计算机图形学

第三章 直线、圆、椭圆生成算法

颜 波

复旦大学计算机科学技术学院
byan@fudan.edu.cn

图形的扫描转换（光栅化）：确定一个像素集合，用于显示一个图形的过程。步骤如下：

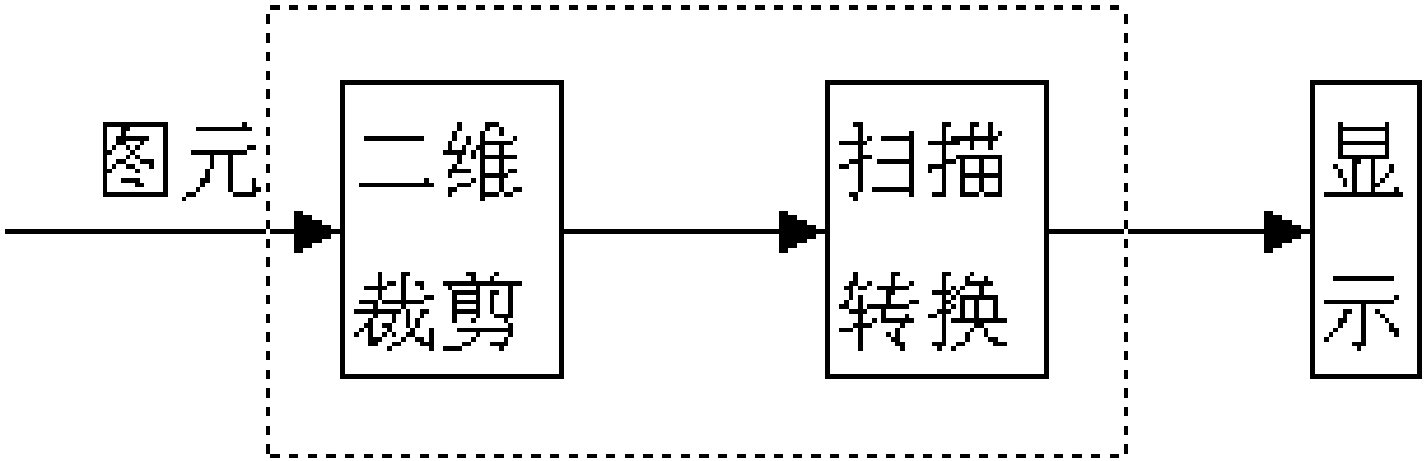
1、确定有关像素

2、用图形的颜色或其它属性，对像素进行写操作。

- 对一维图形，不考虑线宽，则用一个像素宽的直线来显示图形。
- 二维图形的光栅化，即区域的填充：确定像素集，填色或图案。
- 任何图形的光栅化，必须显示在一个窗口内，否则不予显示。即确定一个图形的哪些部分在窗口内，哪些在窗口外，即裁剪。

图形显示前需要：扫描转换+裁剪

- 裁剪---〉扫描转换：最常用，节约计算时间。
- 扫描转换---〉裁剪：算法简单；



本章概述

- 扫描转换直线段
  - DDA算法
  - 中点画线法
  - Bresenham画线算法

- 圆弧、椭圆弧扫描转换
  - 中点算法
  - 内接正多边形迫近法
  - 等面积正多边形逼近法
  - 生成圆弧的正负法

直线段的扫描转换算法

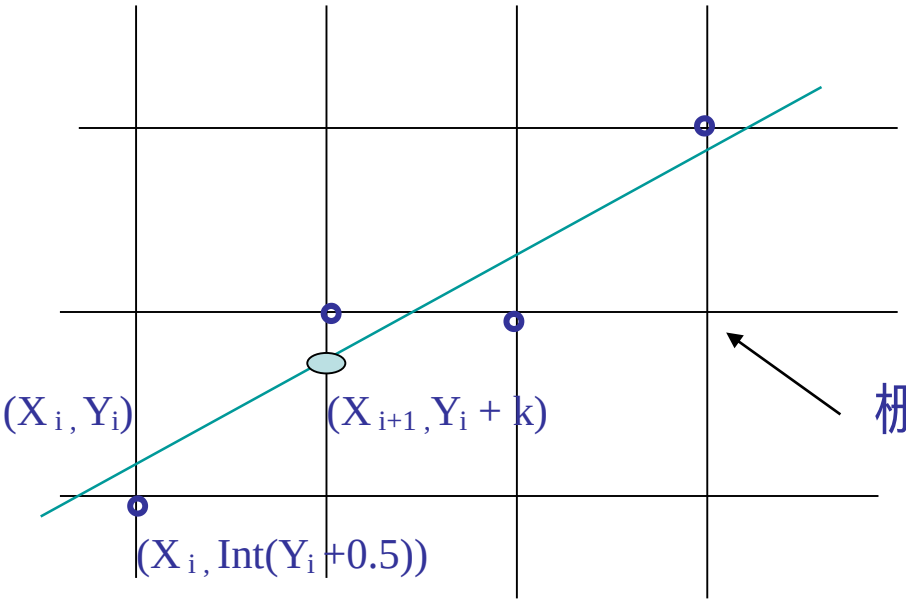
- **直线的扫描转换:**

  确定最佳逼近于该直线的一组象素，并且按扫描线顺序，对这些象素进行写操作。

- **三个常用算法**:
  - 数值微分法（DDA）
  - 中点画线法
  - Bresenham算法

数值微分(DDA)法

假定直线的起点、终点分别为：（x0,y0),(x1,y1)，且都为整数。



数值微分(DDA)法

- **基本思想**

已知过端点P0 (x0, y0), P1(x1, y1)的直线段L

y=kx+b

直线斜率为 $k = \dfrac{y_1 - y_0}{x_1 - x_0}$

令 $x = x_0 \rightarrow x_1; x = x + stepx$

$$y = kx + b$$

$$\therefore \left(x, round(y)\right)$$

这种方法直观，但效率太低，因为每一步需要一次浮点乘法和一次舍入运算。

数值微分(DDA)法

计算 $y_{i+1} = kx_{i+1} + b$

$= kx_i + b + k\Delta x$

$= y_i + k\Delta x$

当 $\Delta x = 1$; $y_{i+1} = y_i + k$

- 即：当x每递增1，y递增k(即直线斜率)；
- 注意上述分析的算法仅适用于$|k| \leq 1$的情形。在这种情况下，x每增加1，y最多增加1。
- 当 $|k| > 1$时，必须把x，y地位互换

数值微分(DDA)法

- **增量算法**：在一个迭代算法中，如果每一步的x、y值是用前一步的值加上一个增量来获得，则称为增量算法。

- DDA算法就是一个增量算法。

数值微分(DDA)法

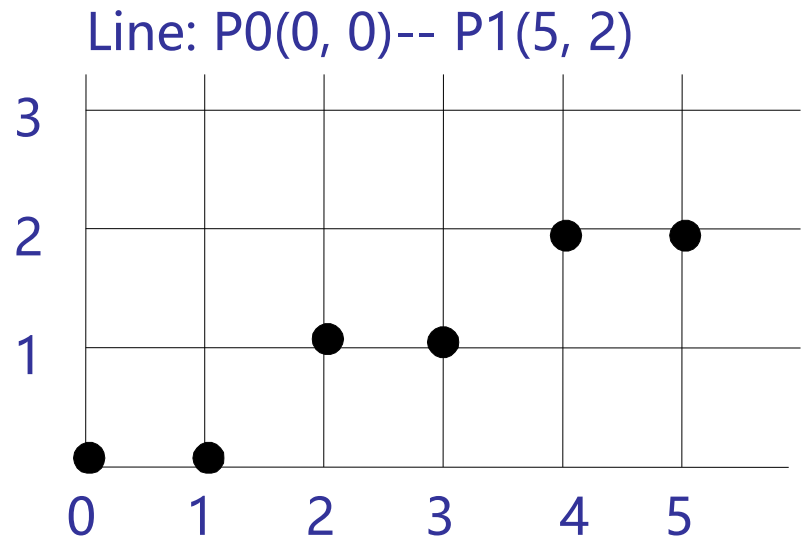
```
void DDALine(int x0,int y0,int x1,int y1,int color)
{  int x;
   float dx, dy, y, k;

   dx = x1-x0, dy=y1-y0;
   k=dy/dx, y=y0;
   for (x=x0; x≤x1, x++)
   {  drawpixel (x, int(y+0.5), color);
      y=y+k;
   }
}
```

