# 实验一 直线段生成绘制算法

## 实验报告

## 一、综述

直线的扫描转换是指在图形输出设备上，按照扫描线的顺序，确定一组最佳逼近于直线的像素点并对像素进行写操作。直线生成的具体问题是已知直线的两个端点,要求在图像输出设备上从起点到中点通过逐次循环迭代，找到最接近直线的像素点集合。

本次实验所涉及的算法有三种：DDA 算法、B 算法和重点分割法。在 MFC 环境中测试三种算法并对比分析三种算法的误差及效率。

## 二、程序框架

实验程序为 MFC 框架，cg2019DrawLineView.h 为视图层的头文件，负责声明各种成员变量和成员函数。cg2019DrawLineView.cpp 为视图层的源文件，负责实现直线的三种绘制、误差分析并显示、圆与圆弧绘制的功能。CCgEditControl.h 为窗口面板中的按键及文本定义成员变量及成员函数，CCgEditControl.cpp 实现面板的功能，如点击按键绘制图像、分析误差等。

## 三、算法描述

### 1. DDA 算法

（1）将给定端点作为输入参数；

（2）初始化，初值加上 0.5 确保精度；

（3）计算起止点水平和垂直的差值 dx 和 dy，选取较大者为最大位移，并作为迭代步数 step；

（4）从起始点开始确定相邻两点间的增量并进行地推计算。若 dx>dy,且 x1<x2，则 x 和 y 方向上的增量分别为 1 和斜率 m，若 dx>dy,且 x1>=x2，则 x 和 y 方向上的增量分别为-1 和斜率-m，若 dx<=dy,且 x1<x2，则 x 和 y 方向上的增量分别为 1/m 和 1，若 dx<=dy,且 x1>=x2，则 x 和 y 方向上的增量分别为-1/m 和-1；

（5）重复上一步直到循环结束；

（6）将所求得坐标取整并设置颜色值。

### 2. B 算法

（1）确定最大位移方向并计算误差初值 e = 2 * min - max;（min 和 max 分别为水平距离和垂直距离的最值）

（2）设置点(Xi, Yi) 的颜色值，并求下一误差 ei+1；

如果 ei > 0　　则 ei+1 = e - 2 * max;;

否则 ei+1 = ei + 2 * min;

（3）根据不同象限，确定 x 和 y 变化符号的正负，进行下一次循环；

（4）如果没有结束，则转到步骤 2，3；否则结束。

## 3. 中点分割法

（1）将直线段求中点坐标，若可以细分，则进行一次递归；

（2）如果中点坐标无法继续递归，则设置改像素的颜色值；

（3）执行至所有点都完成了颜色值的设置，程序结束。

## 4. 算法误差分析参数说明：

（1）RunTime 为多条直线生成的所有运行时间。本实验中三种直线算法生成的直线均相同，便于直接得出结果。时间越短，效率越高。

（2）Error 为直线生成算法所计算出的生成点距离理论直线距离的平均值，作为算法的误差。误差越小，算法精度越高。

（3）Smooth 为直线生成算法所计算出的生成点与直线起始点所连直线的斜率的平均值，作为平滑度。平滑度越小，直线斜率越接近理论斜率，越平滑。

## 5. B 圆弧生成算法

（1）确定误差初值d = 2 * (1 - r)，设置x = 0，y = r;

（2）设置当前点(Xi, Yi) 的颜色值，并求下一误差 di+1 及下一点坐标：

```
如果 d < 0，d1 = 2 * (d + y) - 1;
    如果 d1 < 0，x++;di+1 += 2 * x + 1;
    否则 x++;y--;    di+1 += 2 * (x - y + 1);
如果 d > 0，d2 = 2 * (d - x) - 1;
    如果 d2 < 0，x++;y--;    di+1 += 2 * (x - y + 1);
    否则 y--; di+1 += -2 * y + 1;
```

