Homework 3 - Draw line

黄树凯

16340085

作业要求

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)。
4. 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。（**Bonus**）

实现及结果

1. 绘制三角形边框

Notations

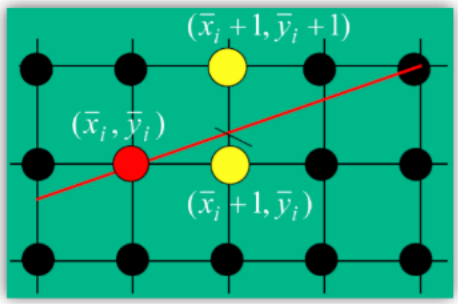
- The line segment is from (x_0,y_0) to (x_1,y_1)
- Denote $\Delta x = x_1 - x_0 > 0, \Delta y = y_1 - y_0 > 0 \quad m = \Delta y / \Delta x$
- Assume that slope $|m| \leq 1$
- Like DDA algorithm, Bresenham Algorithm also starts from $x = x_0$ and increases x coordinate by 1 each time
- Suppose the i-th point is (x_i, y_i)
- Then the next point can only be one of the following two $(\bar{x}_i + 1, \bar{y}_i)$ $(\bar{x}_i + 1, \bar{y}_i + 1)$

设线段的两点为 (x_0, y_0) 和 (x_1, y_1)，其种 $x_1 - x_0 > 0$，$y_1 - y_1 > 0$，并且斜率 m 大于 0 小于 1。

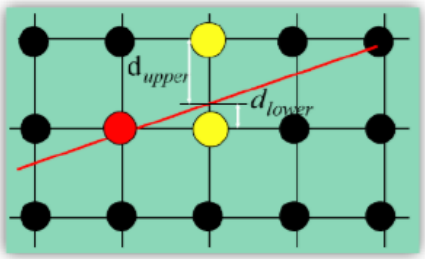
当前点为(x_i, y_i)，那么下一点可以从 $(x_i + 1, y_i)$ 和 $(x_i + 1, y_i + 1)$ 这两点之间选择，根据线段在 $x_i + 1$的位置上的 y 值，选择其距离较小的一点作为下一点。

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

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



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



Bresenham 算法过程

- **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**

实现过程

- 对给出的三组顶点坐标，两两之间分别用 Bresenham算法求出线段上的点
- 求出 dx，dy

```
int dx = end_x - start_x;
int dy = end_y - start_y;
```

- 根据斜率确定x，y从起点到终点每次是递增或者递减

```
int step_x = dx > 0 ? 1 : -1;
int step_y = dy > 0 ? 1 : -1;
```

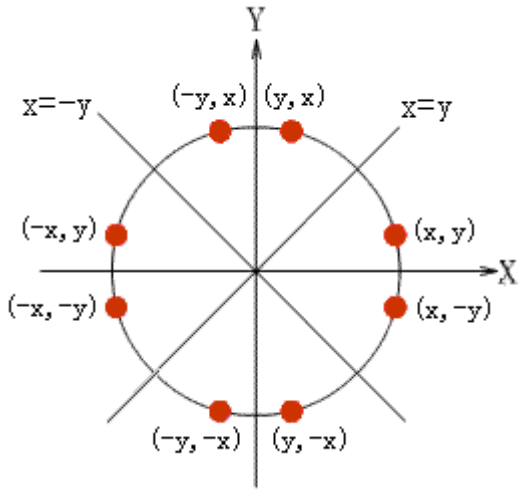
- 根据斜率是否大于1分情况讨论

- 当斜率小于 1 时，以 x 为基准按上述公式计算 p_i，以及 y_i
- 当斜率不下于 1 时，以 y 为基准对上述公式中的 x 和 y 进行替换来计算 p_i 以及 x_i

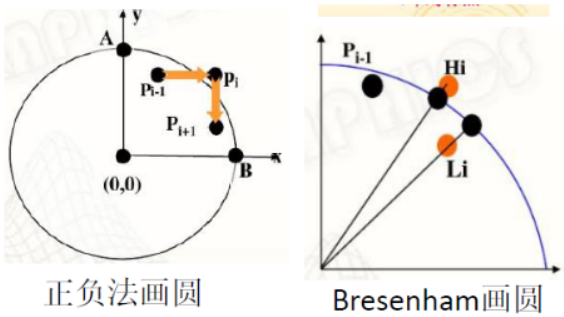
```
int x = start_x, y = start_y;
if (abs(dx) > abs(dy)) {
    int p = 2 * abs(dy) - abs(dx);
    while(x != end_x) {
        points.push_back(x);
        points.push_back(y);
        if (p <= 0) {
            p += 2 * abs(dy);
        }
        else {
            y += step_y;
            p += 2 * abs(dy) - 2 * abs(dx);
        }
        x += step_x;
    }
}
else {
    int p = 2 * abs(dx) - abs(dy);
    while (y != end_y) {
        points.push_back(x);
        points.push_back(y);
        if (p <= 0) {
            p += 2 * abs(dx);
        }
        else {
            x += step_x;
            p += 2 * abs(dx) - 2 * abs(dy);
        }
        y += step_y;
    }
}
```