数值微分(DDA)法

- 例：画直线段$P_0(0,0)$--$P_1(5,2)$，k=0.4

x	int(y+0.5)	y+0.5
0	0	0+0.5
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



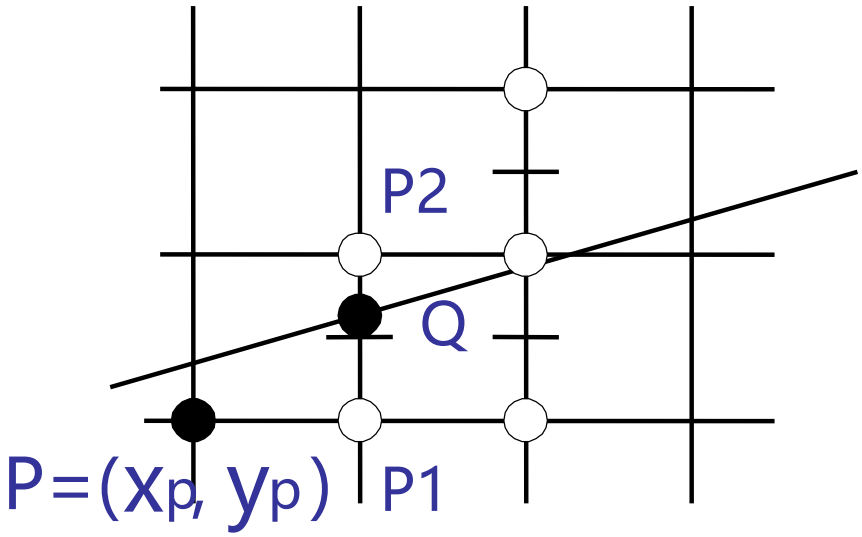
数值微分(DDA)法

- 缺点：

在此算法中，y、k必须是float，且每一步都必须对y进行舍入取整，不利于硬件实现。

中点画线法

- 原理：



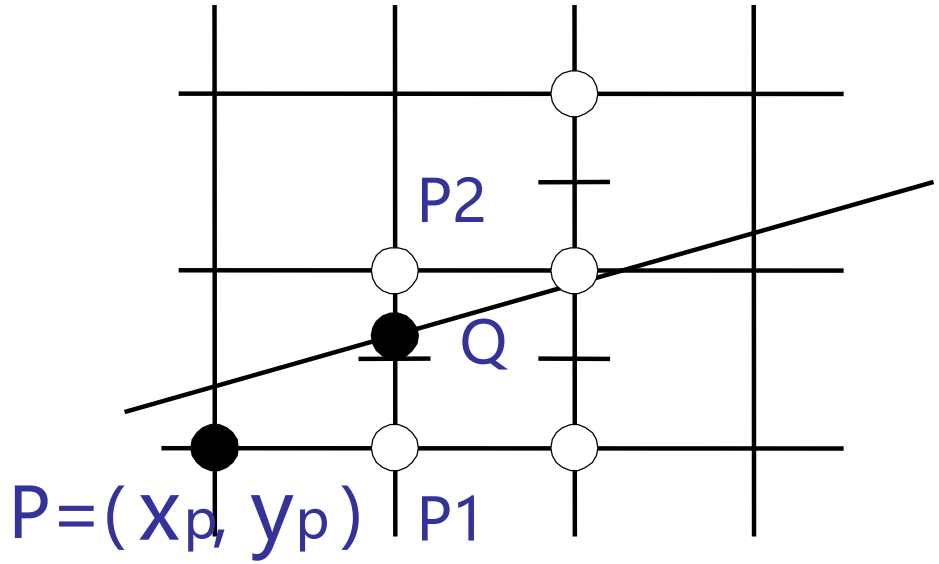
设M为中点，Q为交点

假定直线斜率0<K<1，且已确定点亮象素点P（X_p ，Y_p ），则下一个与直线最接近的像素只能是P1点或P2点。

现需确定下一个点亮的象素。

中点画线法

- 当M在Q的下方-> P_2离直线更近 -> 取P_2。
- M在Q的上方-> P_1离直线更近 -> 取P_1
- M与Q重合， P_1、P_2任取一点。



- 问题：如何判断M与Q点的关系?

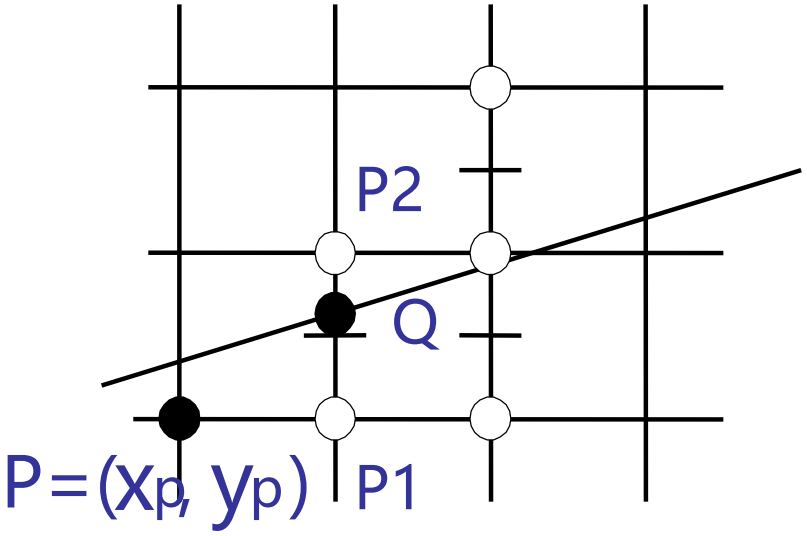
中点画线法

假设直线方程为： $ax+by+c=0$

其中$a=y_0-y_1$, $b=x_1-x_0$, $c=x_0y_1-x_1y_0$

由常识知：

$$\begin{cases} F(x,y)=0 & \text{点在直线上面} \\ F(x,y)>0 & \text{点在直线上方} \\ F(x,y)<0 & \text{点在直线下方} \end{cases}$$



∴欲判断中点M点是在Q点上方还是在Q点下方，只需把M代入$F(x,y)$，并检查它的符号。

中点画线法

构造判别式：

$d=F(M)=F(x_p+1,y_p+0.5)$

$=a(x_p+1)+b(y_p+0.5)+c$

当$d<0$，M在直线(Q点)下方，

取右上方P_2；

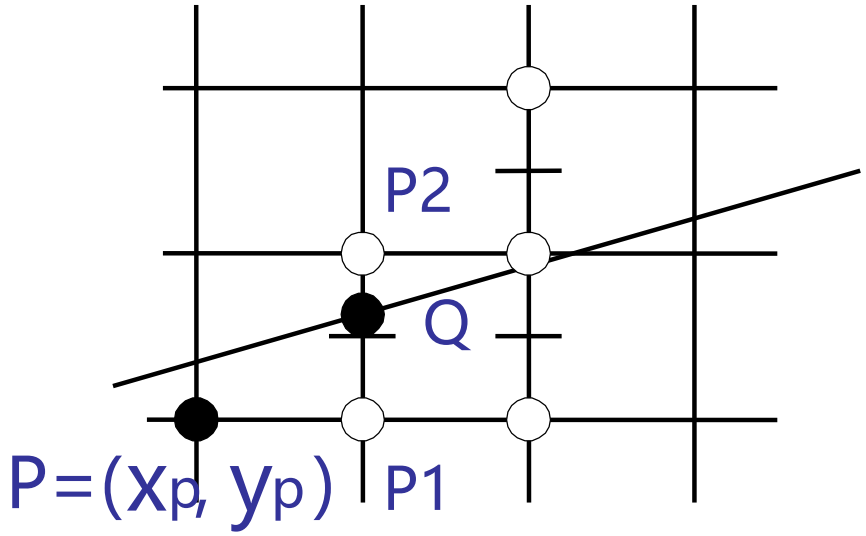
当$d>0$，M在直线(Q点)上方，

取右方P_1；

当$d=0$，选P_1或P_2均可，约定

取P_1；

能否采用增量算法呢?



中点画线法

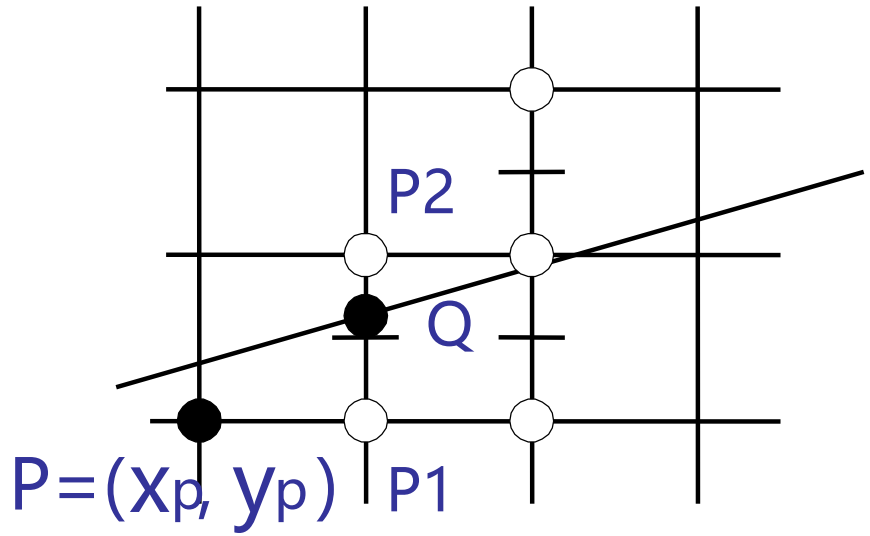
若d≥0 ->M在直线上方->取P1;

此时再下一个象素的判别式为

$d_1=F(x_p+2, y_p+0.5)=a(x_p+2)+b(y_p+0.5)+c$

$= a(x_p+1)+b(y_p+0.5)+c+a = d+a;$

增量为a



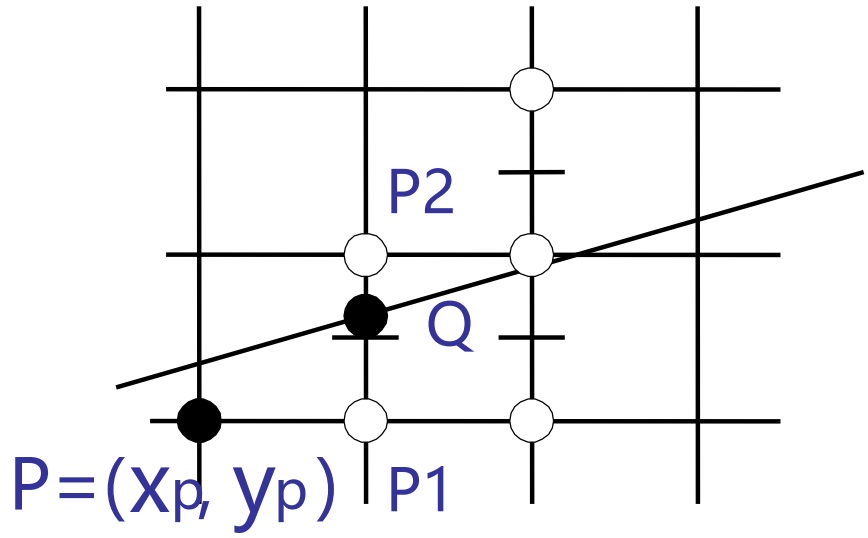
中点画线法

若d<0->M在直线下方->取P2;

此时再下一个象素的判别式为

$$d_2 = F(x_p+2, y_p+1.5) = a(x_p+2)+b(y_p+1.5)+c$$
$$= a(x_p+1)+b(y_p+0.5)+c+a+b = d+a+b$$

增量为a + b



中点画线法

画线从(x_0, y_0)开始，d的初值

$$d_0=F(x_0+1, y_0+0.5)= a(x_0+1)+b(y_0+0.5)+c$$
$$= F(x_0, y_0)+a+0.5b = a+0.5b$$

由于只用d 的符号作判断，为了只包含整数运算，可以用**2d代替d来摆脱小数**，提高效率。

中点画线法

```
void Midpoint Line (int x0,int y0,int x1, int y1,int color)
{   int a, b, d1, d2, d, x, y;
    a=y0-y1, b=x1-x0, d=2*a+b;
    d1=2*a, d2=2* (a+b);        /*不同的增量*/
    x=x0, y=y0;
    drawpixel(x, y, color);
    while (x<x1)
    { if (d<0)       {x++; y++; d+=d2; }
     else       {x++; d+=d1;}
     drawpixel (x, y, color);
     }  /* while */
 } /* mid PointLine */
```

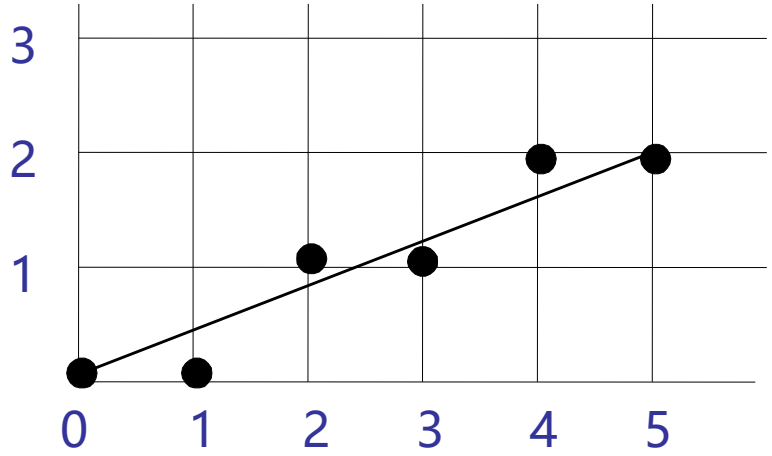
中点画线法

- 例：用中点画线法P0(0,0) P1(5,2)

a=y0-y1=-2　　b=x1-x0=5

d0=2a+b=1　　d1=2a=-4　　d2=2(a+b)=6

i	xi	yi	d
0	0	0	1
1	1	0	-3
2	2	1	3
3	3	1	-1
4	4	2	5



Bresenham画线算法

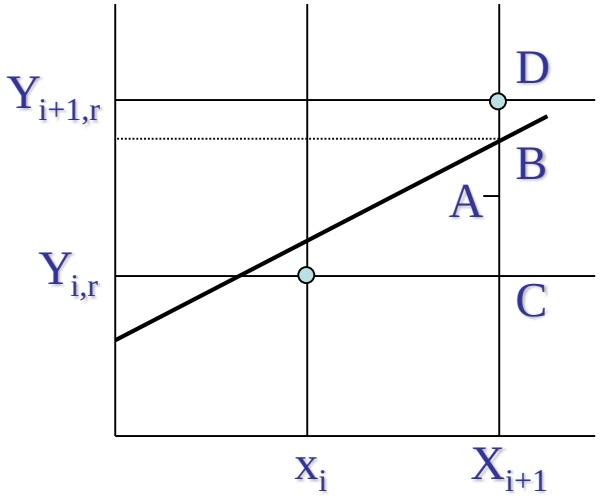
在直线生成的算法中Bresenham算法是最有效的算法之一。令 $k=\Delta y/\Delta x$，就$0\le k\le 1$的情况来说明Bresenham算法。由DDA算法可知：

$$y_{i+1}=y_i+k \quad (1)$$

由于k不一定是整数，由此式求出的y_i也不一定是整数，因此要用坐标为（x_i,y_{ir}）的象素来表示直线上的点，其中y_{ir}表示最靠近y_i的整数。

Bresenham画线算法

- 设图中x_i列上已用（x_i,y_{ir}）作为表示直线的点，又设B点是直线上的点，其坐标为（x_{i+1},y_{i+1}），显然下一个表示直线的点（x_{i+1},$y_{i+1,r}$）只能从图中的C或者D点中去选。
- 设A为CD边的中点。若B在A点上面则应取D点作为（x_{i+1},$y_{i+1,r}$），否则应取C点。



ε(x)的几何意义

为能确定B在A点上面或下面，令

$\varepsilon(x_{i+1})=y_{i+1}-y_{ir}-0.5$ (2) （**B到A的距离**）

若B在A的下面，则有$\varepsilon(x_{i+1})<0$，反之则$\varepsilon(x_{i+1})>0$。由图可知

若$\varepsilon(x_{i+1})\geq 0$， $y_{i+1,r}=y_{ir}+1$ (3)

若$\varepsilon(x_{i+1})\leq 0$， $y_{i+1,r}=y_{ir}$

Bresenham画线算法

由式（2）和式（3）可得到

$$\varepsilon(x_{i+2}) = y_{i+2} - y_{i+1,r} - 0.5$$
$$= y_{i+1} + k - y_{i+1,r} - 0.5 \quad (4)$$

$y_{i+1} - y_{ir} - 0.5 + k - 1$, 当$\varepsilon(x_{i+1}) \geq 0$ $y_{i+1,r} = y_{ir} + 1$

$y_{i+1} - y_{ir} - 0.5 + k$, 当$\varepsilon(x_{i+1}) \leq 0$ $y_{i+1,r} = y_{ir}$

$\varepsilon(x_{i+2}) = \varepsilon(x_{i+1}) + k - 1$, 当$\varepsilon(x_{i+1}) \geq 0$

$\varepsilon(x_{i+2}) = \varepsilon(x_{i+1}) + k$, 当$\varepsilon(x_{i+1}) \leq 0$

由式（1）和式（2）可得到

$$\varepsilon(x_2) = k - 0.5 \quad (5)$$

如何变成整数？？

***2dx**

Bresenham画线算法

Bresenham　画线算法程序//伪代码

```
void Bresenhamline (int x0,int y0,int x1, int y1,int color)
{ int x, y, dx, dy;  float k, e;
  dx = x1-x0;  dy = y1- y0; k=dy/dx;
  e=-0.5; x=x0; y=y0;
  for (i=0; i<=dx; i++)
  { Putpixel (x, y, color);
    x=x+1;  e=e+k;
    if (e>= 0)    { y++, e=e-1;}
  }
}
```

圆的扫描转换算法

下面仅以圆心在原点、半径R为整数的圆为例，讨论圆的生成算法。

假设圆的方程为：

$$X^2 + Y^2 = R^2$$

圆的扫描转换算法

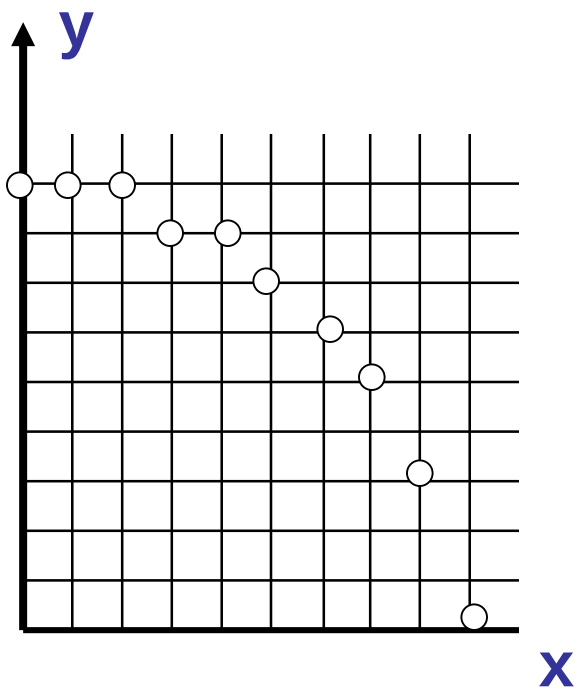
$X^2 + Y^2 = R^2$

$Y = \pm Sqrt(R^2 - X^2)$

在一定范围内，每给定一X值，可得一Y值。

当X取整数时，Y须取整。

缺点：浮点运算，开方，取整，Y不均匀。



角度DDA法

$$x = x_0 + R\cos\theta$$

$$y = y_0 + R\sin\theta$$

$$dx = -R\sin\theta d\theta$$

$$dy = R\cos\theta d\theta$$

$$x_{n+1} = x_n + dx$$

$$y_{n+1} = y_n + dy$$

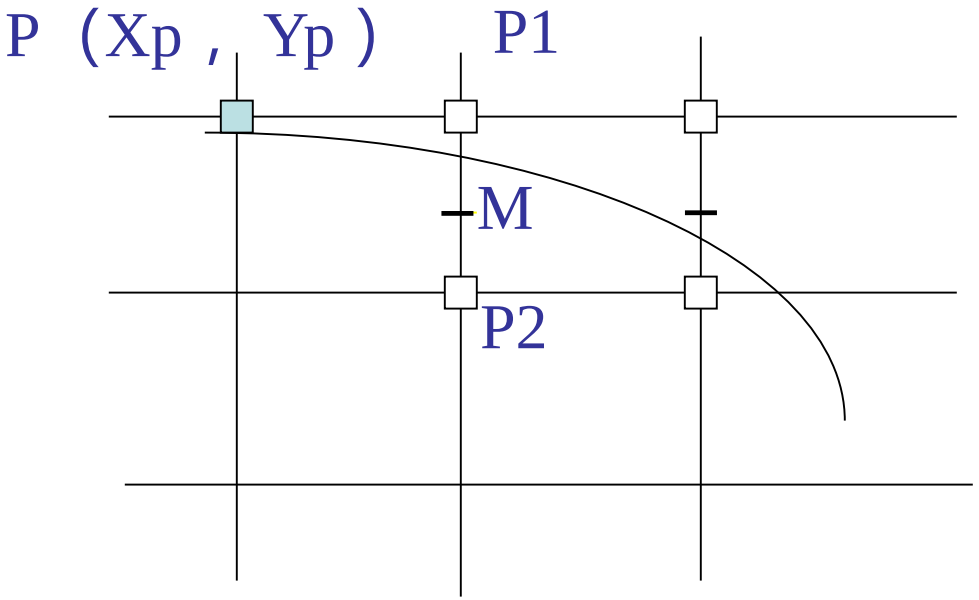
$$x_{n+1} = x_n + dx = x_n - R\sin\theta d\theta = x_n - (y_n - y_0)d\theta$$

$$y_{n+1} = y_n + dy = y_n + R\cos\theta d\theta = y_n + (x_n - x_0)d\theta$$

显然，确定x,y的初值及dθ值后，即可以增量方式获得圆周上的坐标，然后取整可得象素坐标。
但要采用浮点运算、乘法运算、取整运算。

中点画圆法

利用圆的对称性，只须讨论1/8圆。第二个8分圆



P为当前点亮象素，那么下一个点亮的象素可能是P1（Xp+1，Yp）或P2（Xp +1，Yp -1）。

中点画圆法

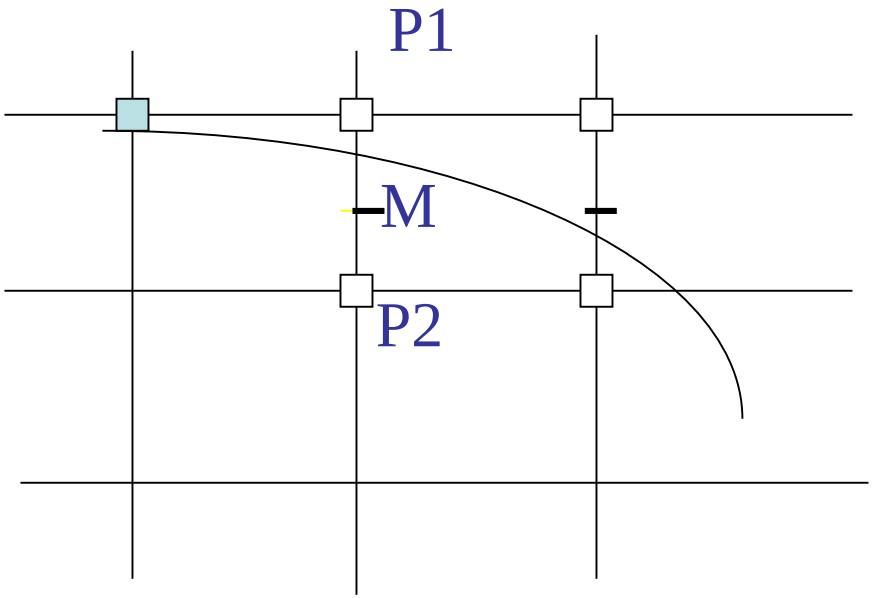
构造函数：$F(X, Y) = X^2 + Y^2 - R^2$；则

$F(X, Y) = 0$　(X, Y) 在圆上；

$F(X, Y) < 0$　(X, Y) 在圆内；

$F(X, Y) > 0$　(X, Y) 在圆外。

设M为P1、P2间的中点，$M=(X_p+1, Y_p-0.5)$



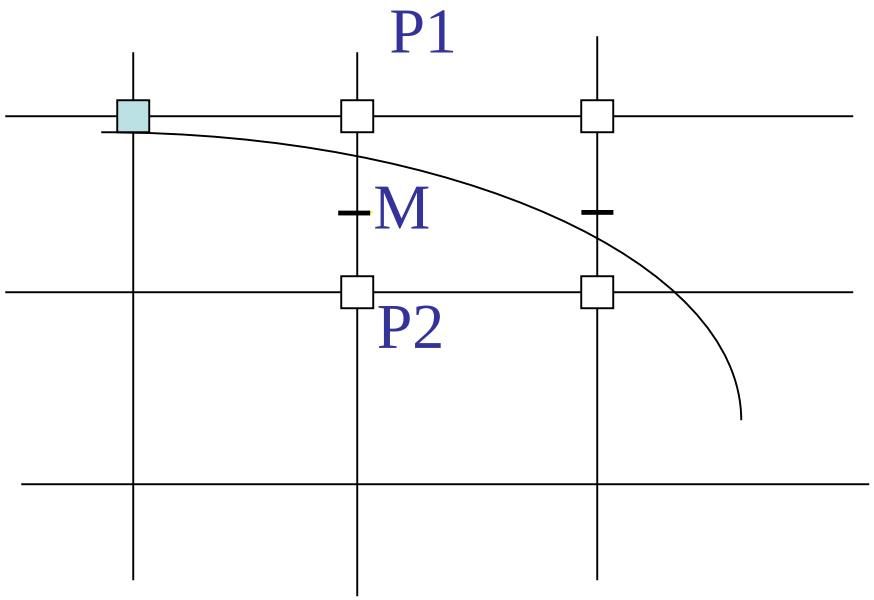
中点画圆法

有如下结论：

F（M）< 0 ->M在圆内-> 取P1

F（M）>= 0 ->M在圆外-> 取P2

为此，可采用如下判别式：

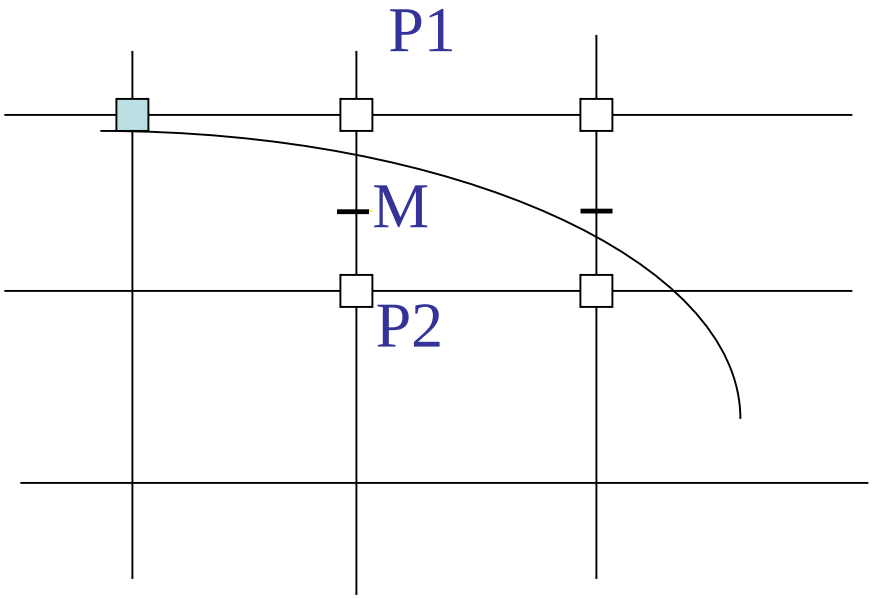


中点画圆法

$$d = F(M) = F(x_p + 1, y_p - 0.5) = (x_p + 1)^2 + (y_p - 0.5)^2 - R^2$$

若d<0, 则P1 为下一个象素，那么再下一个象素的判别式为

$$d1 = F(x_p + 2, y_p - 0.5) = (x_p + 2)^2 + (y_p - 0.5)^2 - R^2$$
$$= d + 2x_p + 3$$



即d 的增量为 $2x_p + 3$.

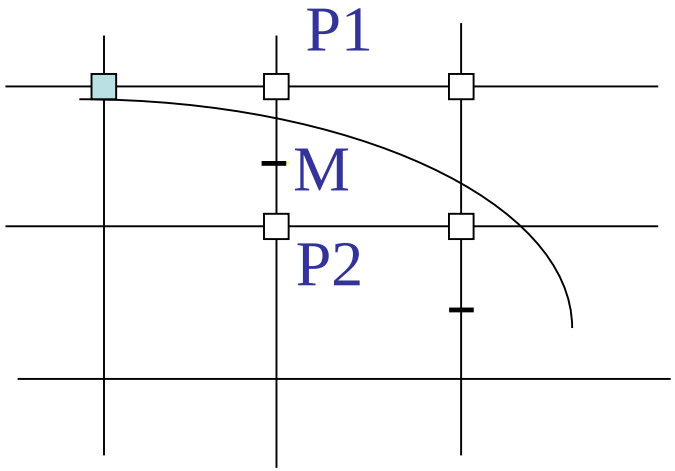
中点画圆法

若d>=0, 则P2 为下一个象素，那再下一个象素的判别式为：

$$d1 = F(x_p + 2, y_p - 1.5)$$
$$= (x_p + 2)^2 + (y_p - 1.5)^2 - R^2$$
$$= d + (2x_p + 3) + (-2y_p + 2)$$