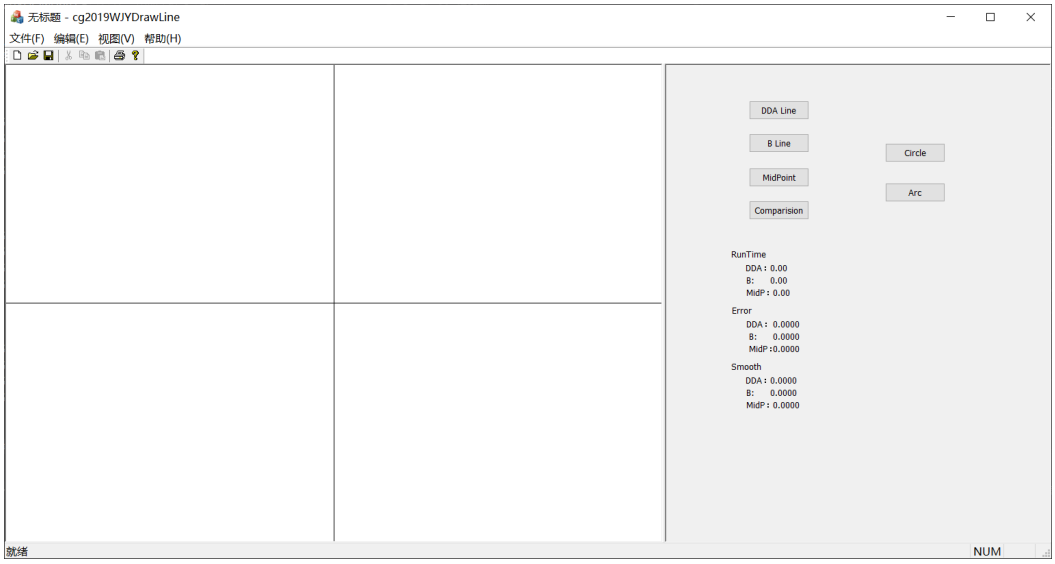
（3）如果没有结束，重复上一步，直到循环结束。

## 6. B 圆弧段生成算法

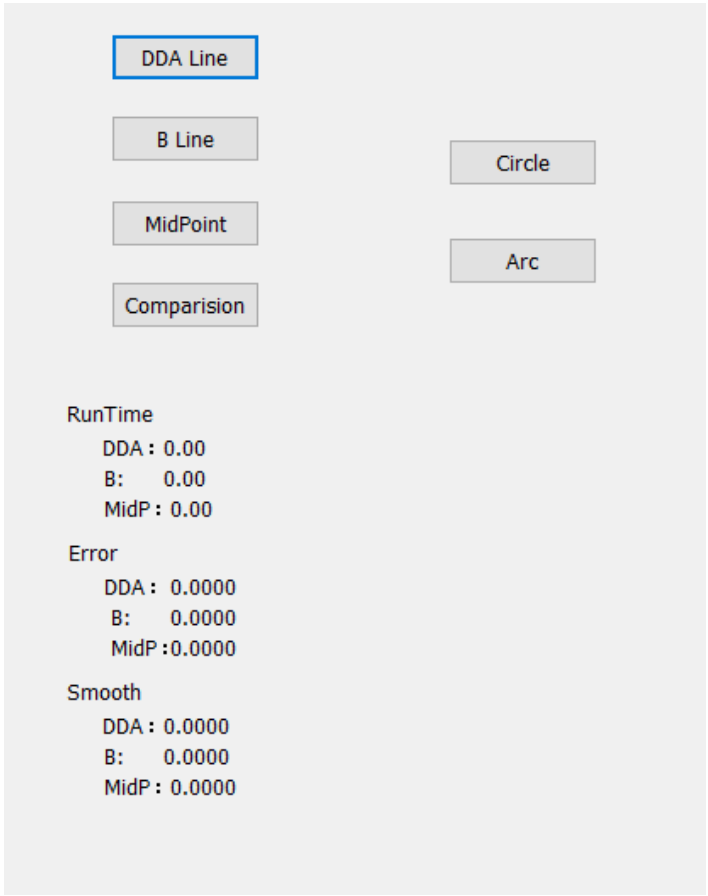
算法原理同 B 圆弧生成算法，通过变换，找出符合特定起始值的圆弧段。

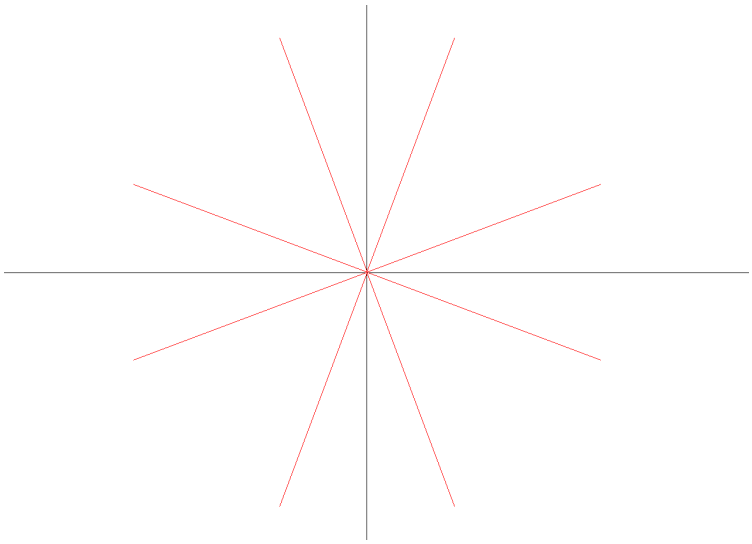
# 四、处理流程说明

首先建立直角坐标系

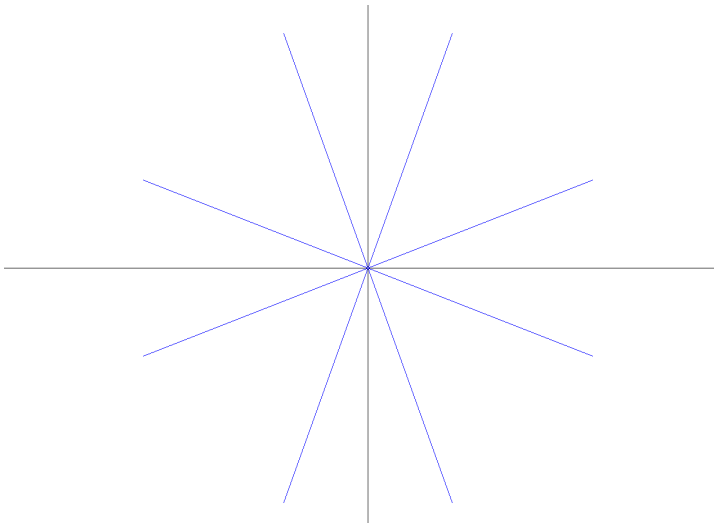



当点击 DDA Line 时调用 DDA 直线生成函数

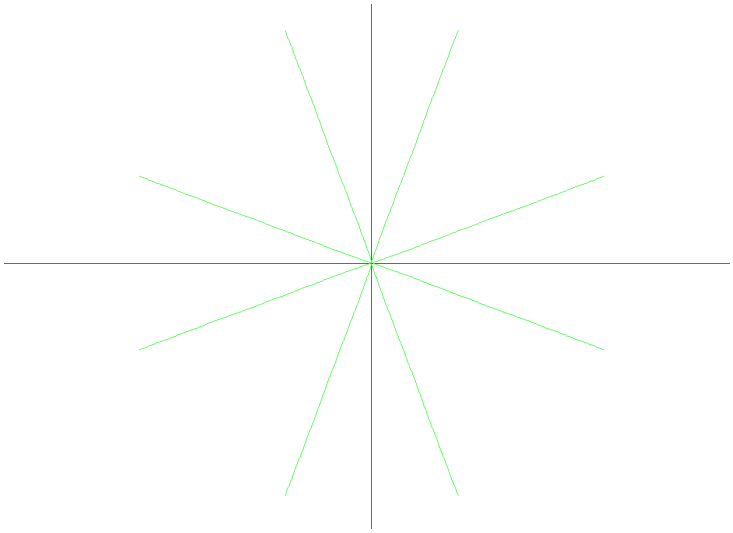




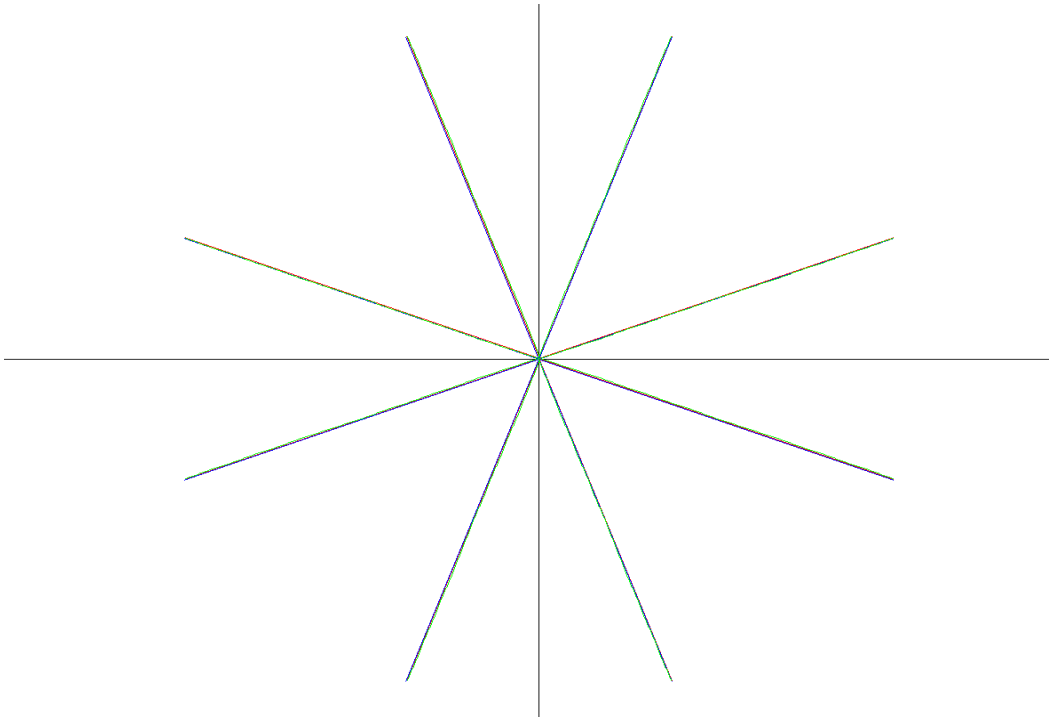
当点击 B Line 时调用 B 直线生成函数



当点击 MidPoint 时调用中点分割直线生成函数



点击 Comparision 时先后调用三种直线生成函数，并绘制在同一坐标系中，便于观察对比，并计算出调用时间、误差和光滑度



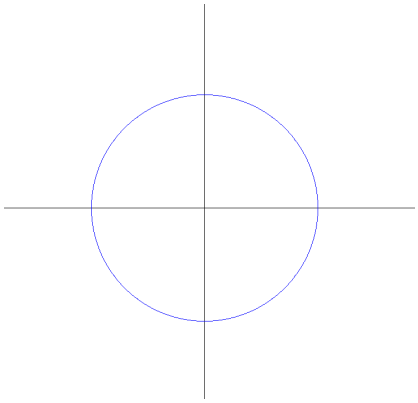
RunTime
DDA：47.00
B:　32.00
MidP：31.00

Error
DDA：0.2926
B:　0.2341
MidP：0.8047

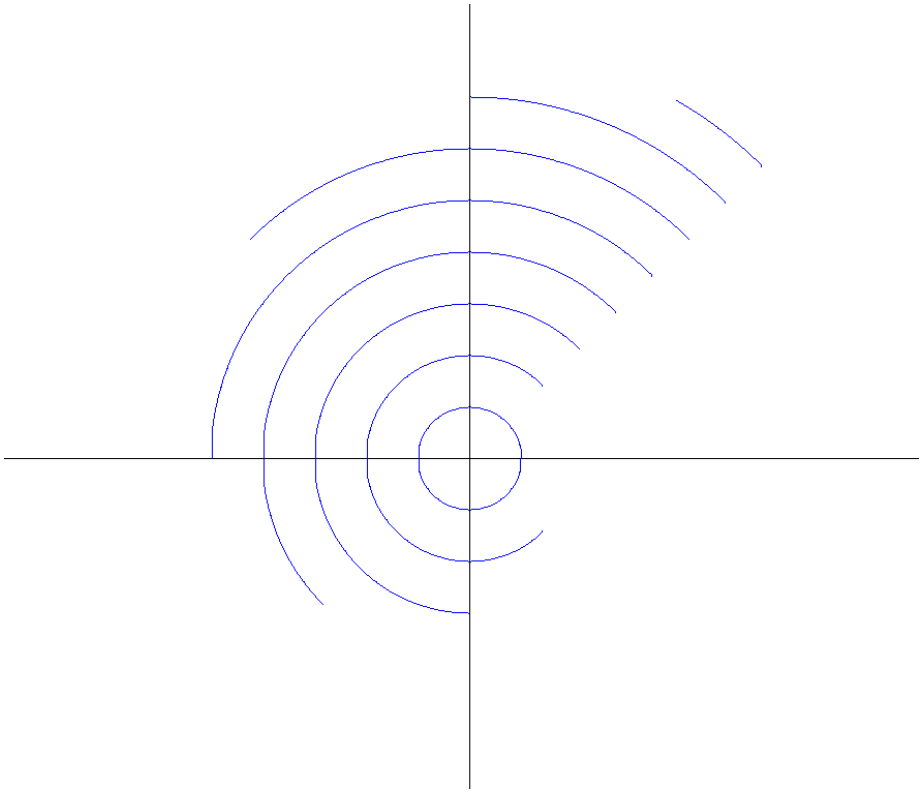
Smooth
DDA：0.0398
B:　0.0197
MidP：0.3725

其中，RunTime 值越低表示效率越高，Error 越小表示误差越小，Smooth 越小表示直线越光滑。

当点击 Circle 时调用 B 算法圆生成函数



当点击 Arc 时调用 B 算法任意弧度圆弧生成函数



## 五、算法对比

从误差结果来看，DDA 算法时间较长，B 算法精度最高，误差最小，重点分割法误差较大，且重点分割法平滑度较低，不如 DDA 算法和 B 算法。因此：

（1）DDA 算法的缺点在于存在除法运算和浮点数，使得程序运行时间延长，会降低程序运行的效率，优点是较为简单，直接明了。

（2）Bresenham 算法的优点在于只有加减运算，因此算法实现效率高，速度快，且精度较高。

（3）中点分割法较为简单，但缺点在于生成的直线较为粗糙，精度较低。

## 六．源代码

```
//DDA算法
void Ccg2019DrawLineView::DDAline(int x0, int y0, int x1, int y1, CDC* pDC)
{
    //误差参数
    float distance = 0.0f; //生成点距理想直线的距离
    float kaverage = 0.0f; //生成点和起点连线的斜率平均值
    float xp, yp; //垂足点的坐标

    int step = abs(x1 - x0) >= abs(y1 - y0) ? abs(x1 - x0) : abs(y1 - y0);
```

```
    float dx = (float)(x1 - x0) / step;
    float dy = (float)(y1 - y0) / step;
    float m = (float)(y1 - y0) / (float)(x1 - x0);
    float x = x0 + 0.5f;
    float y = y0 + 0.5f;
    for (int i = 0; i < step; i++)
    {
        pDC->SetPixel((int)x + m_wndWidth / 2, (int)m_wndHeight / 2 - y, RGB(255, 0,
0));

        //计算垂足坐标
        xp = (y - y0 + x / m + m * x0) / (m + 1 / m);
        yp = -1 / m * (xp - x) + y;
        distance += sqrtf((x - xp) * (x - xp) + (y - yp) * (y - yp)); //生成点距离直线
的距离累加值
        if (x != x0)
            kaverage += fabs(y - y0) / fabs(x - x0); //计算生成点和起点连线的斜率平均
值

        x += dx ;
        y += dy;
    }
    ddaError = distance / step; //误差为距离的平均值
    ddaSmooth = fabs(kaverage / step - m); //平滑度为斜率的平均值
}

//B算法
void Ccg2019DrawLineView::Bline(int x0, int y0, int x1, int y1, CDC* pDC) {

    //误差参数
    float distance = 0.0f; //生成点距理想直线的距离
    float kaverage = 0.0f; //生成点和起点连线的斜率平均值
    float xp, yp; //垂足点的坐标
    float m = (float)(y1 - y0) / (float)(x1 - x0);

    int x = x0, y = y0, dx = x1 - x0, dy = y1 - y0;
    int max = (abs(dy) > abs(dx)) ? abs(dy) : abs(dx);
    int min = (abs(dy) > abs(dx)) ? abs(dx) : abs(dy);
    float e = 2 * min - max;
    for (int i = 0; i < max; i++)
    {
        pDC->SetPixel((int)x + m_wndWidth / 2, (int)m_wndHeight / 2 - y, RGB(0, 0,
255));
```

```
        //计算垂足坐标
        xp = (y - y1 + x / m + m * x1) / (m + 1 / m);
        yp = -1 / m * (xp - x) + y;
        distance += sqrtf((x - xp) * (x - xp) + (y - yp) * (y - yp)); //生成点距离直线的距离累加值
        if (x != x0)
            kaverage += (float)fabs(y - y0) / (float)fabs(x - x0); //计算生成点和起点连线的斜率平均值

        if (e >= 0)
        {
            e = e - 2 * max;
            (abs(dy) > abs(dx)) ? (dx >= 0 ? x++ : x--) : (dy >= 0 ? y++ : y--);
        }
        e += 2 * min;
        (abs(dy) > abs(dx)) ? (dy >= 0 ? y++ : y--) : (dx >= 0 ? x++ : x--);
    }

    bError = distance / max; //误差为距离的平均值
