

Homework 3

Basic

[实验截图](#)

[算法实现](#)

1. Bresenham 画三角形
2. Bresenham 画圆
3. 添加 GUI, 可以改变圆的半径

Bonus

[实验截图](#)

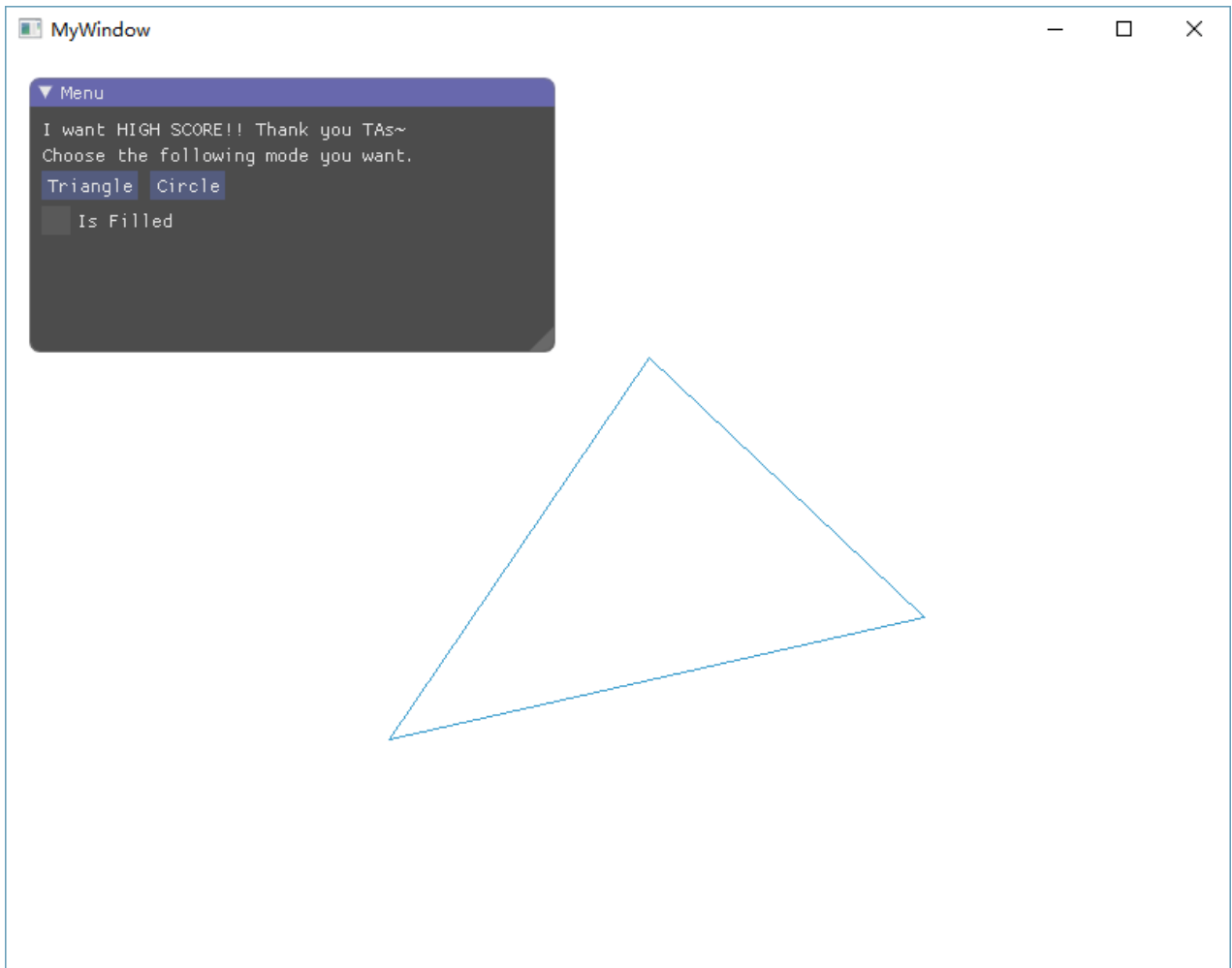
[算法实现](#)

