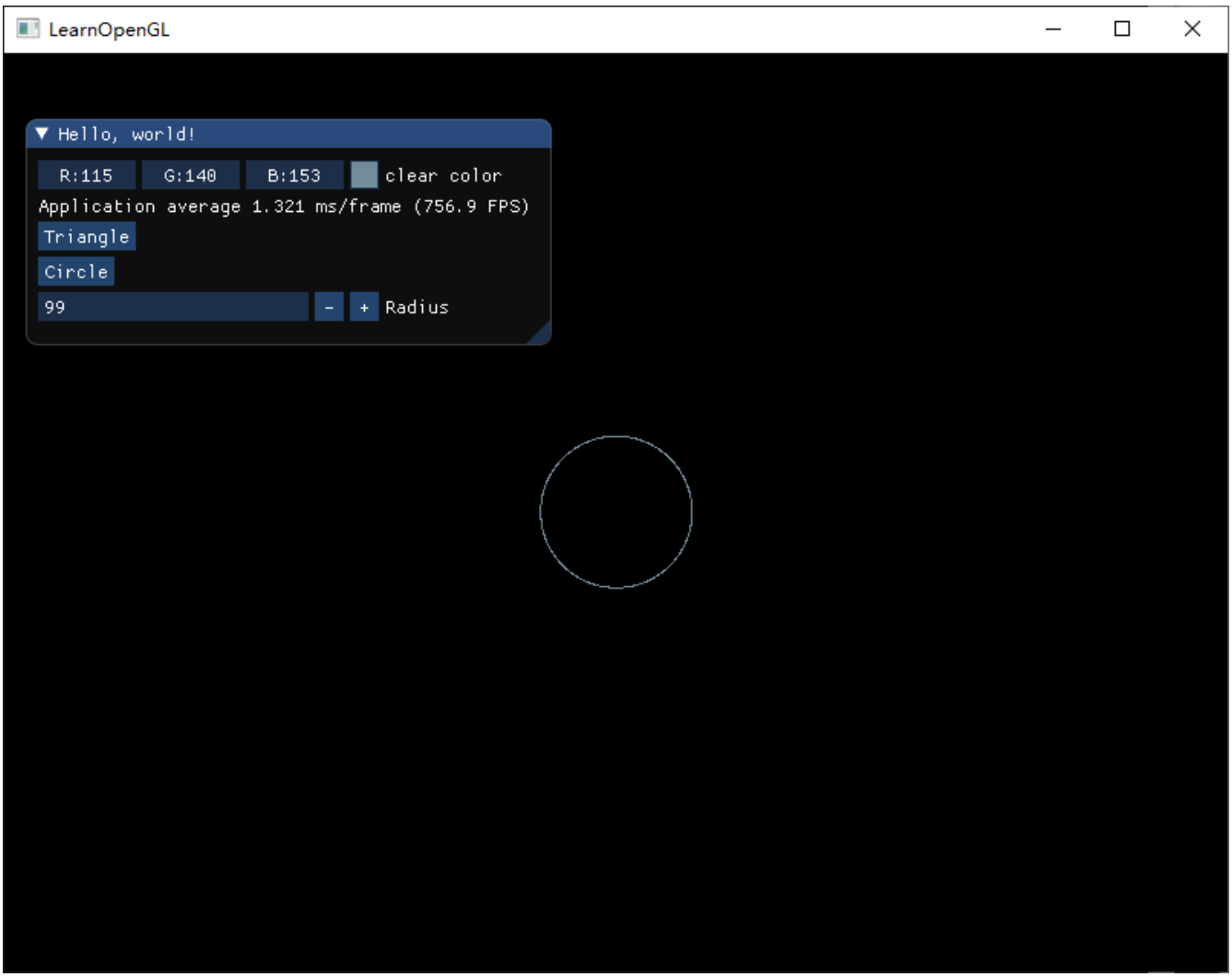
```
if (p1.x > p2.x) {
    swap(p1, p2);
}

//扩展2,保证y是递增
bool isFlipY = p1.y > p2.y;
if (isFlipXY) {
    flipY(p1, p2);
}
```