    bSmooth = fabs(kaverage / max - m); //平滑度为斜率的平均值
}

//中点分割算法
void Ccg2019DrawLineView::Midpoint(int x0, int y0, int x1, int y1, CDC* pDC) {
    if (abs(x0 - x1) <= 1 && abs(y0 - y1) <= 1) {
        pDC->SetPixel((int)((x0 + x1) / 2 + 0.5) + m_wndWidth / 2, (int)m_wndHeight / 2 - ((y0 + y1) / 2 + 0.5), RGB(0, 255, 0)); }
    else {Midpoint(x0, y0, (x0 + x1) / 2, (y0 + y1) / 2, pDC);
        Midpoint((x0 + x1) / 2, (y0 + y1) / 2, x1, y1, pDC);}
}

//中点分割算法误差分析
void Ccg2019DrawLineView::MidpointAnalyse(int x0, int y0, int x1, int y1, CDC* pDC) {

    //误差参数
    float distance = 0.0f; //生成点距理想直线的距离
    float kaverage = 0.0f; //生成点和起点连线的斜率平均值
    float xp, yp; //垂足点的坐标
    static int x00 = x0, y00 = y0, x11 = x1, y11 = y1; //记录起始点的坐标
    float m = (float)(y11 - y00) / (float)(x11 - x00);
    int dx = x11 - x00, dy = y11 - y00;
    int max = (abs(dy) > abs(dx)) ? abs(dy) : abs(dx);

    if (abs(x0 - x1) <= 1 && abs(y0 - y1) <= 1) {
```

```
        pDC->SetPixel((int)((x0 + x1) / 2 + 0.5) + m_wndWidth / 2, (int)m_wndHeight /
2 - ((y0 + y1) / 2 + 0.5), RGB(0, 255, 0));

        //计算垂足坐标
        xp = (y00 - y1 + x00 / m + m * x1) / (m + 1 / m);
        yp = -1 / m * (xp - x00) + y00;
        distance += sqrtf((x00 - xp) * (x00 - xp) + (y00 - yp) * (y00 - yp));  //生成
点距离直线的距离累加值
        kaverage += (float)fabs((y0 + y1) / 2 + 0.5 - y0) / (float)fabs((x0 + x1) / 2
+ 0.5 - x0); //计算生成点和起点连线的斜率平均值
        mpSmooth = fabs(kaverage / max - m); //误差为距离的平均值
        mpError = distance / max; //平滑度为斜率的平均值
    }
    else {
        MidpointAnalyse(x0, y0, (x0 + x1) / 2, (y0 + y1) / 2, pDC);
        MidpointAnalyse((x0 + x1) / 2, (y0 + y1) / 2, x1, y1, pDC);
    }
}
//B圆弧生成算法
void Ccg2019DrawLineView::BCircle(int x0, int y0, int r, CDC* pDC) {

    int x, y, d1, d2, direction, d;
    x = 0;
    y = r;
    d = 2 * (1 - r);
    while (y >= 0) {
        pDC->SetPixel(x0 + x, y0 + y, RGB(0, 0, 255));
        pDC->SetPixel(x0 - x, y0 + y, RGB(0, 0, 255));
        pDC->SetPixel(x0 - x, y0 - y, RGB(0, 0, 255));
        pDC->SetPixel(x0 + x, y0 - y, RGB(0, 0, 255));
        if (d < 0) {
            d1 = 2 * (d + y) - 1;
            if (d1 < 0) direction = 1;
            else     direction = 2;
        }
        else if (d > 0) {
            d2 = 2 * (d - x) - 1;
            if (d2 < 0) direction = 2;
            else     direction = 3;
        }
        else direction = 3;
        switch (direction)
        {
            case 1:
```

```
                x++;
                d += 2 * x + 1;
                break;
            case 2:
                x++;
                y--;
                d += 2 * (x - y + 1);
                break;
            case 3:
                y--;
                d += -2 * y + 1;
                break;
        }
    }
}
//B象限内圆弧生成算法
void Ccg2019DrawLineView::ShortArc(float θ1, float θ2, float R, CDC* pDC) {

    float θa, θb;
    int flag = 0;
    int x, y, d1, d2, direction, d;
    if (cos(θ1) >= 0 && sin(θ1) >= 0 && cos(θ2) >= 0 && sin(θ2) >= 0)
    {
        flag = 1;
        θa = θ1;
        θb = θ2;
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) >= 0 && cos(θ2) - 0 <= 1e-6 && sin(θ
2) >= 0)
    {
        flag = 2;
        θa = PI - θ2;
        θb = PI - θ1;
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) - 0 <= 1e-6 && cos(θ2) - 0 <= 1e-6 &&
sin(θ2) - 0 <= 1e-6)
    {
        flag = 3;
        θa = θ1 - PI;
        θb = θ2 - PI;
    }
    else if (cos(θ1) >= 0 && sin(θ1) - 0 <= 1e-6 && cos(θ2) >= 0 && sin(θ2) - 0 <=
1e-6)
    {
```

```
        flag = 4;
        θa = 2 * PI - θ2;
        θb = 2 * PI - θ1;
    }

    x = R * cos(θa);
    y = R * sin(θa);
    d = (x - 1) * (x - 1) + (y + 1) * (y + 1) - R * R;
    while (abs(x) >= R * fabs(cos(θb))){
        if (flag == 1)
            pDC->SetPixel(m_wndWidth / 2 + x, m_wndHeight / 2 - y, RGB(0, 0, 255));
        if (flag == 2)
            pDC->SetPixel(m_wndWidth / 2 - x, m_wndHeight / 2 - y, RGB(0, 0, 255));
        if (flag == 3)
            pDC->SetPixel(m_wndWidth / 2 - x, m_wndHeight / 2 + y, RGB(0, 0, 255));
        if (flag == 4)
            pDC->SetPixel(m_wndWidth / 2 + x, m_wndHeight / 2 + y, RGB(0, 0, 255));
        if (d < 0) {
            d1 = 2 * (d + y) - 1;
            if (d1 < 0) direction = 1;
            else     direction = 2;
        }
        else if (d > 0) {
            d2 = 2 * (d - x) - 1;
            if (d2 < 0) direction = 2;
            else     direction = 3;
        }
        else direction = 3;
        switch (direction)
        {
        case 1:
            y++;
            d += 2 * y + 1;
            break;
        case 2:
            y++;
            x--;
            d += 2 * (y - x + 1);
            break;
        case 3:
            x--;
            d += -2 * x + 1;
            break;
        }
```

```
    }

}

//B任意弧度圆弧生成算法
void Ccg2019DrawLineView::BresenhamArc(float θ1, float θ2, float R, CDC* pDC) {

    if (θ1 == θ2)
    {