Homework 3

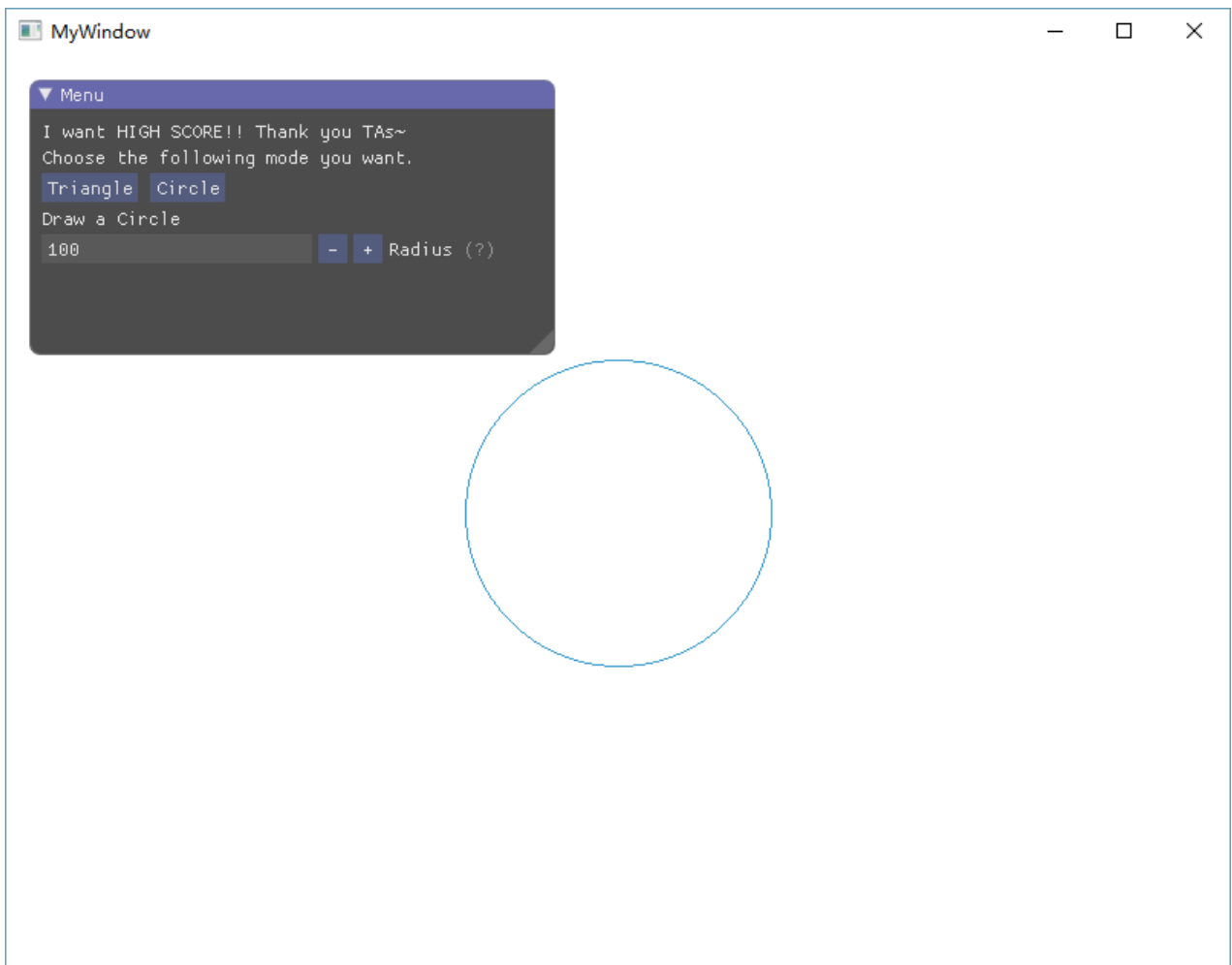
更加详细的实验结果可以参考gif文件

Basic

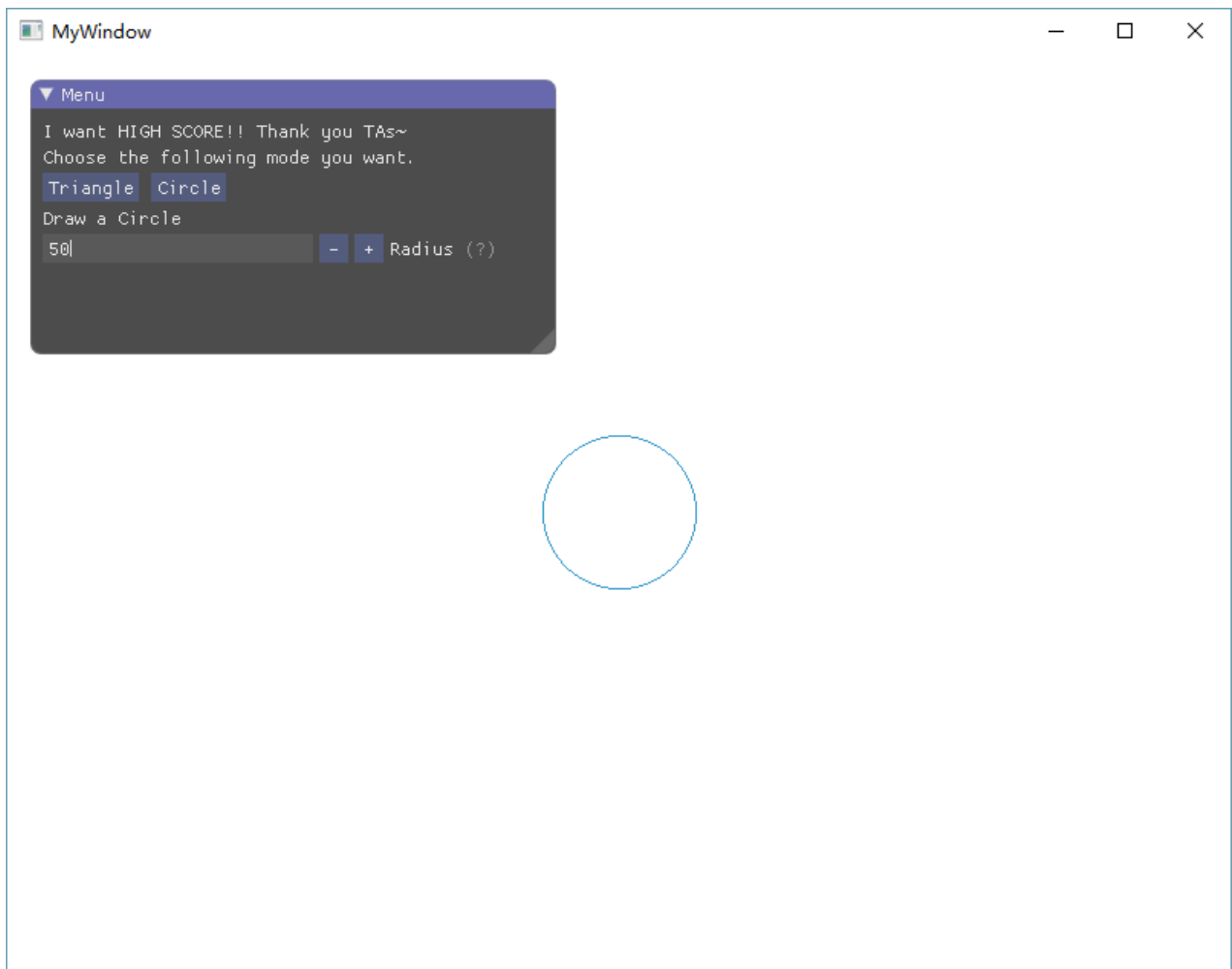
实验截图



任务1



任务2



任务3

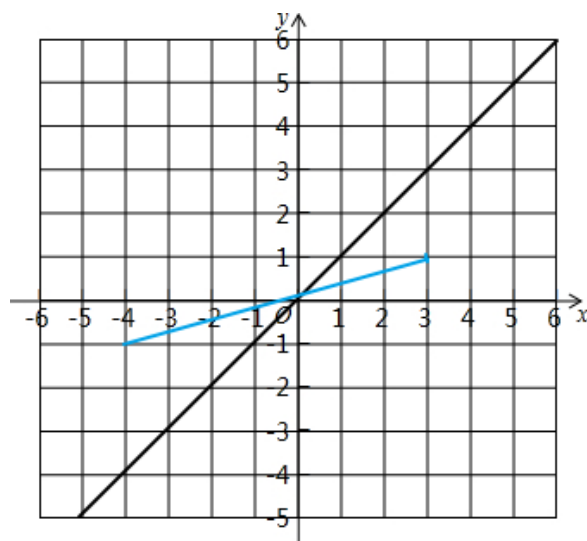
算法实现

1. Bresenham 画三角形

假设 $v0.x < v1.x$

$dx = v1.x - v0.x$

$dy = v1.y - v0.y$



基本情况

对于基本情况（蓝线）： $dx > dy$, $dy > 0$ ，使用老师课件上的算法完成：

Summary of Bresenham Algorithm

- **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**

```

int count = delta_x;
while (count-- > 0) {
    Point lastP = pv.back();
    if (p <= 0) {
        pv.push_back(Point(lastP.x + 1, lastP.y));
        p += 2 * delta_y;
    }
    else {
        pv.push_back(Point(lastP.x + 1, lastP.y + 1));
        p = p + 2 * delta_y - 2 * delta_x;
    }
}