将点的像素坐标计算完成后，再使用 getGlPoint() 转换为适合openGL的坐标。

函数 getTriData() 分别绘制三条直线，再统一显示出来。

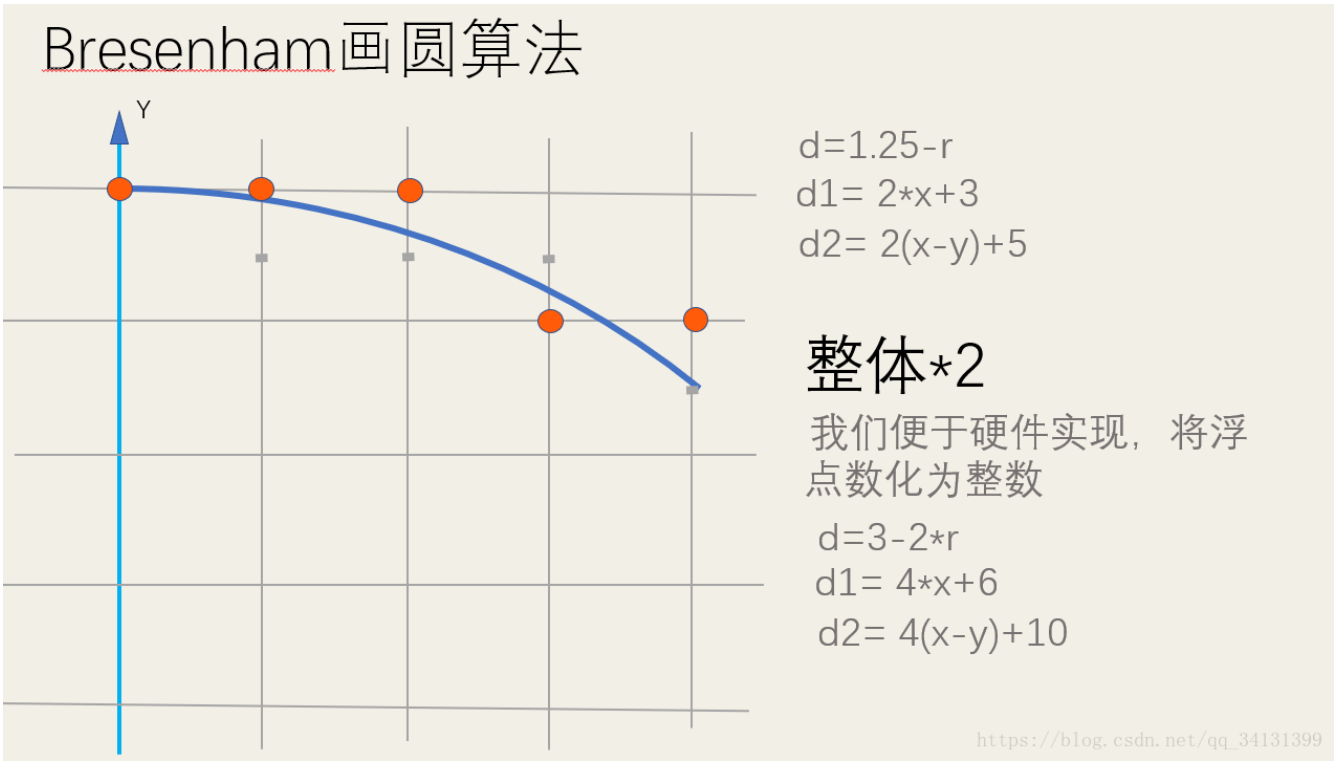
2. 绘制圆形边框

实验截图



算法实现

圆具有八对称的特性，所以只需要画出八分之一个圆就好了。我们通过中点到圆的距离（通过符号判定），我们就能选择相应的点。我们每次的选择中点时，增量都是有规律的。我们通过对直线算法的借鉴，我们发现我们能从圆的方程推导出每次坐标变换的增量。