即d 的增量为 $2(x_p - y_p) + 5$

d的初值:$d0 = F(1, R-0.5) = 1 + (R-0.5)^2 - R^2 = 1.25 - R$



如何变成整数？？

***4**

中点画圆法

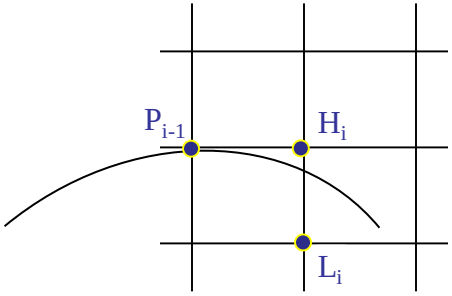
- 为了进一步提高算法的效率，可以将上面的算法中的浮点数改写成整数，将乘法运算改成加法运算，即仅用整数实现中点画圆法。

$$d0 = 1 - R$$

中点画圆法

```
MidpointCircle(int r, int color)
{
        int x,y;
        float d;
        x=0; y=r; d=1.25-r;
        drawpixel(x,y,color);
        while(x<y){
                if(d<0){ d+ = 2*x+3; x++ }
                else{d+ = 2*(x-y) + 5; x++;y--;}
                 }
}
```

Bresenham画圆算法



应取H_i还是取L_i

现在从A点开始向右下方逐点来寻找弧AB要用的点。

如图中点P_{i-1}是已选中的一个表示圆弧上的点，根据弧AB的走向，下一个点应该从H_i或者L_i中选择。显然应选离AB最近的点作为显示弧AB的点。

假设圆的半径为R，显然，

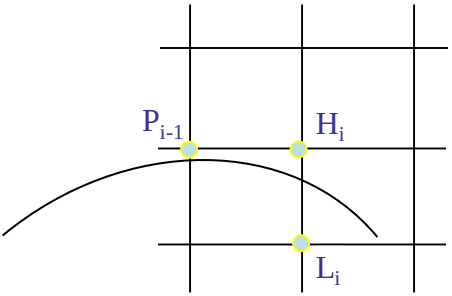
当$x_{hi}^2 + y_{hi}^2 - R^2 \geq R^2 - (x_{li}^2 + y_{li}^2)$时，应该取$L_i$。否则取$H_i$。

令$d_i = x_{hi}^2 + y_{hi}^2 + x_{li}^2 + y_{li}^2 - 2R^2$

显然，当$d_i \geq 0$ 时应该取L_i。

否则，取H_i。

Bresenham画圆算法



应取H_i还是取L_i

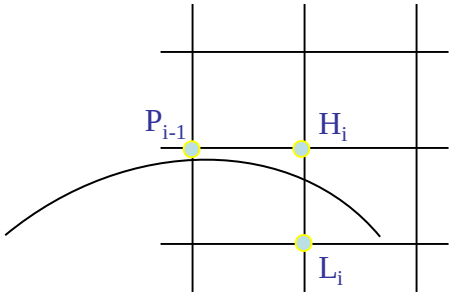
剩下的问题是如何快速的计算d_i。

设图中P_{i-1}的坐标为(x_{i-1},y_{i-1})，则H_i和L_i的坐标为(x_i,y_{i-1})和$(x_i,y_{i-1}-1)$

$$d_i = x_i^2 + y_{i-1}^2 + x_i^2 + (y_{i-1}-1)^2 - 2R^2$$
$$= 2x_i^2 + 2y_{i-1}^2 - 2y_{i-1} - 2R^2 + 1$$

$$d_{i+1} = (x_i + 1)^2 + y_i^2 + (x_i + 1)^2 + (y_i - 1)^2 - 2R^2$$
$$= 2x_i^2 + 4x_i + 2y_i^2 - 2y_i - 2R^2 + 3$$

Bresenham画圆算法



应取H_i还是取L_i

当$d_i<0$时 -> 取H_i -> $y_i=y_{i-1}$，
则$d_{i+1} = d_i + 4x_{i-1} + 6$

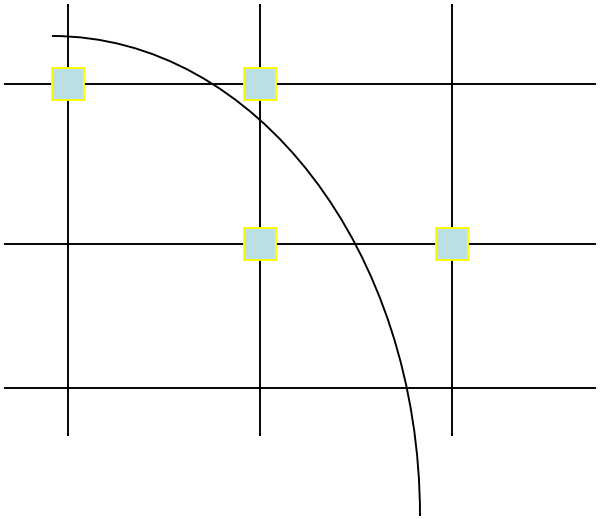
当$d_i \geq 0$时 -> 取L_i -> $y_i=y_{i-1}-1$，
则$d_{i+1} = d_i + 4(x_{i-1}-y_{i-1}) + 10$

易知 $x_0=0$，$y_0=R$，$x_1=x_0+1$
因此

$$d_0=1^2 + y_0^2 + 1^2 +(y_0 - 1)^2 - 2R^2$$
$$= 3 - 2y_0$$
$$= 3 - 2R$$

生成圆弧的正负法

原理：



设圆的方程为$F(x,y)=X^2 + Y^2 - R^2=0$;
假设求得P_i的坐标为(x_i,y_i);
则当P_i在圆内时-> $F(x_i,y_i)<0$ -> 向右-> 向圆外
P_i在圆外时-> $F(x_i,y_i)>0$ -> 向下-> 向圆内

生成圆弧的正负法

即求得P_i点后选择下一个象素点P_{i+1}的规则为：
当$F(x_i,y_i) \le 0$ 取$x_{i+1} = x_i+1$，$y_{i+1} = y_i$;
当$F(x_i,y_i) > 0$ 取$x_{i+1} = x_i$，$y_{i+1} = y_i - 1$;

- 这样用于表示圆弧的点均在圆弧附近，且使$F(x_i,y_i)$ 时正时负，故称正负法。

快速计算的关键是$F(x_i,y_i)$ 的计算，能否采用增量算法?

生成圆弧的正负法

若$F(x_i,y_i)$ 已知，计算$F(x_{i+1},y_{i+1})$ 可分两种情况：

1、$F(x_i,y_i)\leq 0$-> $x_{i+1} = x_i+1$， $y_{i+1} = y_i$;

-> $F(x_{i+1},y_{i+1})= (x_{i+1})^2 +(y_{i+1})^2 -R^2$

-> $= (x_i+1)^2+ y_i^2 -R^2 = F(x_i,y_i) +2x_i +1$

2、 $F(x_i,y_i) > 0$-> $x_{i+1} = x_i$， $y_{i+1} = y_i -1$;

-> $F(x_{i+1},y_{i+1})= (x_{i+1})^2 +(y_{i+1})^2 -R^2$

-> $= x_i^2+(y_i-1)^2-R^2 = F(x_i,y_i) - 2y_i +1$

3、初始值：略

生成圆弧的多边形逼近法

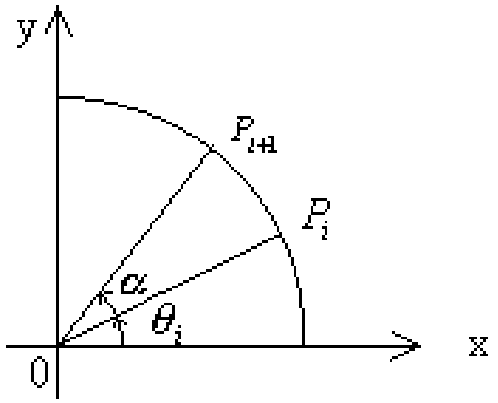
- 圆的内接正多边形迫近法
- 圆的等面积正多边形迫近法

圆的内接正多边形迫近法

- 思想：当一个正多边形的边数足够多时，该多边形可以和圆无限接近。即

$$\lim_{n\to\infty} (\text{n边内接多边形}) = \text{圆}$$

- 因此，在允许的误差范围内，可以用正多边形代替圆。
- 设内接正n边形的顶点为$P_i(x_i,y_i)$，P_i的幅角为θi，每一条边对应的圆心角为a，则有
  - $x_i = R\cos\theta i$
  - $y_i = R\sin\theta i$