```

对于其他情况，通过先作对称变换预处理，把情况转到基本情况，画完线后再把所有点逆变换回去

1. $dx < dy, dy > 0$: 将两个端点作关于直线 $y=x$ 的对称点 (`flipXY()`), 转到基本情况
2. $dx > dy, dy < 0$: 以左点的x轴方向为对称轴, 作右点关于对称轴的对称点(`flipX()`), 转到基本情况
3. $dx < dy, dy < 0$: 先 `flipXY()` ,再 `flipX()` , 转到基本情况。

```

1. if (v0.x > v1.x) {
2.     swap(v0, v1);
3. }
4. bool isflipXY = false;
5. if (std::abs(v0.x - v1.x) < std::abs(v0.y - v1.y)) {
6.     flipXY(v0);
7.     flipXY(v1);
8.     isflipXY = true;
9. }
10. if (v0.x > v1.x) {
11.     swap(v0, v1);
12. }
13. bool isflipX = false;
14. if (v0.y > v1.y) {
15.     flipX(v0, v1);
16.     isflipX = true;
17. }

```

将点的像素坐标表示完成后，使用 `scrCoor2glCoor()` 函数将其转为适合 OpenGL 的坐标。

```
std::vector<float> scrCoor2glCoor(std::vector<float>& _data, const unsigned int scr_width,
                                const unsigned int scr_height) {
    std::vector<float> data;
    data.resize(_data.size());
    for (int i = 0; i < _data.size(); i = i + 3) {
        data[i] = 2 * _data[i] / scr_width;
        data[i + 1] = 2 * _data[i + 1] / scr_height;
    }
    return data;
}
```

分别画出三条线，再统一显示出来，画出三角形。

2. Bresenham 画圆

初始化 $d = 3 - 2 * R$ ，然后根据d的值来迭代选点。每次选八分之一，然后使用 `addCirclePlot` 函数来添加其他7个对称的点的坐标。

```
while (x < y) {
    if (d < 0) {
        d = d + 4 * x + 6;
    }
    else {
        d = d + 4 * (x - y) + 10;
        --y;
    }
    ++x;
    addCirclePlot(pv, origin, x, y);
}
```

[参考资料](#)

