计算机图形学

第二章：光栅图形学算法

随着光栅显示器的出现，为了在计算机上处理、显示图形，需要发展一套与之相适应的算法：

光栅图形学算法

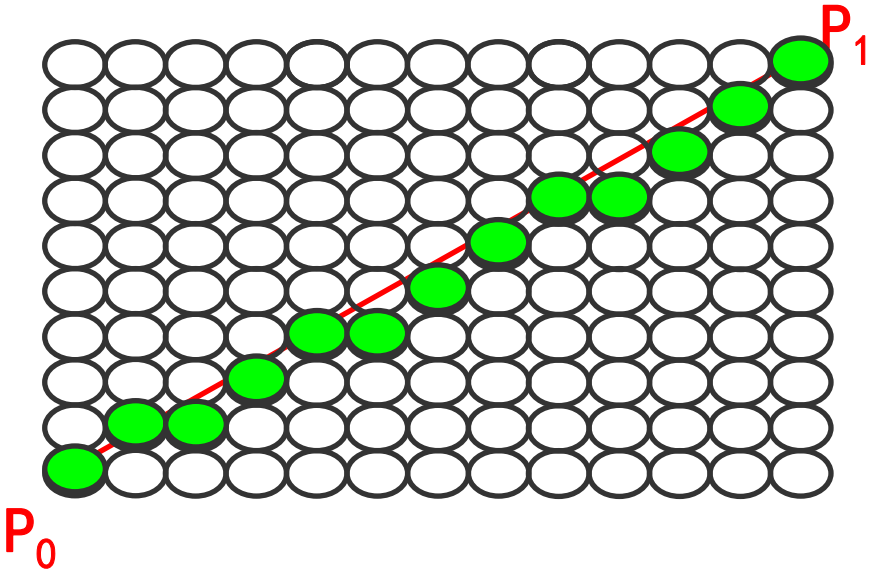
光栅图形算法多数属于计算机图形的底层算法，很多图形学的基本概念和思想都在这一章

光栅图形学算法的研究内容

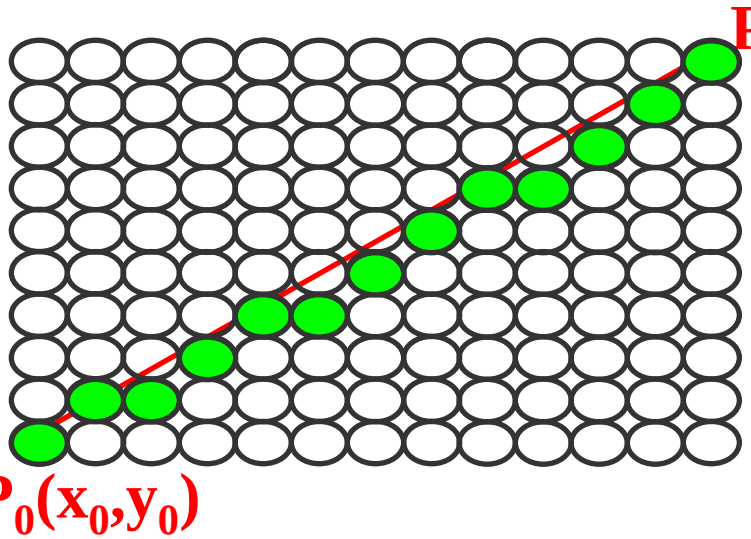
- 直线段的扫描转换算法
- 多边形的扫描转换与区域填充算法
- 裁剪算法
- 反走样算法
- 消隐算法

一、直线段的扫描转换算法

在数学上，直线上的点有无穷多个。但当在计算机光栅显示器屏幕上表示这条直线时需要做一些处理。



为了在光栅显示器上用这些离散的像素点逼近这条直线，需要知道这些像素点的x，y坐标。



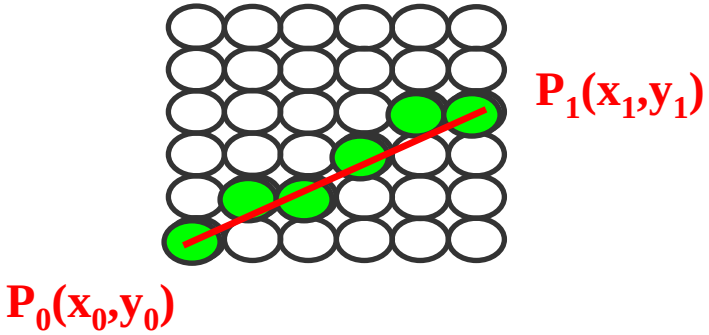
求出过P_0, P_1的直线段方程：

$$y = kx + b$$

$$k = \frac{(y_1 - y_0)}{(x_1 - x_0)} \qquad (x_1 \neq x_0)$$

$$y = kx + b \qquad k = \frac{(y_1 - y_0)}{(x_1 - x_0)} \qquad (x_1 \neq x_0)$$

假设x已知，即从x的起点x_0开始，沿x方向前进一个像素（步长 = 1），可以计算出相应的y值。



因为像素的坐标是整数，所以y值还要进行取整处理

如何把数学上的一个点扫描转换一个屏幕像素点?

如：p（1.7，0.8） —取整→ p（1，0）

p（1.7，0.8） —+0.5→ p（2.2，1.3）

p（2.2，1.3） —取整→ p（2，1）

$$y = kx + b$$

直线是最基本的图形，一个动画或真实感图形往往需要调用成千上万次画线程序，因此直线算法的好坏与效率将直接影响图形的质量和显示速度。

回顾一下刚才的算法：

$$y = kx + b$$

为了提高效率，把计算量减下来，关键问题就是如何把**乘法**取消？

二、直线绘制的三个著名的常用算法

1、数值微分法（DDA）

2、中点画线法

3、Bresenham算法

1、数值微分DDA(Digital Differential Analyzer)法

引进图形学中一个很重要的思想---增量思想

$$y_i = kx_i + b$$

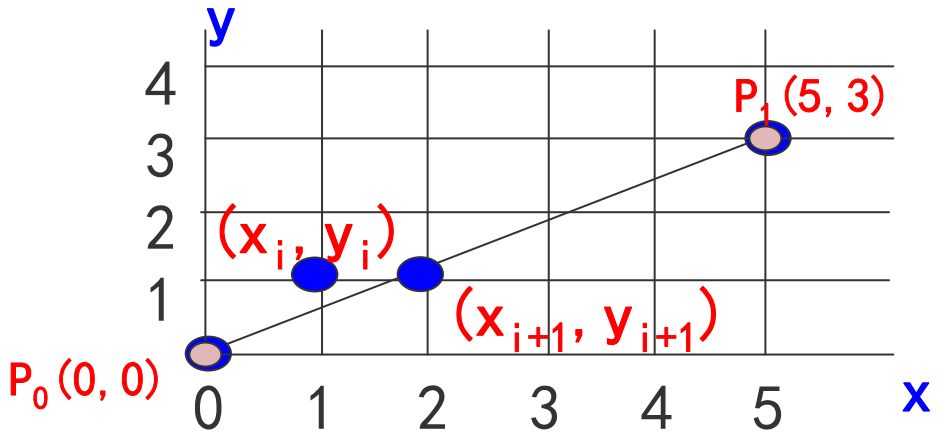
$$y_{i+1} = kx_{i+1} + b$$

$$= k(x_i + 1) + b$$

$$= kx_i + k + b$$

$$= kx_i + b + k$$

$$= y_i + k$$



$$y_{i+1} = y_i + k$$

$$y_{i+1} = y_i + k$$

增量

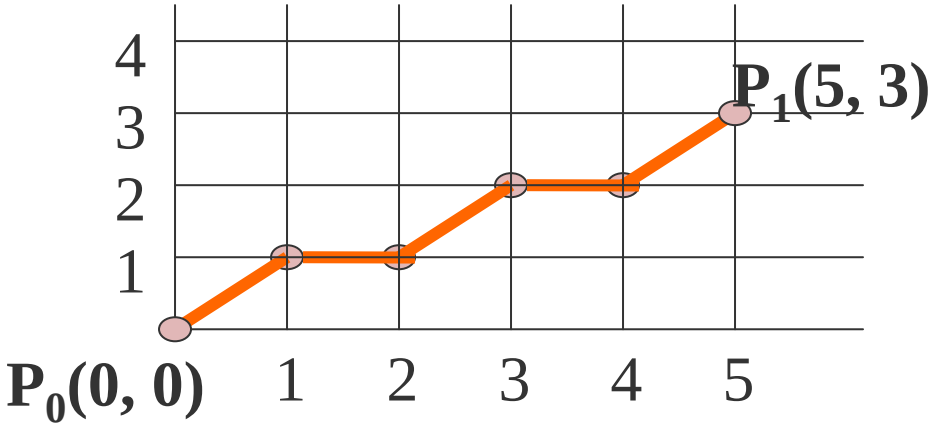