结果如下:

2. 绘制圆形边框



根据圆的特点，每次画出一个点可以找到 7 个对称点，因此画出 1/8 的圆弧，根据对称翻转，就可以绘制出一整个圆来

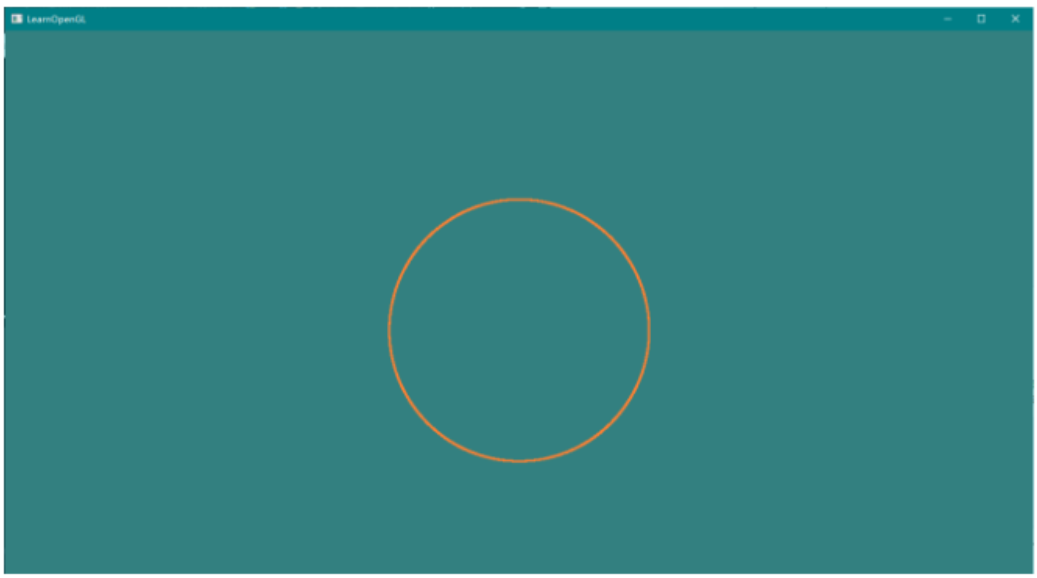


首先，第一个点选取圆的正上方 (0, r) 作为 (x_0, y_0)，通过判断 (x_i+1, y_i) 和 (x_i+1, y_i-1) 的中点是否在圆中，如果在圆中，那么就选择下面的点，否则就选择上面的点。而判断这个点是否在圆中可以把中点 $(x_i+1, y_i - 0.5)$ 代入圆的公式来看结果是否小于 0。

根据 Bresenham 算法的递推方法，推算出当 p 大于 0 或者小于 0 两种更新方法。然后算出第一个点 p_0为 1.25−r，为了避免浮点数的运算，我们把所有的结果扩大两倍。再限制圆形半径不得小于 2，p_0 就可以近似为3−2r

```
// 第一个点选取圆的正上方 (0, r) 作为 (x0, y0)
int x = 0;
int y = r;
int p = 3 - 2 * r;
while (x < y) {
    // 根据 Bresenham 算法的递推方法更新 p
    if (p < 0) {
        p = p + 4 * x + 6;
    }
    else {
        p = p + 4 * (x - y) + 10;
        y--;
    }
    x++;
    points.push_back(x0 + x); points.push_back(y0 + y);
    points.push_back(x0 + y); points.push_back(y0 + x);
    points.push_back(x0 + x); points.push_back(y0 - y);
    points.push_back(x0 - y); points.push_back(y0 + x);
    points.push_back(x0 - x); points.push_back(y0 + y);
    points.push_back(x0 + y); points.push_back(y0 - x);
    points.push_back(x0 - x); points.push_back(y0 - y);
    points.push_back(x0 - y); points.push_back(y0 - x);
}
```