        ShortArc(0, PI / 2, R, pDC);
        ShortArc(PI / 2, PI, R, pDC);
        ShortArc(PI, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2, 0, R, pDC);
    }
    else if (cos(θ1) >= 0 && sin(θ1) >= 0 && cos(θ2) >= 0 && sin(θ2) >= 0)
    {
        ShortArc(θ1, θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) >= 0 && cos(θ2) - 0 <= 1e-6 && sin(θ
2) >= 0)
    {
        ShortArc(θ1, θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) - 0 <= 1e-6 && cos(θ2) - 0 <= 1e-6 &&
sin(θ2) - 0 <= 1e-6)
    {
        ShortArc(θ1, θ2, R, pDC);
    }
    else if (cos(θ1) >= 0 && sin(θ1) - 0 <= 1e-6 && cos(θ2) >= 0 && sin(θ2) - 0 <=
1e-6)
    {
        ShortArc(θ1, θ2, R, pDC);
    }


    else if (cos(θ1) >= 0 && sin(θ1) >= 0 && cos(θ2) - 0 <= 1e-6 && sin(θ2) >= 0)
    {
        ShortArc(θ1, PI / 2, R, pDC);
        ShortArc(PI / 2, θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) >= 0 && cos(θ2) - 0 <= 1e-6 && sin(θ2)
- 0 <= 1e-6)
    {
        ShortArc(θ1, PI, R, pDC);
        ShortArc(PI, θ2, R, pDC);
```

```
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) - 0 <= 1e-6 && cos(θ2) >= 0 && sin(θ2)
- 0 <= 1e-6)
    {
        ShortArc(θ1, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2,  θ2, R, pDC);
    }
    else if (cos(θ1) >= 0 && sin(θ1) - 0 <= 1e-6 && cos(θ2) - 0 >= 0 && sin(θ2) >=
0)
    {
        ShortArc(θ1, 0, R, pDC);
        ShortArc(0,  θ2, R, pDC);
    }

    else if (cos(θ1) >= 0 && sin(θ1) >= 0 && cos(θ2) - 0 <= 1e-6 && sin(θ2) - 0 <=
1e-6)
    {
        ShortArc(θ1, PI / 2, R, pDC);
        ShortArc(PI / 2, PI, R, pDC);
        ShortArc(PI,  θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) >= 0 && cos(θ2) - 0 >= 0 && sin(θ2) - 0
