Homework 3 - Draw line

Homework 3 - Draw line
- 作业要求
- Bresenham直线
- Bresenham画圆
- GUI

作业要求

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。

Bresenham直线

Bresenham直线算法是用来描绘由两点所决定的直线的算法，它会算出一条线段在 n 维光栅上最接近的点。这个算法只会用到较为快速的整数加法、减法和位元移位，常用于绘制电脑画面中的直线。是计算机图形学中最先发展出来的算法。

经过少量的延伸之后，原本用来画直线的算法也可用来画圆。且同样可用较简单的算术运算来完成，避免了计算二次方程式或三角函数，或递归地分解为较简单的步骤。

初始时：$d = 2 \cdot \Delta y - \Delta x$

递推式：

当d≥0时：{ d=d+2·(Δy - Δx);　　y++;　　x++;　　} 否则：　{ d=d+2·Δy;　　x++;　　}

```
vector<Point> points;
    int flag;
    int ix, iy;
    int x = A.x, y = A.y;
    int dx = abs(B.x - A.x);
    int dy = abs(B.y - A.y);

    if (B.x > A.x) {
        ix = 1;
    }
    else {
        ix = -1;
    }
    if (B.y > A.y) {
        iy = 1;
    }
    else {
        iy = -1;
    }
    if (abs(B.y - A.y) > abs(B.x - A.x))
    {
        swap(&dx, &dy);
        flag = 1;
    }
    else
        flag = 0;

    int p = 2 * dy - dx;
    for (int i = 1; i <= dx; i++) {
        points.push_back(Point(x, y));
        if (p >= 0) {
            if(flag == 0)
                y = y + iy;
            else
                x = x + ix;
            p = p - 2 * dx;
        }
        if (flag == 0)
            x = x + ix;
        else
            y = y + iy;
        p = p + 2 * dy;
    }
    return points;
```

Bresenham画圆

圆是中心对称的特殊图形，具有八对称性，所以可以将圆八等分，因此我们只需要对八分之一的圆弧求解，就可以通过对称变换得到其他圆弧。

我们对$(0,R)-(-\sqrt{2}R,\sqrt{2}R)$的圆弧进行求解，$x_0=0,y_0=R$，此时最大位移方向为x方向，每次对x自增，然后判断y是否减1，直到x >= y为止。误差量由$F(x,y)=x^2+y^2-R^2$给出。

因为递推关系中只有整数运算, d的初值$d_0=F(1,R-0.5)=1.25-R=3-2*R$

```
vector<Point> Bresenham::drawCircle(Point center, int radius)
{
    int x = 0;
    int y = radius;
    int d = 3 - 2 * radius;
    vector<Point> points, res;
    while (x <= y) {
        points = getCirclePoints(center, Point(x, y));
        for (auto p : points) {
            res.push_back(p);
        }
        if (d < 0) {
            d = d + 4 * x + 6;
        }
        else {
            d = d + 4 * (x - y) + 10;
            y--;
        }
        x++;
    }
    points = getCirclePoints(center, Point(x, y));
    for (auto p : points) {
        res.push_back(p);
    }
    return res;
}
```

GUI

```
    ImGui::Begin("Draw Line\n");
    ImGui::SetWindowSize(ImVec2(300, 220));

    if (ImGui::RadioButton("Draw Triangle", triangle)) {
        triangle = true;
```

```
        circle = false;
    };
    if (ImGui::RadioButton("Draw Circle", circle)) {
        triangle = false;
        circle = true;
    };

    if(triangle){
        ImGui::Text("\nTriangle Rasterization");
        ImGui::Checkbox("Rasterize", &rasterization);
    }
    else if (circle) {
        ImGui::Text("Choose Radius:");
        ImGui::SliderFloat("Radius", &radius, 0.0f, 200.0f);
    }

    ImGui::End();
```