效果如下：



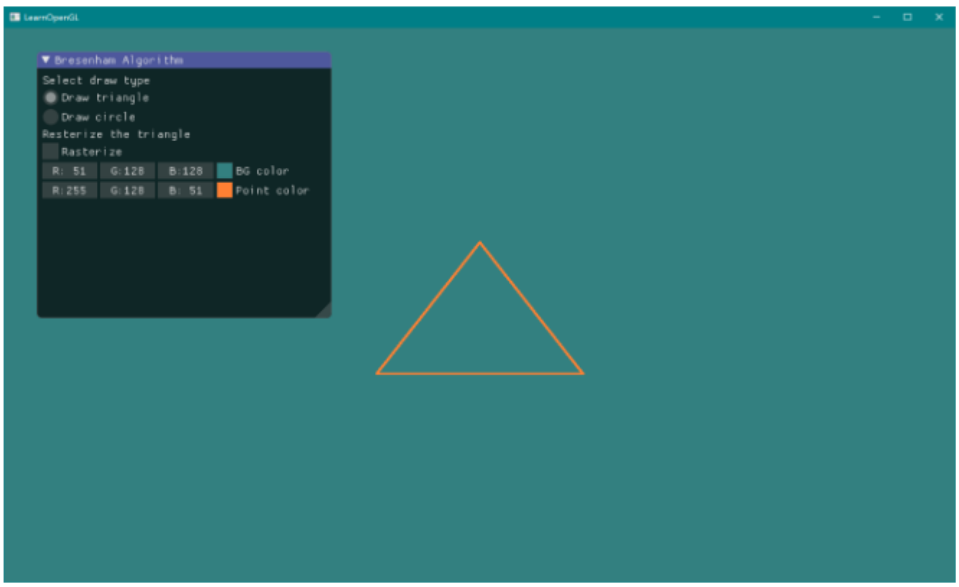
3. 添加菜单及调整圆大小

添加一个布尔型变量 draw_type，绑定到两个 RadioButton 上，分别绑定不同的值 0 和 1，即可实现单选菜单功能。再在循环渲染中判断 draw_type 的值，来决定绘制三角形或是圆形

```
ImGui::RadioButton("Draw triangle", &draw_type, 0);
ImGui::RadioButton("Draw circle", &draw_type, 1);
if (draw_type) {
    // draw circle...
} else {
    // draw triangle...
}
```

添加一个整型变量 radius 绑定到滑条 SliderInt 上，在调用 Bresenham算法函数计算圆上的点时将 radius 作为参数传入，从而控制圆的半径大小

```
ImGui::SliderInt("Radius", &radius, 1, 360);
drawPoints = Bresenham::drawCircle(0, 0, radius);
```

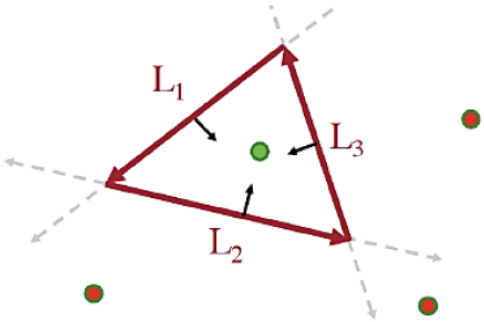


4. 填充三角形

用 Edge Equations 算法来实现光栅化

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);
    }
  }
}
```



- 计算出三角形的包围盒：x_{min}, x_{max}, y_{min}, y_{max}

```
// 遍历三角形包围盒中的所有点，找出满足在三角形内的点
int min_x = min(x1, min(x2, x3));
int max_x = max(x1, max(x2, x3));
int min_y = min(y1, min(y2, y3));
int max_y = max(y1, max(y2, y3));
```

- 计算出三角形三条边的直线方程
  - 直线方程可以使用直线的两点式来求出 Ax+By+C 中的ABC参数

    ```
    int A = y2 - y1;
    int B = x1 - x2;
    int C = x2 * y1 - x1 * y2;
    ```

  - 中心化直线方程

```
int x_temp[3] = { x3, x2, x1 };
int y_temp[3] = { y3, y2, y1 };
for (int i = 0; i < 3; i++) {
    // Ax + By + C
    if (lines[i][0] * x_temp[i] + lines[i][1] * y_temp[i] + lines[i][2] < 0) {
        for (int j = 0; j < lines[i].size(); j++) {
            lines[i][j] *= -1;
        }
    }
}
```

- 判断点是否在三角形的内部，是则对其进行填充。

```
for (int x = min_x; x <= max_x; x++) {
    for (int y = min_y; y <= max_y; y++) {
        bool isInside = true;
        for (int i = 0; i < 3; i++) {
            if (lines[i][0] * x + lines[i][1] * y + lines[i][2] < 0) {
                isInside = false;
                break;
            }
        }
        if (isInside) {
            // 填充
            pixels.push_back(x);
            pixels.push_back(y);
        }
    }
}
```

结果如下: