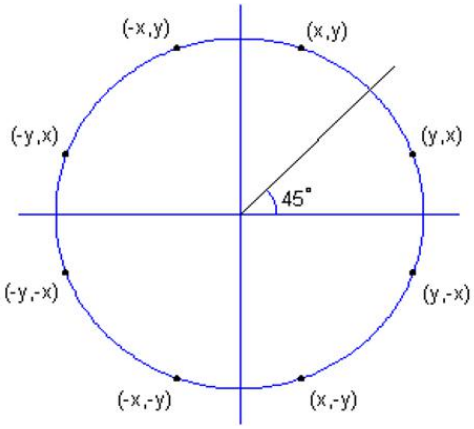
二、Bresenham 画圆算法原理和实践

1、原理

1）圆的八对称性

假设圆的圆心位于坐标原点，此时知道圆上的一个点 (x, y)，根据圆的对称性便可以得到关于四条对称轴的其它 7 个点，即：

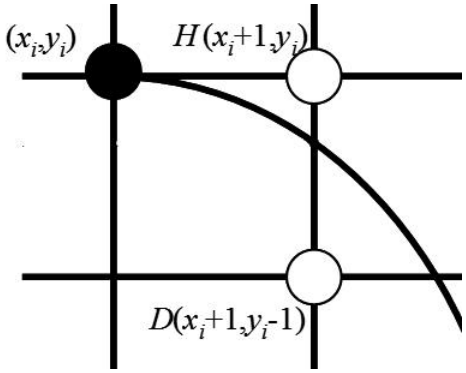


这种性质称为八对称性。

因此，只要能够画出八分之一圆弧，就可以通过圆弧的八对称性得到整个圆。

2）算法推导

在 0≤x≤y 的 1/8 圆周上，坐标 x 值单调增加，y 值单调减少。



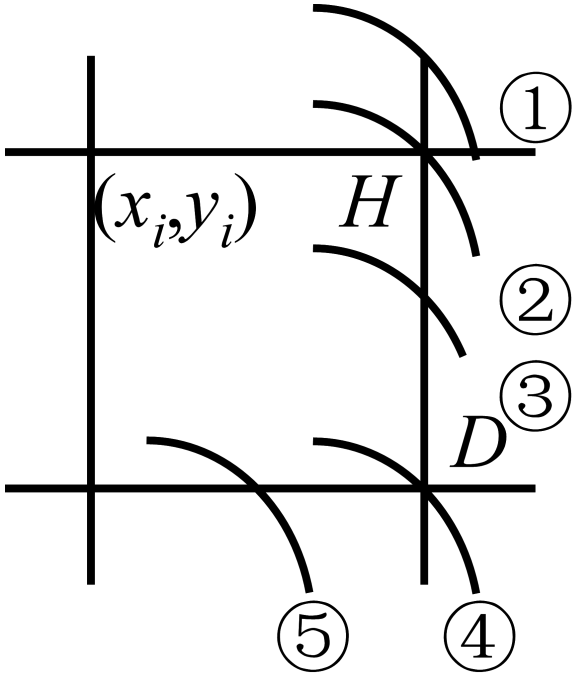
设第 i 步已确定 (xi, yi) 是要画圆上的象素点，第 i+1 步象素点只能是点 H (xi+1, yi) 或者点 D (xi+1, yi - 1) 中的一个。

可以使用公式：F(x, y) = |x^2 + y^2 - R^2|，来计算点 (x, y) 和圆之间的距离。

所以，**H 点和圆之间的距离为：dH = (xi + 1)^2 + yi^2 - R^2**

D 点和圆之间的距离为：dD = R^2 - (xi + 1)^2 – (yi – 1)^2

第 i+1 步象素点的选择情况分析（假设 pi = dH - dD）：



①或②，选 H 点，此时 dH≤0，dD>0，必有 pi<0；
④或⑤，选 D 点，此时 dH>0，dD≤0，必有 pi>0；
③，则 dH>0 和 dD>0．若 pi<0，即 dH<dD 应选 H 点，若 pi≥0，即 dH≥dD 应选 D 点。
得出结论：pi 做判别量，pi<0 选 H 点为下一个象素点，当 pi≥0 时，选 D 点为下一个象素点。
从 pi 计算 p(i+1)：

$$p_i = 2(x_i+1)^2 + 2y_i^2 - 2y_i - 2R^2 + 1$$

$$p_{i+1} - p_i = 2(y_{i+1}^2 - y_i^2 - y_{i+1} + y_i) + 4x_i + 6$$

当 pi≥0 时，应选 D 点，即选 y(i+1) = yi - 1
所以 **p(i+1) = pi + 4(xi - yi) + 10**
当 pi<0 时，应选 H，即选 y(i+1) = yi
所以 **p(i+1) = pi + 4xi + 6**
画圆的起始点是(0，R)，即 x1=0，y1=R，令 i=1，就得到：**p1 = 3 – 2R**

在以上推导过程中，我们一直假设圆心为坐标原点，如果圆的圆心为 (x, y)，只需要用 xi – x 代替 xi，用 yi-y 代替 yi 即可。

2、实践

1）题目要求

因为使用鼠标左键拖拽在视图区绘图功能、橡皮线功能、视图重画功能、双缓冲技术（防止闪屏）等在“一、DDA 直线扫描转换算法原理和实践”中已经实现，所以在这次实践中只是简单的验证一下 Bresenham 画圆算法即可，要求使用 SetPixel 函数在视图区域绘制一个圆心为(100, 100)，半径为 50 的红色的圆。

2）代码实现

（1）Bresenham 画圆算法实现：

```
void CTestView::BresenhamCircle(CDC * pDC, int x, int y, int R, COLORREF color)
{
    int x1 = x;
    int y1 = y + R;
    int p = 3 - 2 * R;
    for (; x1 - x <= y1 - y; x1++)
    {
        int x2 = x1 - x;
        int y2 = y1 - y;
        pDC->SetPixel(x + x2, y + y2, color);
        pDC->SetPixel(x + y2, y + x2, color);
        pDC->SetPixel(x + x2, y - y2, color);
        pDC->SetPixel(x + y2, y - x2, color);
        pDC->SetPixel(x - x2, y - y2, color);
        pDC->SetPixel(x - y2, y - x2, color);
        pDC->SetPixel(x - x2, y + y2, color);
        pDC->SetPixel(x - y2, y + x2, color);

        if (p >= 0)
        {
            p += 4 * (x1 - x - y1 + y) + 10;
            y1--;
        }
        else
        {
            p += 4 * (x1 - x) + 6;
        }
    }
}
```

（2）实现 OnDraw 函数：

```
void CTestView::OnDraw(CDC* pDC)
{
	CTestDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	if (!pDoc)
		return;

	BresenhamCircle(pDC, 100, 100, 50, RGB(255, 0, 0));
}
```

3）效果