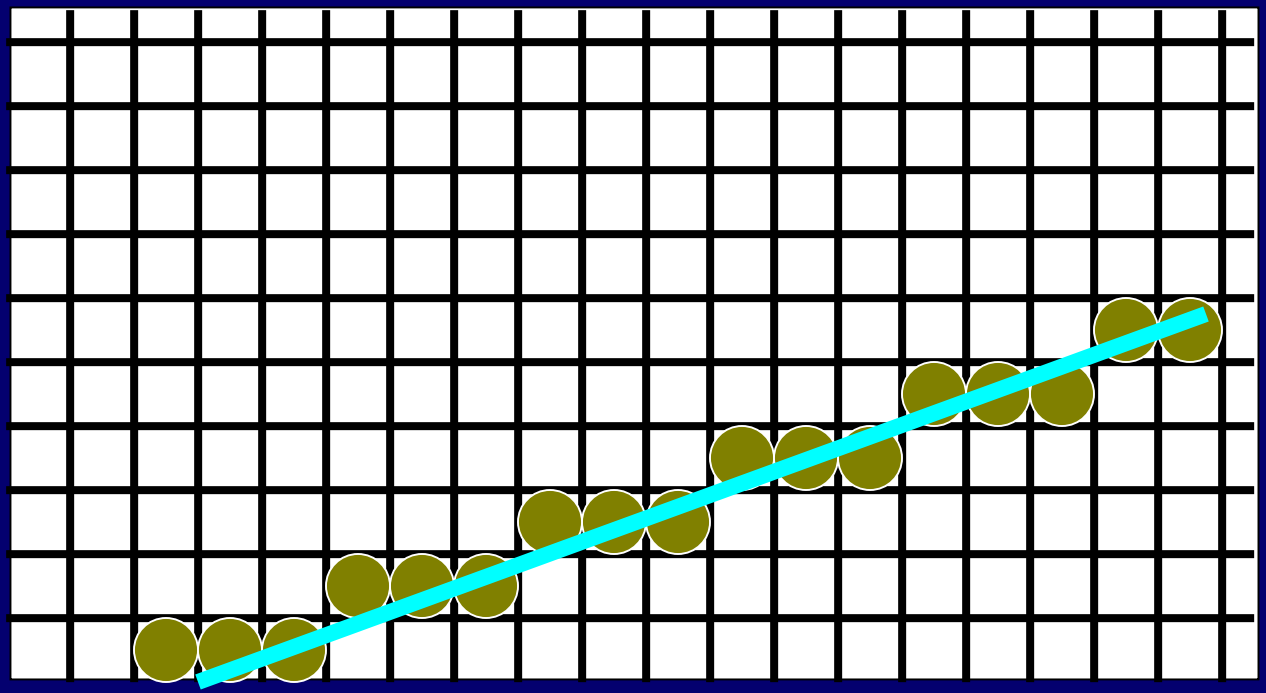
基本图元的生成算法

- **提出问题**

  如何在指定的输出设备上根据坐标描述构造基本二维几何图形（点、直线、圆、椭圆、多边形域、字符串及其相关属性等）。

  光栅化图形

- 光栅图形

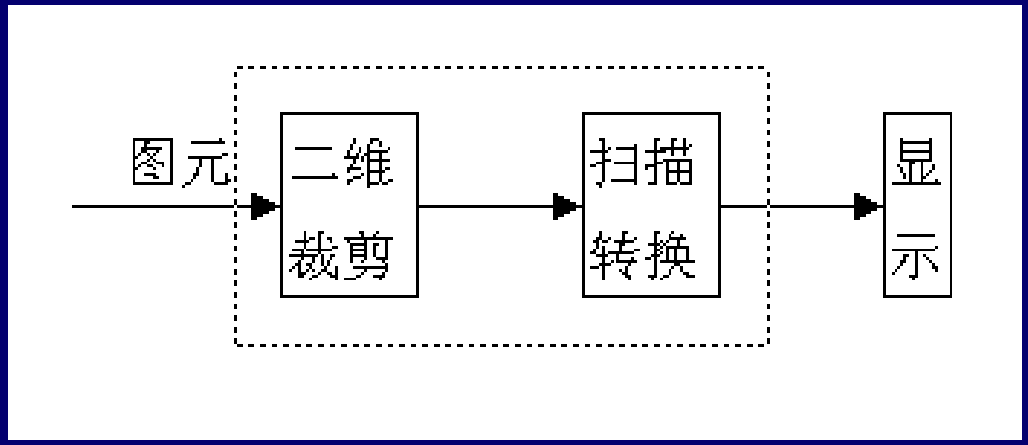


简单的二维图形显示流程图

- 扫描转换

用户　　光栅显示系统

- 简单流程图



- 图元生成的基本方法是扫描转换
  - 扫描转换的主要任务是确定最佳逼近于图形的像素集合

- 扫描转换的步骤
  - 首先确定有关的像素的位置（坐标等）
  - 用颜色或其他属性对所涉及的像素进行写操作

- 图元生成的基本要求（以直线为例）
  - **理想化直线的要求**
    1.生成的直线要直----尽可能逼近
    2.精确过端点----如何终止
    3.速度要快----优化算法

4.1 直线的扫描转换

- *解决的问题:*

给定直线两端点P0(x0,y0)和P1(x1,y1)，画出该直线。

4.1.1 数值微分法(DDA法)

直线的微分方程:

$$\frac{dy}{dx}=\frac{\Delta y}{\Delta x}=\frac{y_1-y_0}{x_1-x_0}=k$$

DDA算法原理：

基本思想：

在绘制直线的过程中，每绘制一个点就是原直线四舍五入到整数象素点，这样一步步逼近直线

- 直线段p(0,0)--P1(5,2)