这个式子的含义是：当前步的y值等于前一步的y值加上斜率k

这样就把原来一个乘法和加法变成了一个加法！

用DDA扫描转换连接两点$P_0(0,0)$和$P_1(5,3)$的直线段。

$$k=\frac{y_1-y_0}{x_1-x_0}=\frac{3-0}{5-0}=0.6<1 \qquad y_{i+1}=y_i+k$$

x	y	int(y+0.5)
0	0	0
1	0+0.6	1
2	0.6+0.6	1
3	1.2+0.6	2
4	1.8+0.6	2
5	2.4+0.6	3



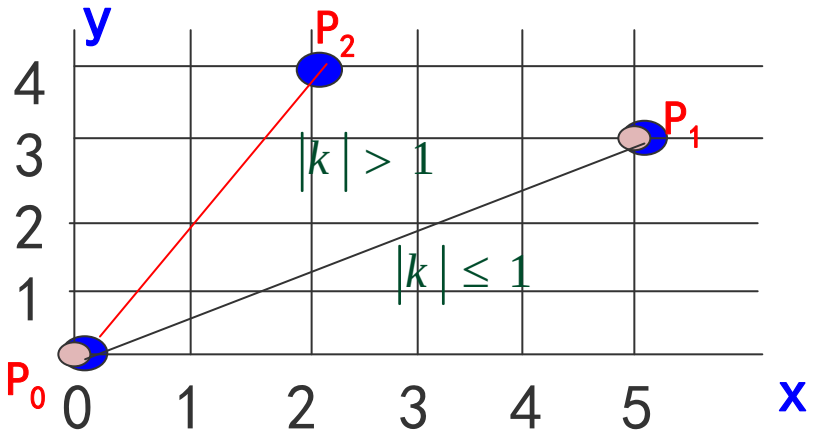
给大家留二个思考题

（1）DDA画直线算法：x每递增1，y递增斜率k。是否适合任意斜率的直线？

$$|k| \le 1$$

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i + k$$

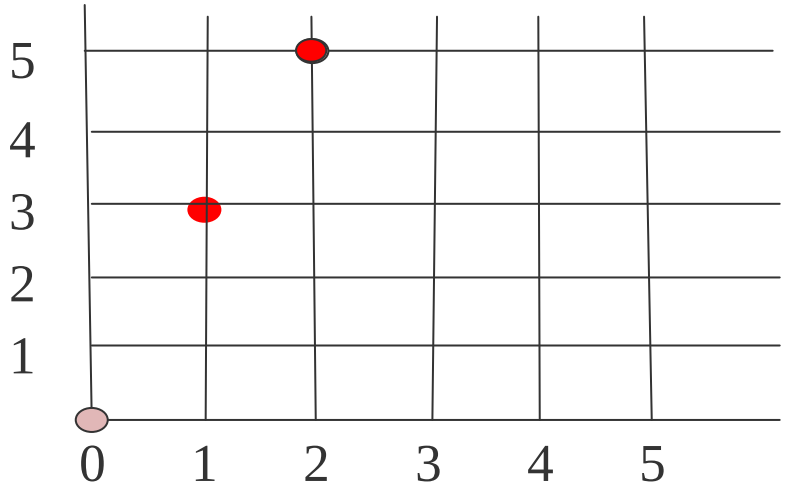


$|k| > 1$

用DDA方法画连接两点P_0（0,0）和P_1（2,5）的直线段

K=5/2=2.5>1　　$y_{i+1} = y_i + k$

x	y	int(y+0.5)
0	0	0
1	2.5	3
2	5	5



再比如直线点从（0，0）到（2，100），也只用3个点来表示