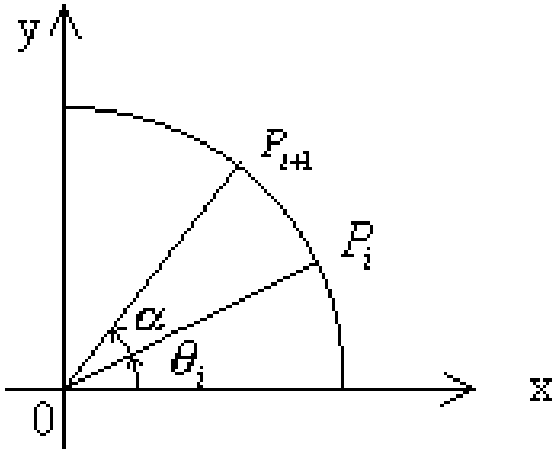


圆的内接正多边形逼近法

内接正n边形代替圆

计算多边形各顶点的递推公式

$$\begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix} = \begin{pmatrix} R\cos(a+\theta_i) \\ R\sin(a+\theta_i) \end{pmatrix}$$



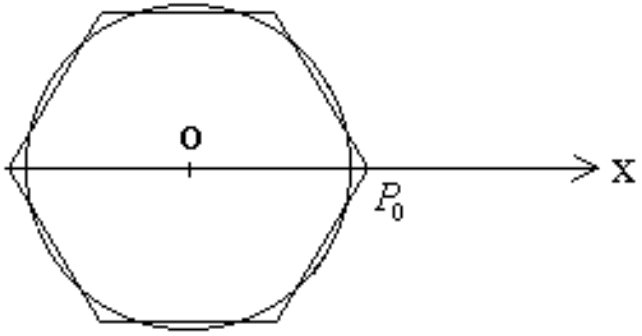
$$\begin{pmatrix} X_{i+1} \\ Y_{i+1} \end{pmatrix} = \begin{pmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \end{pmatrix}$$

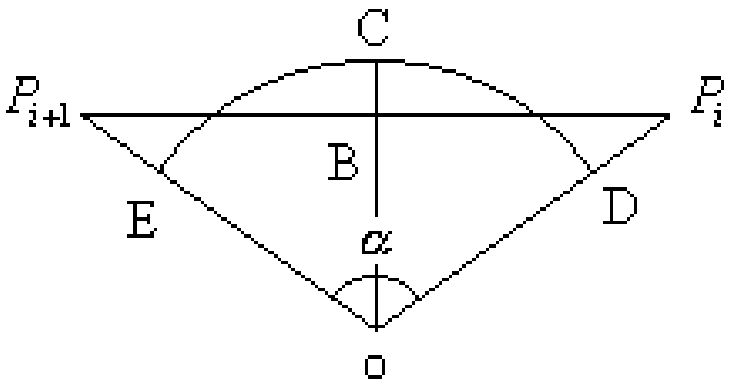
因为：a是常数， sina， cosa只在开始时计算一次所以，一个顶点只需4次乘法，共4n次乘法，外加直线段的中点算法的计算量。

圆的等面积正多边形逼近法

- 当用内接正多边形逼近圆时，其面积要小于圆的面积；
- 而当用圆的外切正多边形逼近圆时，其面积则要大于圆的面积。
- 为了使近似代替圆的正多边形和圆之间在面积上相等，只有使该正多边形和圆弧相交，称之为圆的等面积正多边形。

圆的等面积正多边形逼近法





步骤：

- 求多边形径长，从而求所有顶点坐标值
- 由逼近误差值，确定边所对应的圆心角α

椭圆的扫描转换

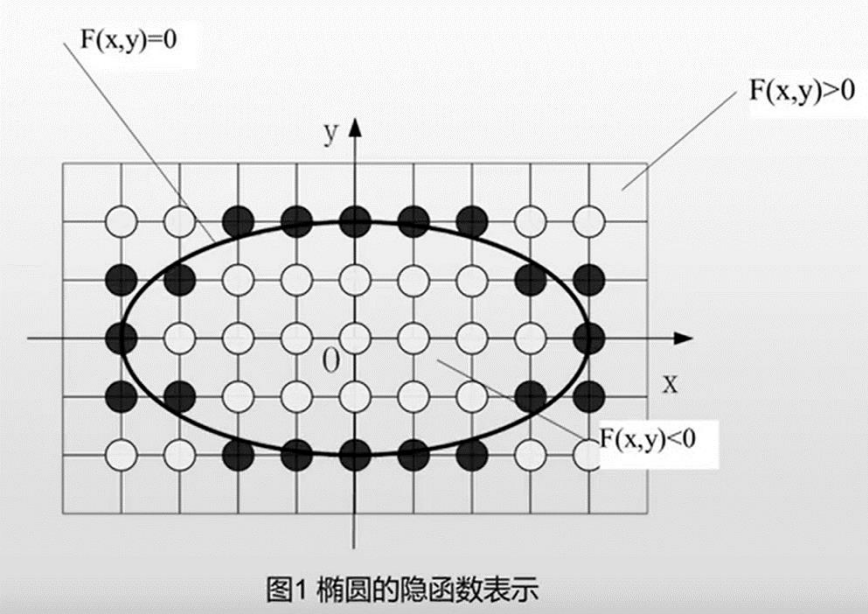


图1 椭圆的隐函数表示

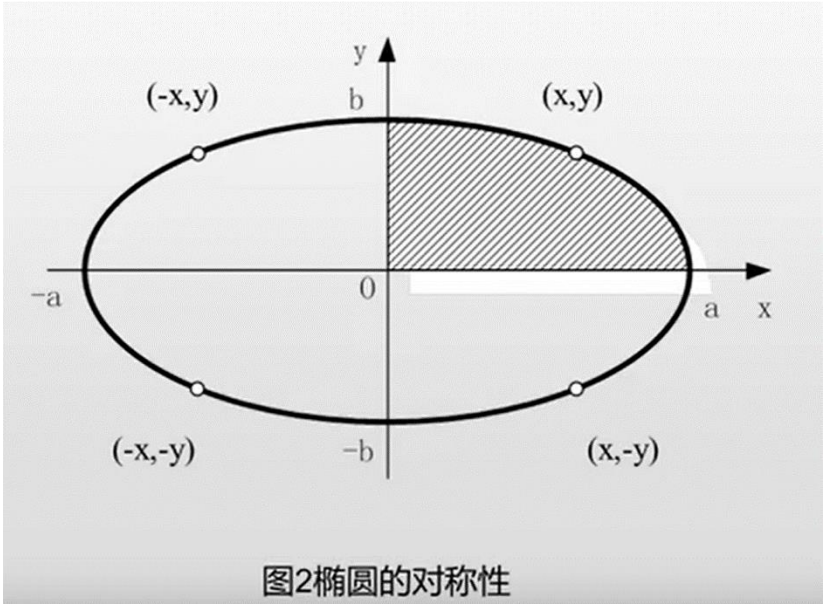
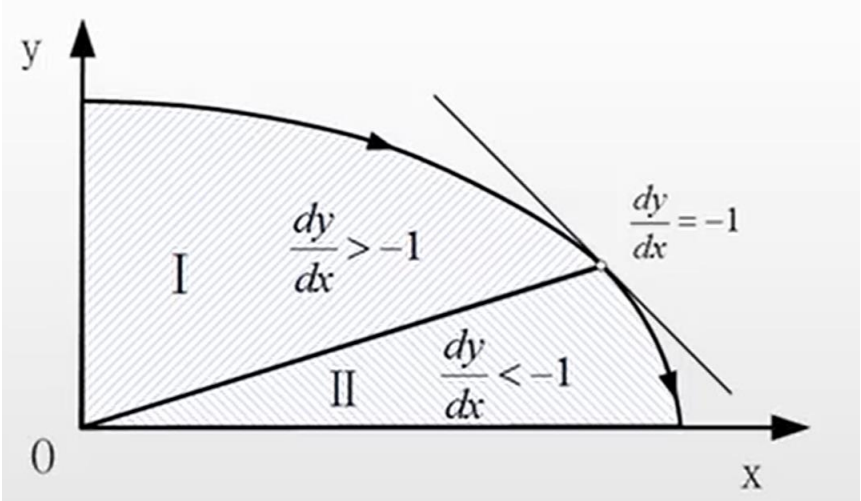