x	y	四舍五余
0	0	0
1	0.4	0
2	0.8	1
3	1.2	1
4	1.6	2
5	2.0	2

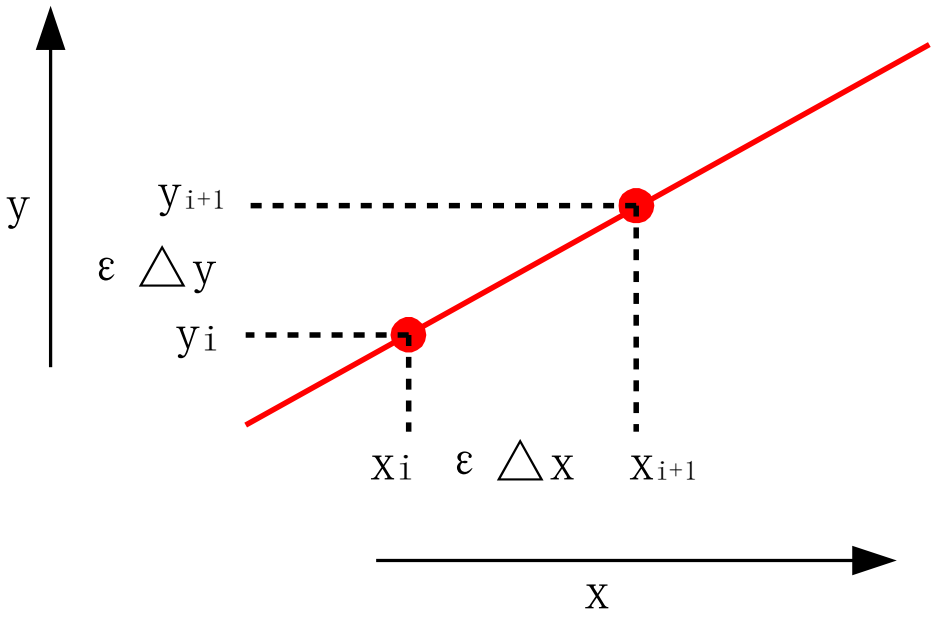


图5-2　DDA算法原理

DDA算法原理：

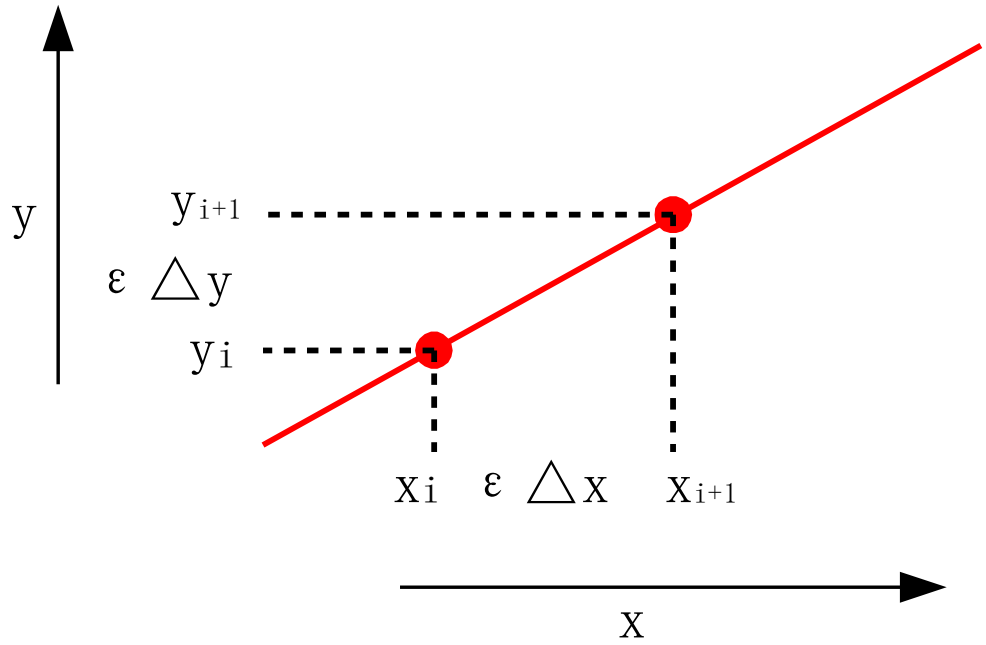


图5-2　DDA算法原理

$$x_{i+1} = x_i + \Delta x$$

$$y_{i+1} = y_i + \Delta y = y_i + k \cdot \Delta x$$

如果 $\Delta y < \Delta x$ ，取 $\Delta x = 1$

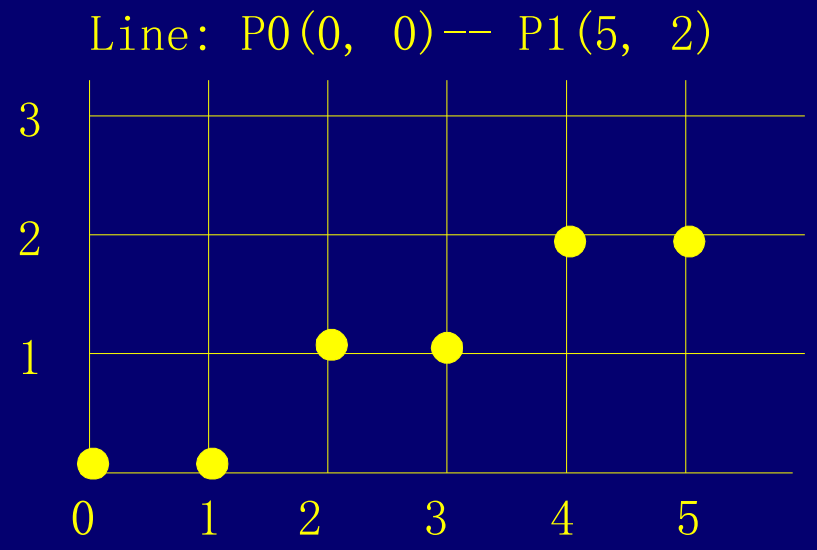
取象素（X_i+1, round(y_i+K)）

- 从而：
  - 当△x=1,有$y_{i+1}=y_i+k$, 即x每增1,y递增k
  - 简化后的计算：

```
for (x=x1;x≤x2;x++)
{
        drawpixel(x,int(y+0.5),color);
        //pDC->SetPixel(x,int(y+0.5),RGB(0,0,255));
        y=y+k;
}
```

- 例：画直线段p(0,0)--P1(5,2)

x	int(y+0.5)	y+0.5
0	0	0
1	0	0.4+0.5
2	1	0.8+0.5
3	1	1.2+0.5
4	2	1.6+0.5
5	2	2.0+0.5



- 其他情况:
  - 在｜k｜>1时将X、Y的值互换一下，Y每增加1，X相应增加1／k:
  - $x_{i+1}=x_i+1/k$
  - $y_{i+1}=y_i+1$

- **特点:**

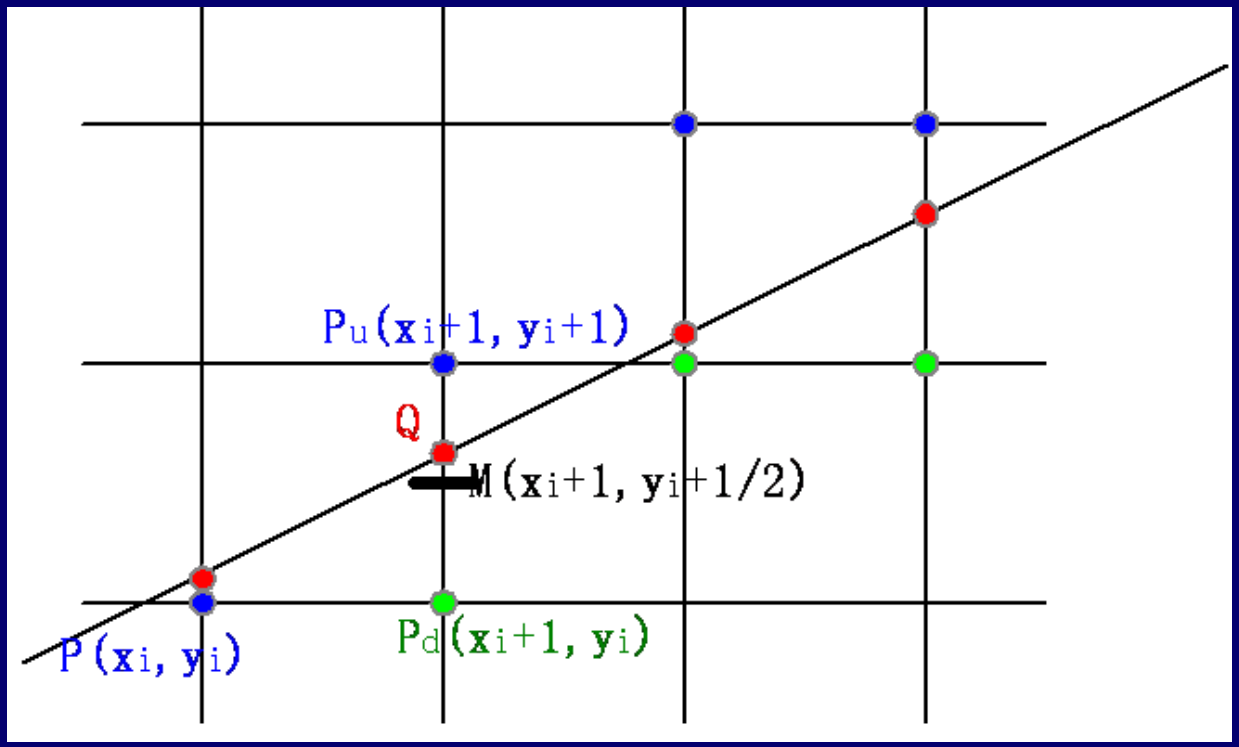
  直观、易实现

  不利于用硬件实现

5.1.2 中点Bresenham算法

基本原理：

假定0≤k≤1，x是最大位移方向



要判断下一步画P_u还是P_d，只需要判断M点在线上还是线下.

直线的方程

$$F(x,y)=y-kx-b=0\ ,\text{其中}k=\frac{\Delta y}{\Delta x}=\frac{y_1-y_0}{x_1-x_0}$$

如何判断一个点在直线下方还是上方：

- 对于直线上的点，F(x,y)=0；
- 对于直线上方的点，F(x,y)>0；
- 对于直线下方的点，F(x,y)<0。

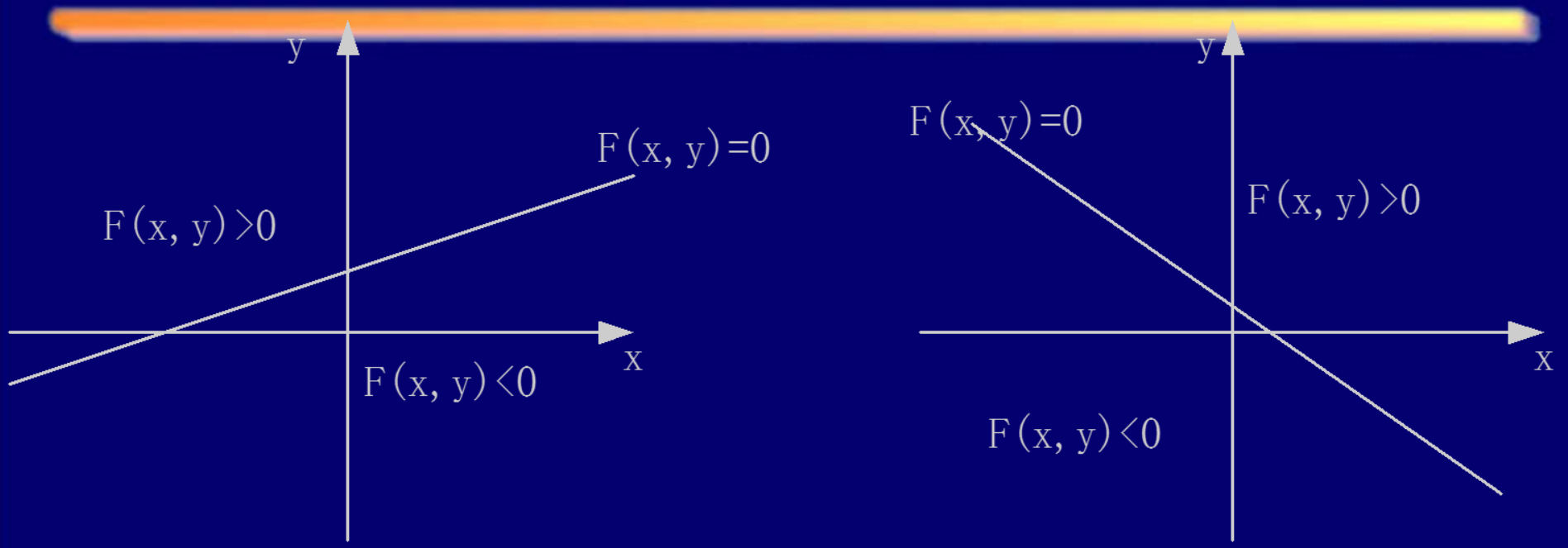


图5-4　直线将平面分为三个区域

判别式：

$$d_i = F(x_{iM}, y_{iM}) = F(x_i + 1, y_i + 0.5) = y_i + 0.5 - k(x_i + 1) - b$$

则有：

$$y_{i+1} = \begin{cases} y_i + 1 & (d_i < 0) \\ y_i & (d_i \geq 0) \end{cases}$$

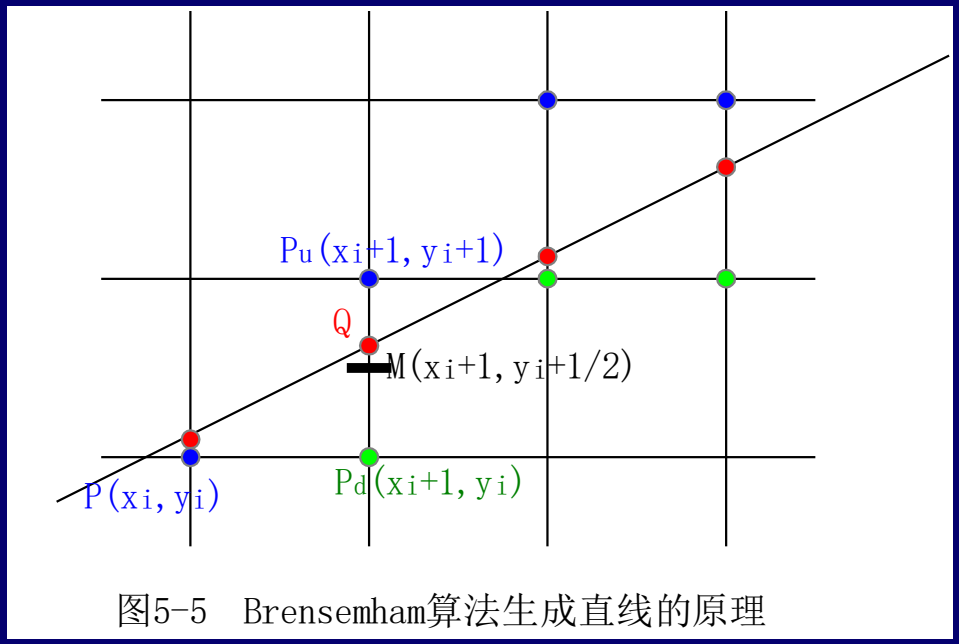
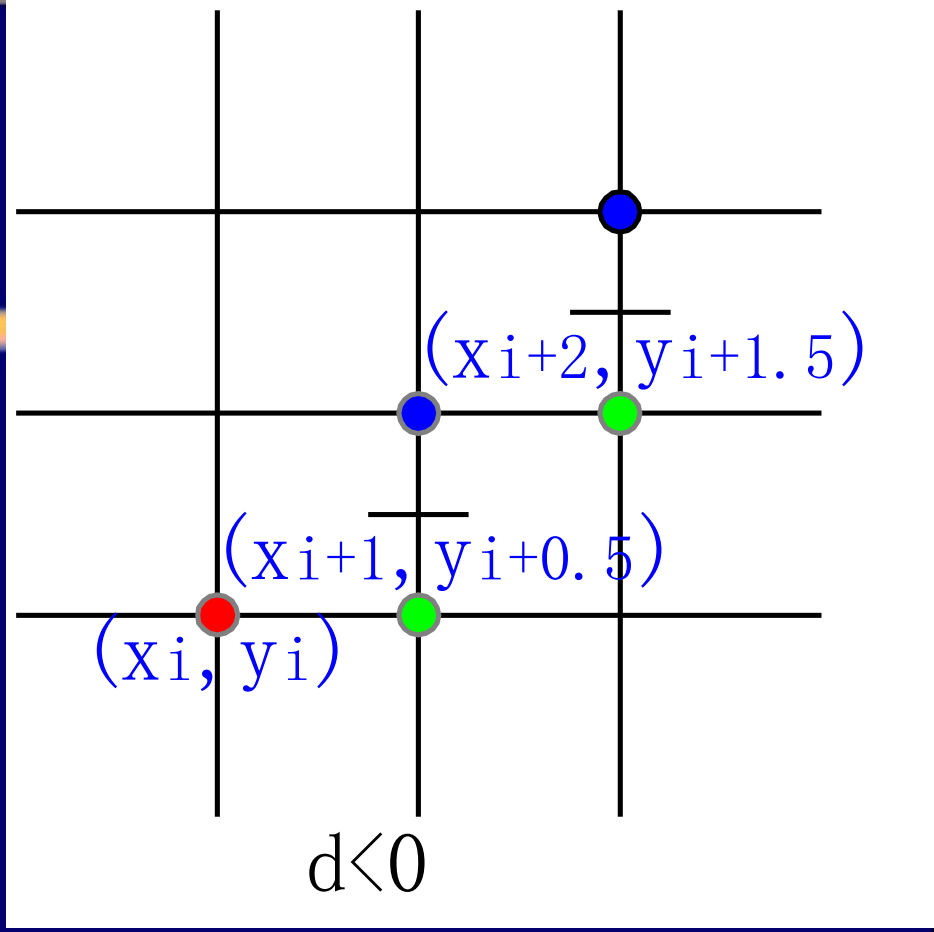


图5-5　Brensemham算法生成直线的原理

递推公式:

$d_i<0$:

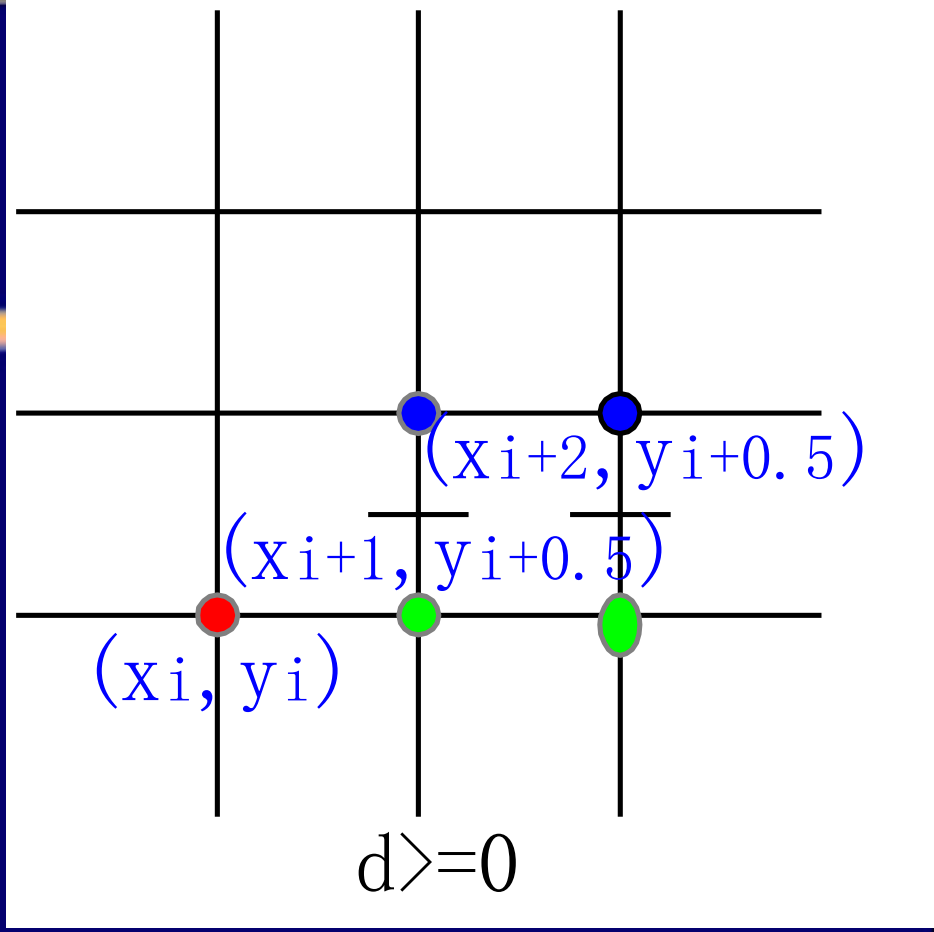
$$
\begin{aligned}
d_{i+1} &= F(x_i+2, y_i+1.5) \\
&= y_i+1.5-k(x_i+2)-b \\
&= y_i+1.5-k(x_i+1)-b-k \\
&= d_i+1-k
\end{aligned}
$$



误差项的递推

di≥0：

$$
\begin{aligned}
d_{i+1} &= F(x_i + 2, y_i + 0.5) \\
&= y_i + 0.5 - k(x_i + 2) - b \\
&= y_i + 0.5 - k(x_i + 1) - b - k \\
&= d_i - k
\end{aligned}
$$



初始值d0的计算

$$
\begin{aligned}
d_0 &= F(x_0 + 1, y_0 + 0.5) \\
&= y_0 + 0.5 - k(x_0 + 1) - b \\
&= y_0 - kx_0 - b - k + 0.5 \\
&= 0.5 - k
\end{aligned}
$$

0≤k≤1时Bresenham算法的算法步骤为：

1.输入直线的两端点$P_0(x_0,y_0)$和$P_1(x_1,y_1)$。

2.计算初始值△x、△y、d=0.5-k、$x=x_0$、$y=y_0$；

3.绘制点(x,y)。判断d的符号；

若d<0，则(x,y)更新为(x+1,y+1)，d更新为d+1-k；

否则(x,y)更新为(x+1,y)，d更新为d-k。

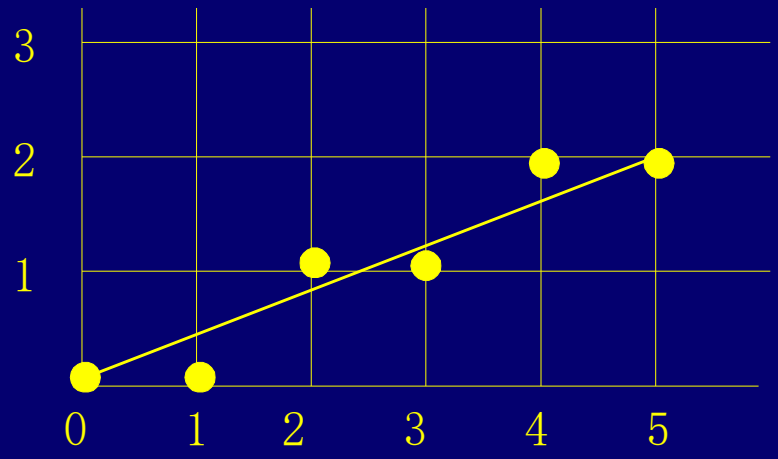
4.当直线没有画完时，重复步骤3。否则结束。

$d_0=0.5-k$, $\begin{cases} x_{i+1}=x_i+1 \\ y_{i+1}=\begin{cases} y_i+1 & (d_i<0) \\ y_i & (d_i\geq 0) \end{cases} \end{cases}$ $\begin{cases} d_{i+1}=d_i-k,(d_i\geq 0) \\ d_{i+1}=d_i+1-k,(d_i<0) \end{cases}$

例：$P_0(0,0)$, $P_1(5,2)$

k=dy/dx=0.4

序号	x	y	d
0	0	0	
1	1	0	0.5-0.4=0.1
2	2	1	0.1-0.4=-0.3
3	3	1	-0.3+1-0.4=0.3
4	4	2	0.3-0.4=-0.1
5	5	2	-0.1+1-0.4=0.5



思考

- 还有什么方法可以提高算法的效率?

改进1：令$e_0=d_0-0.5$

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (e_i > 0) \\ y_i & (e_i \le 0) \end{cases} \end{cases}$$

- $e_0=-k$，
- 每走一步有$e_{i+1}=e_i+k$。
- if (e>0) then e=e-1

改进2： 用2d△x代替d

1.输入直线的两端点$P_0(x_0,y_0)$和$P_1(x_1,y_1)$。

2.计算初始值△x、△y、**d=△x-2△y**、$x=x_0$、$y=y_0$。

3.绘制点(x,y)。判断d的符号。

若d<0，则(x,y)更新为(x+1,y+1)，d更新为

d+2△x-2△y;

否则(x,y)更新为(x+1,y), d更新为**d-2△y**。

4.当直线没有画完时，重复步骤3。否则结束。

$e_0=-k$,

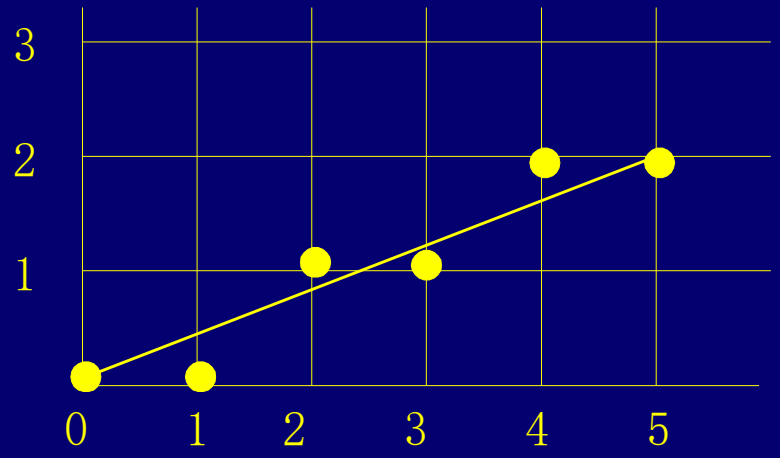
$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (e_i > 0) \\ y_i & (e_i \le 0) \end{cases} \end{cases}$$

$$\begin{cases} e_{i+1} = e_i - k, (e_i > 0) \\ e_{i+1} = e_i + 1 - k, (e_i \le 0) \end{cases}$$

例：$P_0(0, 0)$, $P_1(5,2)$

k=dy/dx=0.4

序号	x	y	e
0	0	0	
1	1	0	-0.4
2	2	1	0.2
3	3	1	-0.2
4	4	2	0.4
5	5	2	0.0



两种画线算法的比较

4.2 圆的扫描转换

要解决的问题：

圆的光栅化扫描转换算法设计

图形的扫描转换： 在光栅显示器等数字设备上确定一个最佳逼近于图形的象素集的过程。

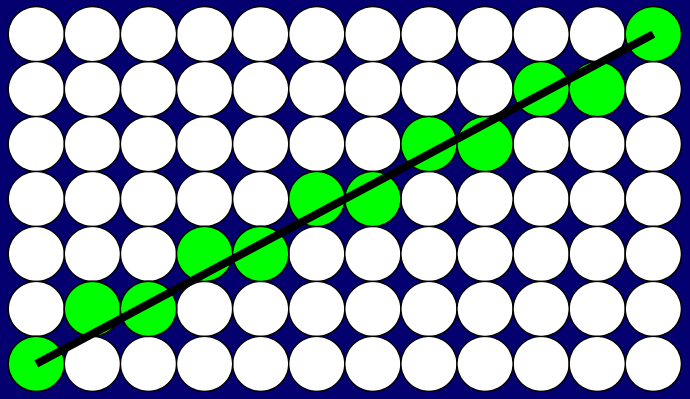
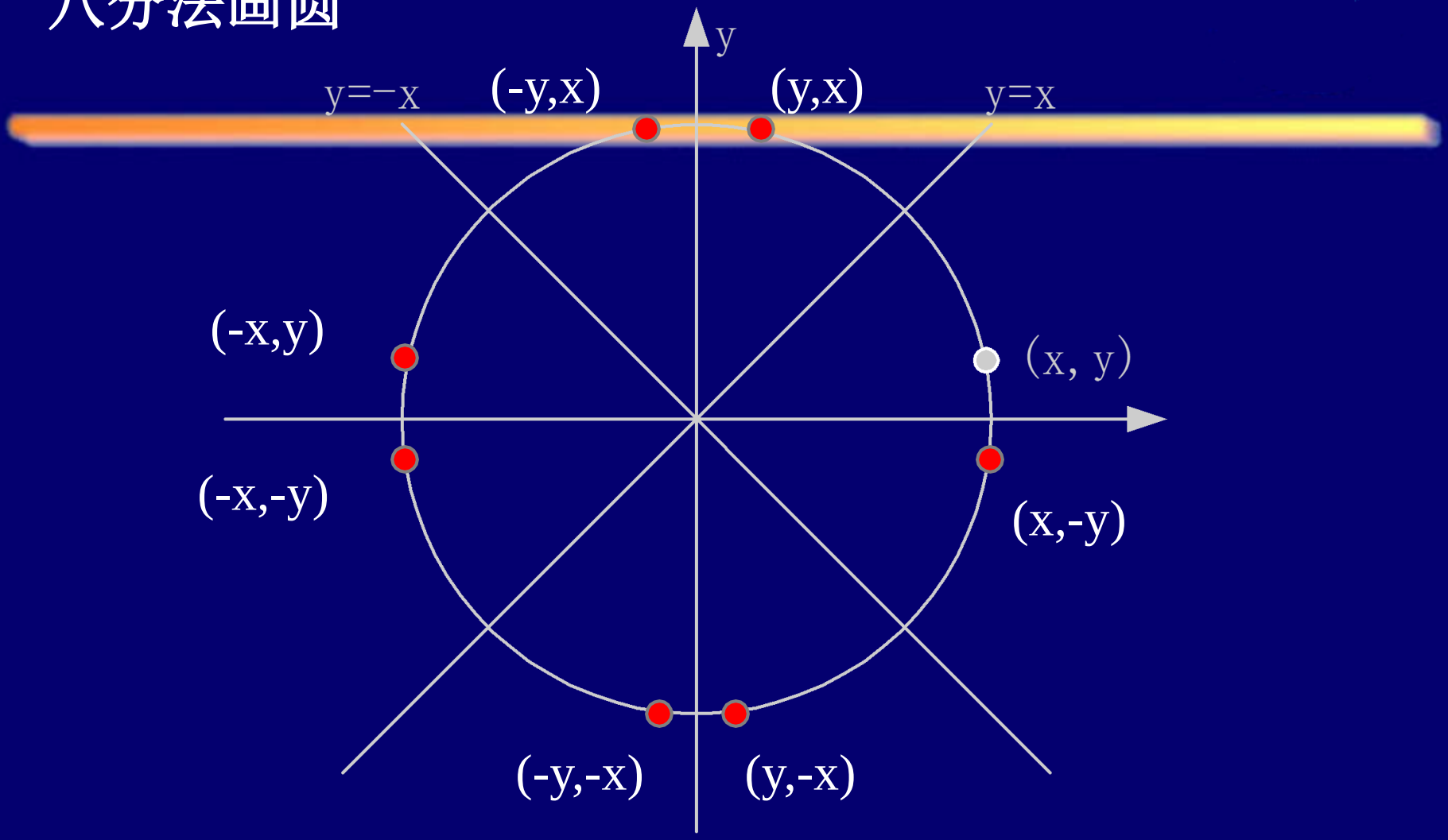


图4-1　用一系列的象素点来逼近直线

八分法画圆



解决问题：

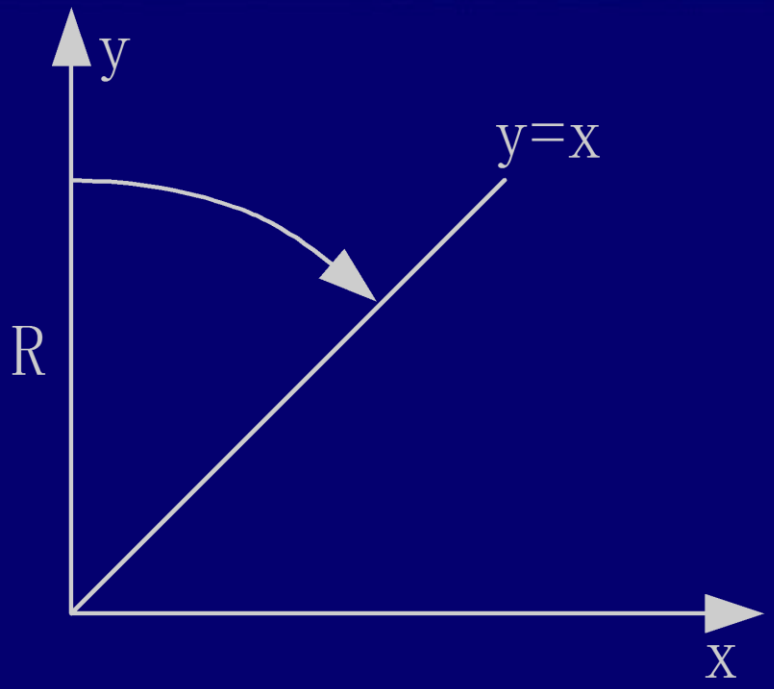


图5-10　1/8圆弧

4.2.1 简单方程产生圆弧

算法原理：利用其函数方程，直接离散计算

圆的函数方程为：$x^2 + y^2 = R^2$

$$x_{i+1} = x_i + 1 \qquad x \in \left[0, R/\sqrt{2}\right]$$

$$y_{i+1} = round(\sqrt{R^2 - x_{i+1}^2})$$

圆的极坐标方程为：

$$x = R\cos\theta$$

$$y = R\sin\theta$$

$\theta_{i+1} = \theta_i + \Delta\theta$ （$\Delta\theta$为一固定角度步长）

$$x_{i+1} = round(R\cos\theta_{i+1})$$

$$y_{i+1} = round(R\sin\theta_{i+1})$$

4.2.2 圆弧的Bresenham算法

待绘制的圆弧：

圆心在原点，半径为R的第一象限上的一段圆弧。且取（0，R）为起点，按顺时针方向绘制该1/8圆弧。

圆的方程： $F(x,y)=x^2+y^2-R^2$。

则有：

对于圆上的点，有$F(x,y)=0$；

对于圆外的点，$F(x,y)>0$；

对于圆内的点，$F(x,y)<0$。

基本原理

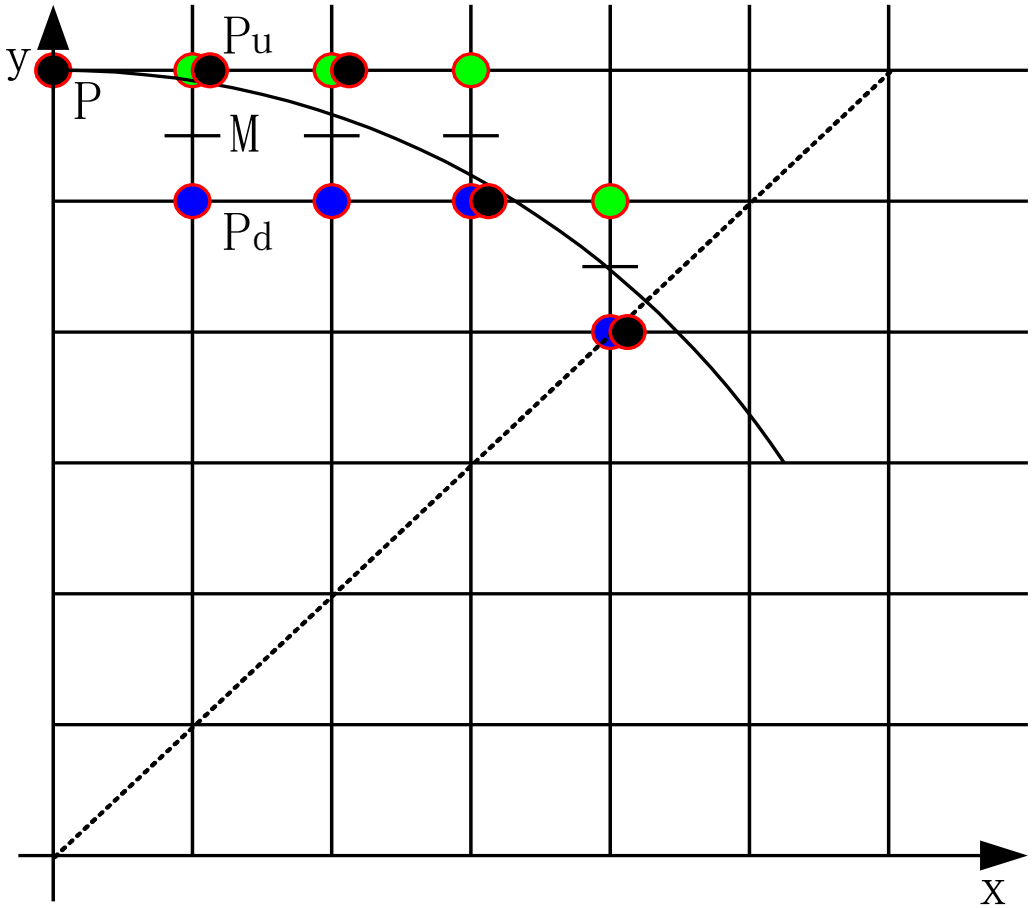


图4-3　中点Bresenham画圆的原理

M的坐标为：$M(x_i+1, y_i-0.5)$

- 当$F(x_M, y_M)<0$时，取$P_u(x_i+1, y_i)$
- 当$F(x_M, y_M)>0$时，取$P_d(x_i+1, y_i-1)$
- 当$F(x_M, y_M)=0$时，约定取P_u。

