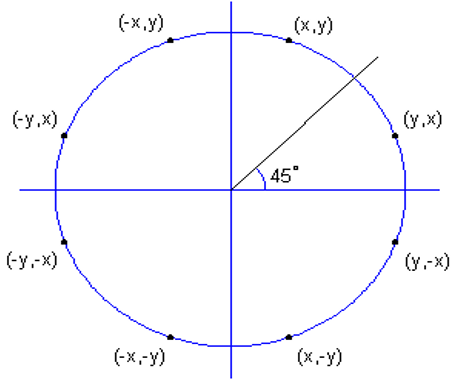
# 二、Bresenham画圆算法原理和实践

## 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：

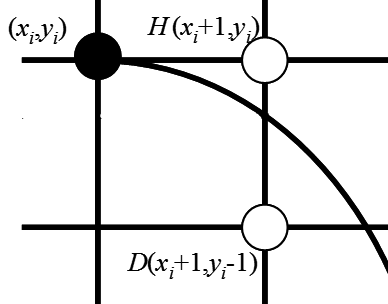


这种性质称为八对称性。

**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）：

figure2_7

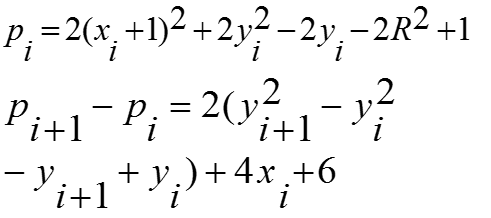
①或②，选H点，此时dH≤0，dD>0，必有pi<0；

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)：



**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