图2椭圆的对称性

$$F(x,y)=b^2x^2+a^2y^2-a^2b^2=0$$

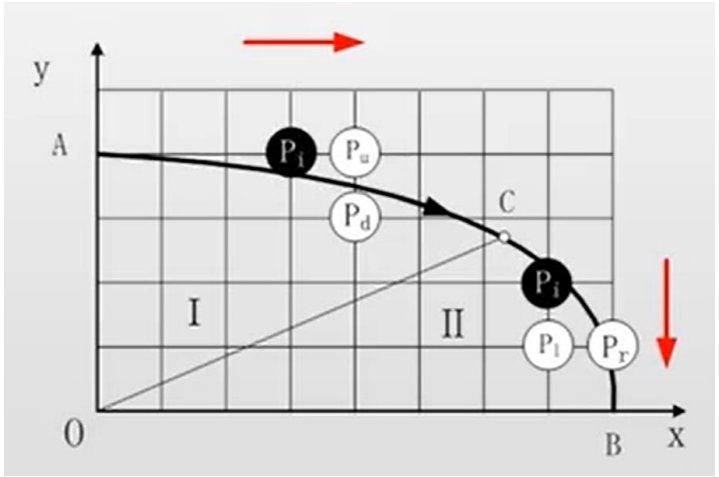
椭圆的扫描转换

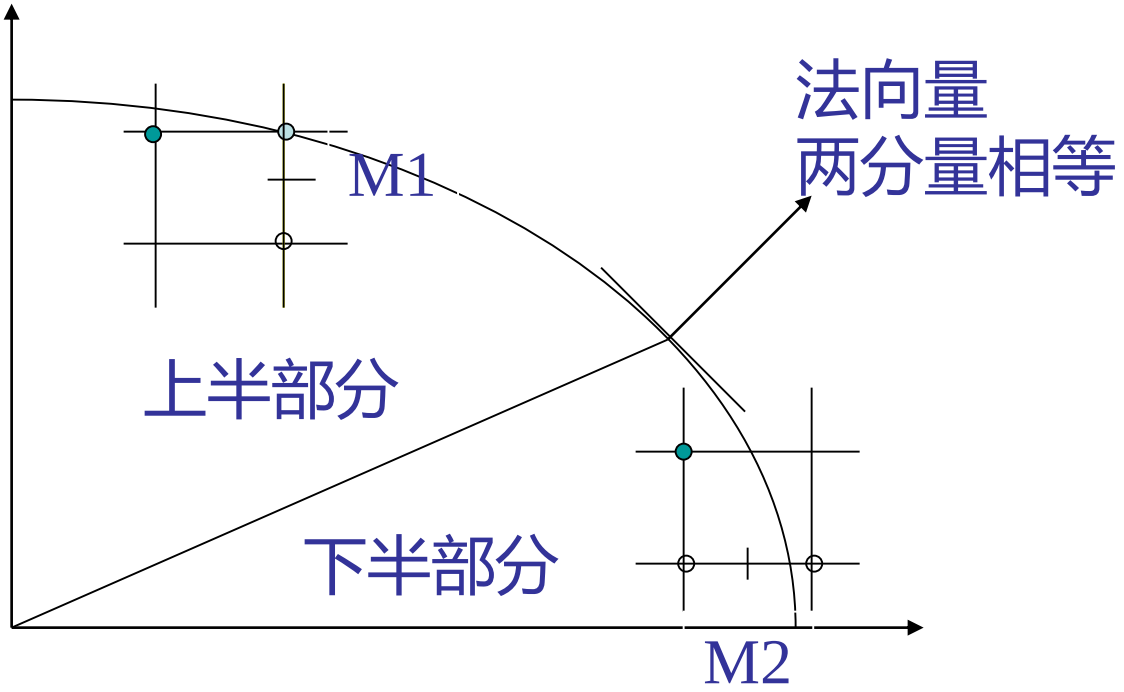


- $F(x,y)=b^2x^2+a^2y^2-a^2b^2=0$
- 椭圆的对称性，只考虑第一象限椭圆弧生成，分上下两部分，以切线斜率为-1的点作为分界点。
- 椭圆上一点处的法向：

$$N(x,y) = N_x i + N_y j = \frac{\partial F}{\partial x} i + \frac{\partial F}{\partial y} j = 2b^2 x i + 2a^2 y j$$

在上半部分,法向量的y分量大
在下半部分,法向量的x分量大





- 在当前中点处，法向量（ $2b^2(Xp+1)$， $2a^2(Yp-0.5)$)的y分量比x分量大，即： $b^2(Xp+1) < a^2(Yp-0.5)$,
- 而在下一中点，不等式改变方向，则说明椭圆弧从上部分转入下部分

椭圆的中点画法

- 与圆弧中点算法类似：确定一个象素后，接着在两个候选象素的中点计算一个判别式的值，由判别式的符号确定更近的点

先讨论椭圆弧的上部分

① 设(Xp，Yp)已确定，则下一待选像素的中点是(Xp+1,Yp-0.5)

② $d1=F(Xp+1,Yp-0.5)= b^2(Xp+1)^2+a^2(Yp-0.5)^2-a^2b^2$

根据d1的符号来决定下一像素是取正右方的那个，还是右下方的那个。

- 若d1＜0，中点在椭圆内，取正右方象素，判别式更新为：

  $d1'=F(Xp+2,Yp-0.5)=d1+b^2(2Xp+3)$

  d1的增量为$b^2(2Xp+3)$

- 当d1≥0，中点在椭圆外，取右下方象素，更新判别式：

  $d1'=F(Xp+2,Yp-1.5)=d1+b^2(2Xp+3)+a^2(-2Yp+2)$

  d1的增量为$b^2(2Xp+3)+a^2(-2Yp+2)$

d1的初始条件：椭圆弧起点为(0，b)；

- 第一个中点为(1, b-0.5)

初始判别式：d10=F(1,b-0.5)=b*b+a*a(-b+0.25)

- 转入下半部分，下一象素可能是正下方或右下方，此时判别式要初始化。

$d2 = F(Xp+0.5,Yp-1) = b^2(Xp+0.5)^2+a^2(Yp-1)^2-a^2b^2$

① 若d2<0,取右下方像素，则

$d2'=F(Xp+1.5,Yp-2) = d2 + b^2(2Xp+2)+a^2(-2Yp+3)$

② 若d2>=0,取正下方像素，则

$d2' = F(Xp+0.5,Yp-2) = d2 + a^2(-2Yp+3)$

下半部分弧的终止条件为 y = 0

程序：MidpointEllipe(a,b, color)

```
int a,b,color;
{  int x,y;  float d1,d2;
   x = 0; y = b;
   d1 = b*b +a*a*(-b+0.25);
   putpixel(x,y,color);
  while( b*b*(x+1) < a*a*(y-0.5))
  {    {  if (d1<0)
                  d1 +=b*b*(2*x+3); x++;  }
         else  {  d1 +=(b*b*(2*x+3)+a*a*(-2*y+2))
                    x++;  y--;          }
      putpixel(x,y,color);
}//上部分
   d2 = sqr(b*(x+0.5))  +sqr(a*(y-1)) – sqr(a*b);
   while(y >0)
 {  if (d2 <0) { d2 +=b*b*(2*x+2)+a*a*(-2*y+3);
                  x++;  y--;  }
    else  {d2 += a*a*(-2*y+3);  y--; }
  putpixel(x,y,color); }}
```