构造判别式：

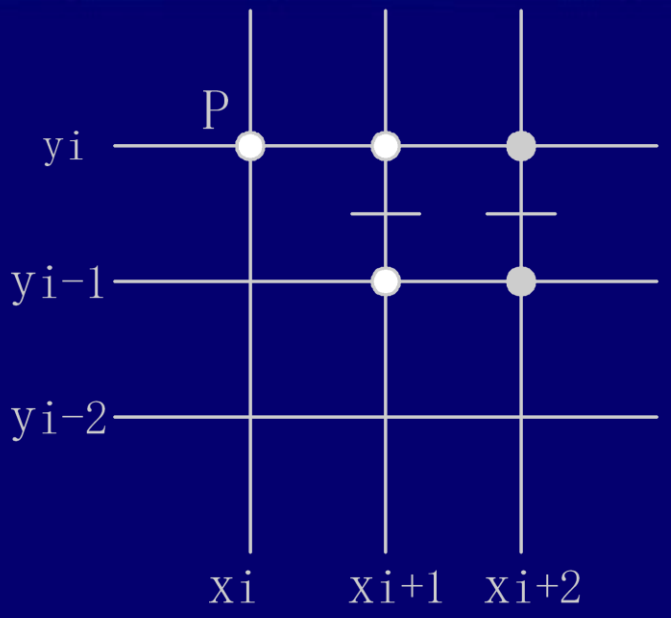
$$d_i = F(x_M, y_M) = F(x_i+1, y_i-0.5) = (x_i+1)^2 + (y_i-0.5)^2 - R^2$$

- 当$d_i \le 0$时，下一点取$P_u(x_i+1, y_i)$；
- 当$d_i > 0$时，下一点取$P_d(x_i+1, y_i-1)$。

误差项的递推

$d_i \le 0$：

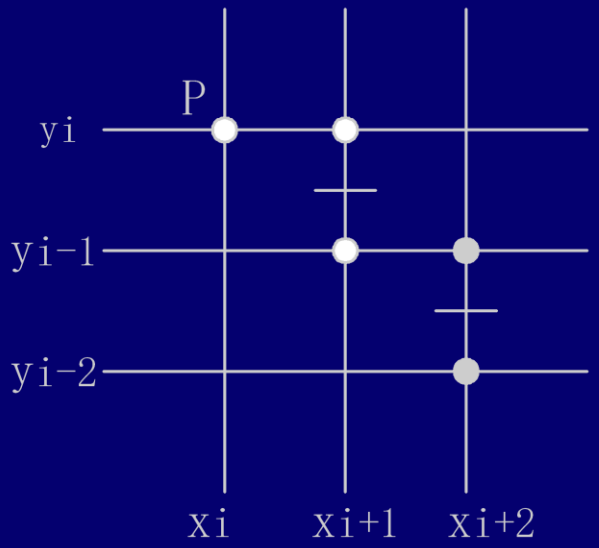
$$d_{i+1} = F(x_i + 2, y_i - 0.5)$$
$$= (x_i + 2)^2 + (y_i - 0.5)^2 - R^2$$
$$= (x_i + 1)^2 + (y_i - 0.5)^2 - R^2 + 2x_i + 3$$
$$= d_i + 2x_i + 3$$



(a)　di<=0的情况

$d_i>0$：

$$d_{i+1}=F(x_i+2,y_i-1.5)$$
$$=(x_i+2)^2+(y_i-1.5)^2-R^2$$
$$=(x_i+1)^2+(y_i-0.5)^2-R^2+(2x_i+3)+(-2y_i+2)$$
$$=d_i+2(x_i-y_i)+5$$



(b)　di>0的情况

判别式的初始值：

$$
\begin{aligned}
d_0 &= F(1, R-0.5) \\
&= 1+(R-0.5)^2 - R^2 \\
&= 1.25 - R
\end{aligned}
$$

循环何时结束？

$$x > y$$

圆弧的Bresenham算法:

判别式:

$$d_i = F(x_M, y_M) = (x_i+1)^2 + (y_i - 0.5)^2 - R^2$$

$$d_0 = 1.25 - R$$

当 $d_i <= 0$ ：

$$\begin{cases} x_{i+1} = x_i + 1, \quad y_{i+1} = y_i \\ d_{i+1} = d_i + 2x_i + 3 \end{cases}$$

当 $d_i > 0$ ：

$$\begin{cases} x_{i+1} = x_i + 1, \quad y_{i+1} = y_i - 1 \\ d_{i+1} = d_i + 2(x_i - y_i) + 5 \end{cases}$$

算法步骤：

1.输入圆的半径R。

2.计算初始值d=1.25-R、x=0、y=R。

3.绘制点(x,y)及其在八分圆中的另外七个对称点。

4.判断d的符号。若d≤0，则先将d更新为d+2x+3，再将(x,y)更新为(x+1,y)；否则先将d更新为d+2(x-y)+5，再将(x,y)更新为(x+1,y-1)。

5.当x<y时，重复步骤3和4。否则结束。

源程序

```
BresenhamCircle(int r, color)
        {
                int x,y,delta, d1, d2, dir
                x=0; y=r;
                delta = 2*(1-r)
                while(y>=0){
                    drawpixel(x,y,color);
                     if(delta < 0){
                        d1 = 2* (delta + y) -1;
                         if(d1 <=0) dir  = 1;
                         else            dir = 2;
                     }
                     else if( delta > 0){
                         d2 = 2*(delta-x)-1;
                         if(d2 <= 0) dir = 2;
                         else            dir = 3;
                     }
                     else       dir = 2;
```

```
switch(dir){
        case 1:
                x++;
                delta+=2*x + 1;
                 break;
        case 2:
                x++; Y--;
                delta+=2*(x-y+1) + 1;
                 break;
        case 3:
                y--;
                delta+=-2*y + 1;
                 break;
     }/*end of switch*/
   }/*end of while*/
}/*end of BresenhamCircle*/
```

程序

思考：

- 如何改进该算法来提高算法效率？
- 怎样设计椭圆的Bresenham算法？
- 圆心不在原点时，圆的扫描转换？

改进的Bresenham算法

改进：用d-0.25代替d

算法步骤：

1.输入圆的半径R。

2.计算初始值**d=1-R**、x=0、y=R。

3.绘制点(x,y)及其在八分圆中的另外七个对称点。

4.判断d的符号。若**d≤0**，则先将d更新为d+2x+3，再将(x,y)更新为(x+1,y)；否则先将d更新为d+2(x-y)+5，再将(x,y)更新为(x+1,y-1)。

5.当x<y时，重复步骤3和4。否则结束

二维几何图形扫描转换算法的一般方法：

(1) 四舍五入逼近的方法

(2) 递推判别式构造法

- 构造判别式；
- 构造判别式的递推公式；
- 计算判别式的初值。
- 进行递推逐步得到应该画的像素点

- 实践作业1：
  - 自己画直线或圆
    - 基本要求：给定坐标画出直线或圆，方法不限，可以用MFC，C#，也可以采用今天学的直线或圆的算法。
    - 有能力的同学可以自己扩展，包括实现鼠标拖动画线，颜色、粗细的改变，基本的动画等。