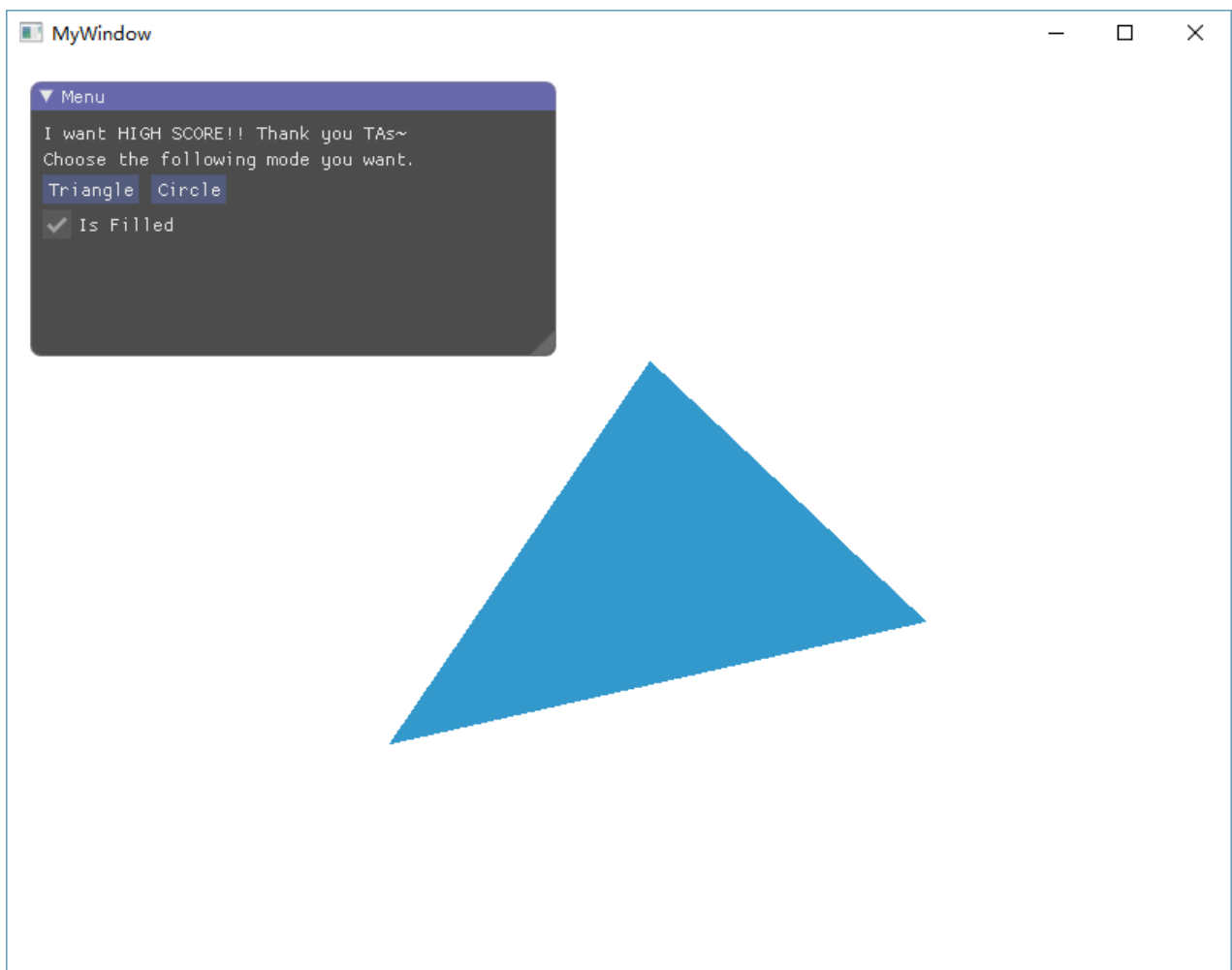
3. 添加 GUI，可以改变圆的半径

将一个 `curr_radius` 值和GUI关联起来，然后每次发现 `curr_radius` 和当前radius 的值不同时，更新radius 值，并且重新计算一次圆的数据点的数据，将其重新刷进对应的 VBO 中，在下次的 render loop 中渲染出新的圆形。

```
if (curr_radius != radius) {
    circleData.clear();
    circleData = Bresenham::genCircleData(origin, curr_radius);
    radius = curr_radius;
    pointData2vao(VAO[1], VBO[1], Utils::scrCoor2glCoor(circleData, scr_width, scr_height));
}
```

Bonus

实验截图



算法实现

主要使用 PPT 中的 edge equations 的转换算法。

实现的主要流程如下：

1. 通过两点计算一条直线的一般式 $Ax + By + C = 0$ 的公式如下：

$$\begin{aligned} C &= x_0y_1 - x_1y_0 \\ A &= y_0 - y_1 \\ B &= x_1 - x_0 \end{aligned}$$

2. 为了决定点在三角形内部的正负号，设立一个 `flag` 数组，使用除了两个端点以外的第三个点来确定符号的判断。

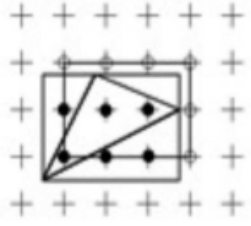
```
// use flag to determine halfspaces
int flag[3];
flag[0] = (f(lines[0], p2) > 0) ? 1 : -1;
flag[1] = (f(lines[1], p1) > 0) ? 1 : -1;
flag[2] = (f(lines[2], p0) > 0) ? 1 : -1;
```

3. 计算三角形的 bounding box

```

bound3( vert v[3], bbox& b )
{
    b.xmin = ceil(min(v[0].x, v[1].x, v[2].x));
    b.xmax = ceil(max(v[0].x, v[1].x, v[2].x));
    b.ymin = ceil(min(v[0].y, v[1].y, v[2].y));
    b.ymax = ceil(max(v[0].y, v[1].y, v[2].y));
}

```



4. 遍历 bounding box 里面的所有点，然后代入三条直线的一般式方程进行检验，如果算出来的三个值的符号和对应的 flag 值符号都一样的话，将该点的坐标加进数据vector中。

```

for (int i = b.xmin; i <= b.xmax; ++i) {
    for (int j = b.ymin; j <= b.ymax; ++j) {
        bool inside = true;
        for (int k = 0; k < lines.size(); ++k) {
            if ((lines[k][0] * i + lines[k][1] * j + lines[k][2]) * flag[k] < 0) {
                inside = false;
                break;
            }
        }
        if (inside) {
            pv.push_back(Point(i, j));
        }
    }
}
}

```