<= 1e-6)
    {
        ShortArc(θ1, PI, R, pDC);
        ShortArc(PI, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2,  θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) - 0 <= 1e-6 && cos(θ2) >= 0 && sin(θ
2) >= 0)
    {
        ShortArc(θ1, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2, 0, R, pDC);
        ShortArc(0,  θ2, R, pDC);
    }
    else if (cos(θ1) >= 0 && sin(θ1) - 0 <= 1e-6 && cos(θ2) <= 1e-6 && sin(θ2) >=
0)
    {
        ShortArc(θ1, 0, R, pDC);
        ShortArc(0, PI / 2, R, pDC);
        ShortArc(PI / 2,  θ2, R, pDC);
    }

    else if (cos(θ1) >= 0 && sin(θ1) >= 0 && cos(θ2) >= 0 && sin(θ2) - 0 <= 1e-6)
```

```
    {
        ShortArc(θ1, PI / 2, R, pDC);
        ShortArc(PI / 2, PI, R, pDC);
        ShortArc(PI, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2,  θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) >= 0 && cos(θ2) - 0 >= 0 && sin(θ2) >=
0)
    {
        ShortArc(θ1, PI, R, pDC);
        ShortArc(PI, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2, 0, R, pDC);
        ShortArc(0,  θ2, R, pDC);
    }
    else if (cos(θ1) - 0 <= 1e-6 && sin(θ1) - 0 <= 1e-6 && cos(θ2) - 0 <= 1e-6 &&
sin(θ2) >= 0)
    {
        ShortArc(θ1, 3 * PI / 2, R, pDC);
        ShortArc(3 * PI / 2, 0, R, pDC);
        ShortArc(0, PI / 2, R, pDC);
        ShortArc(PI / 2,  θ2, R, pDC);
    }
    else if (cos(θ1) >= 0 && sin(θ1) - 0 <= 1e-6 && cos(θ2) <= 1e-6 && sin(θ2) <=
1e-6)
    {
        ShortArc(θ1, 0, R, pDC);
        ShortArc(0, PI / 2, R, pDC);
        ShortArc(PI / 2, PI, R, pDC);
        ShortArc(PI,  θ2, R, pDC);
    }
}
```