函数 addCirclePlot() 用来添加其他7个对称点。

3. 添加GUI，可以改变圆的半径

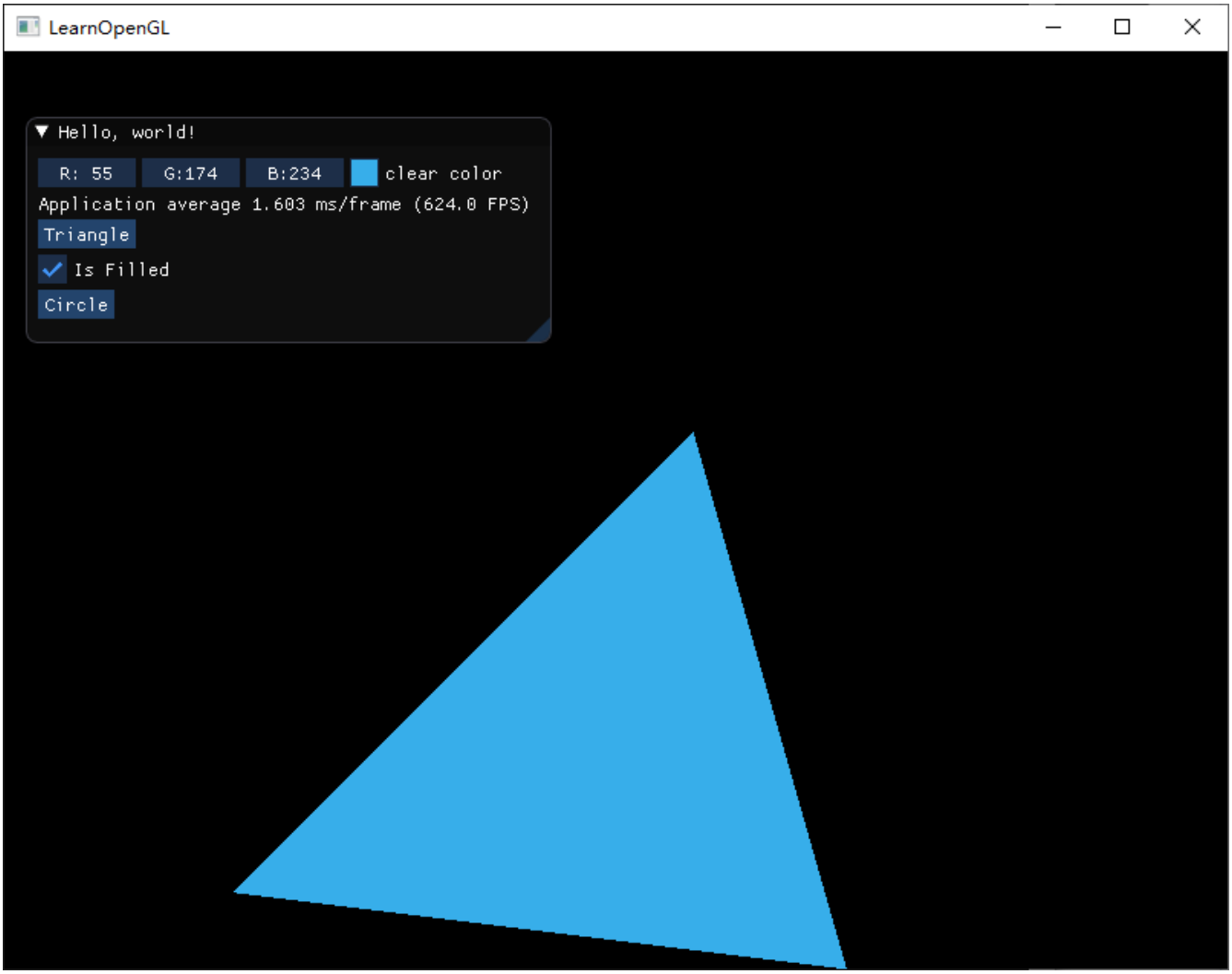
添加一个 ImGui::InputInt 用于设置圆的半径，然后当控件的值发生改变时，就重新计算并渲染。

```
        if (curr_radius != radius) {
            circleData.clear();
            circleData = getCircleData(origin, curr_radius);
            radius = curr_radius;
            pointData2vao(VAO[2], VBO[2], getGlPoint(circleData, SCR_WIDTH, SCR_HEIGHT));
            glDrawArrays(GL_POINTS, 0, circleData.size());
        }
        else {
            glBindVertexArray(VAO[2]);
            glDrawArrays(GL_POINTS, 0, circleData.size());
        }
```

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。

实验截图



算法实现

1. 通过两点计算一条直线的一般式Ax+By+C=0
2. 对于三角形的每一条边，在三角形内的点都与另外一个顶点在边的同侧，表现为带入一般式同号
3. 计算包围三角形的最小矩形
4. 对于矩形内的每一个点，带入三条直线的一般式进行检验，如果都满足2的要求的话，将该点的坐标加入res vector中

代码实现在函数 getFilledTri() 中。