（1） $|k| > 1$

$$x_{i+1} = x_i + ?$$

$$y_{i+1} = y_i + ?$$

（2）DDA画直线算法是否最优呢？若非，如何改进？

采用增量思想的DDA算法，直观、易实现，每计算一个象素坐标，只需计算一个加法。

$$y_{i+1} = y_i + k$$

这个算法是否最优呢？若非最优，如何改进？

（1）改进效率。这个算法每步只做一个加法，能否再提高效率？

$$y_{i+1} = y_i + k$$

一般情况下k与y都是小数，而且每一步运算都要对y进行四舍五入后取整。

唯一改进的途径是把浮点运算变成整数加法！

（2）第二个思路是从直线方程类型做文章

$$y = kx + b$$

而直线的方程有许多类型，如两点式、一般式等。如用其它的直线方程来表示这条直线会不会有出人意料的效果？

直线绘制的三个著名的常用算法

1、数值微分法（DDA）

2、中点画线法

3、Bresenham算法

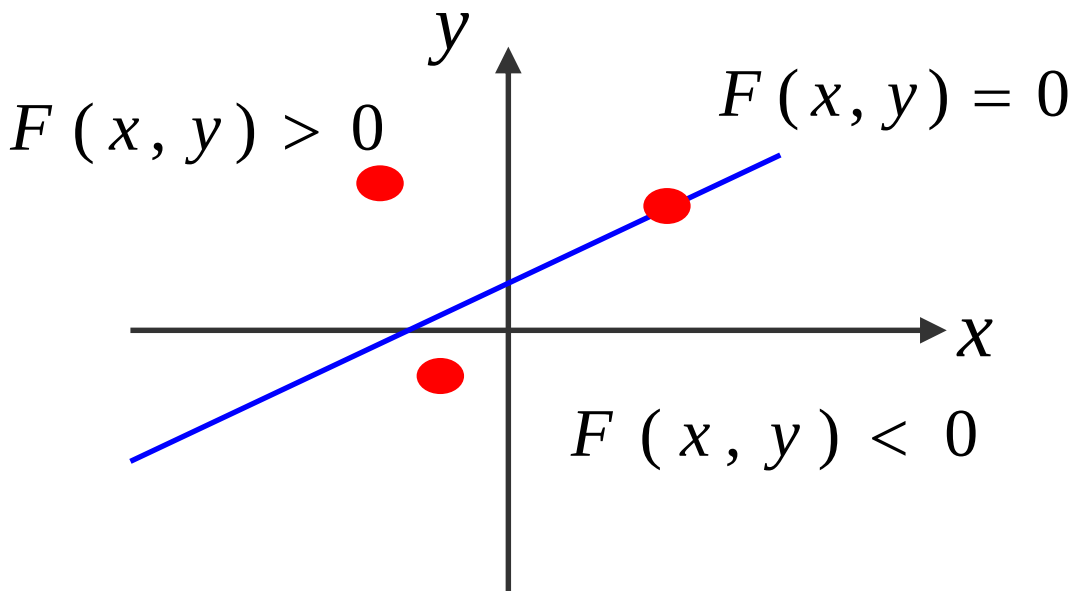
中点画线法

直线的一般式方程：

$$F(x,y)=0$$

$$Ax+By+C=0$$

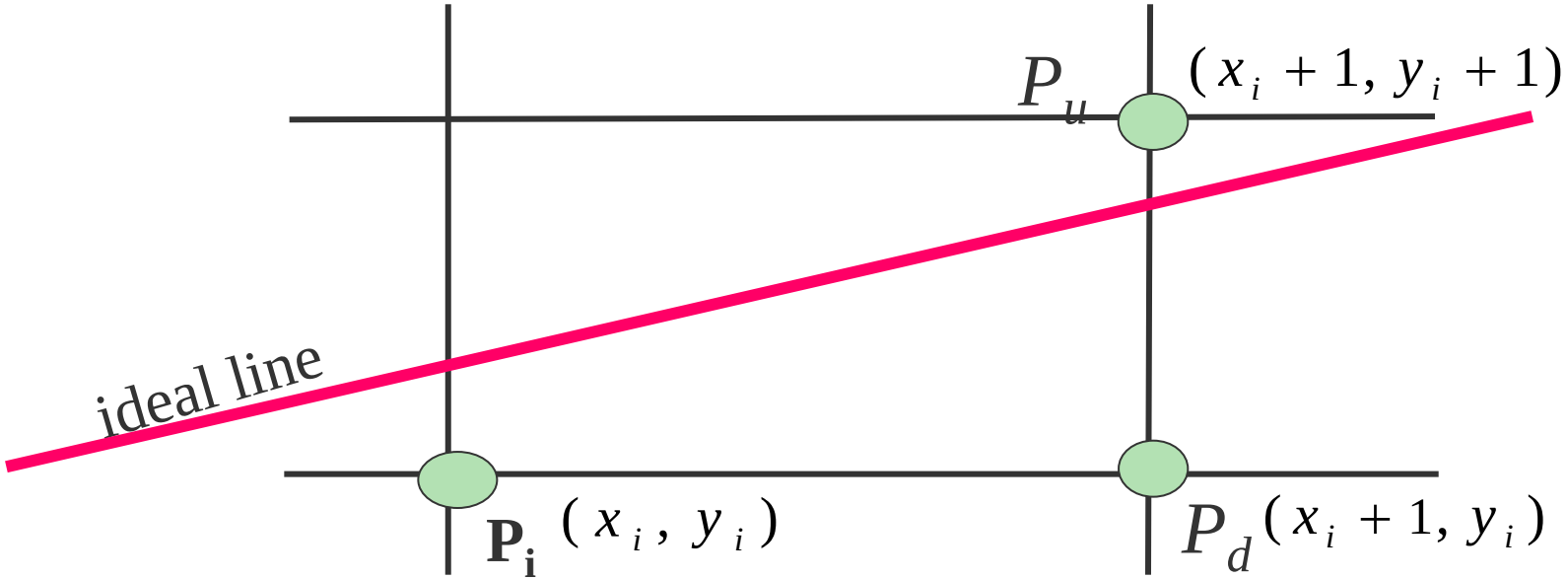
其中：$A=-(\Delta y);\quad B=(\Delta x);\quad C=-B(\Delta x)$

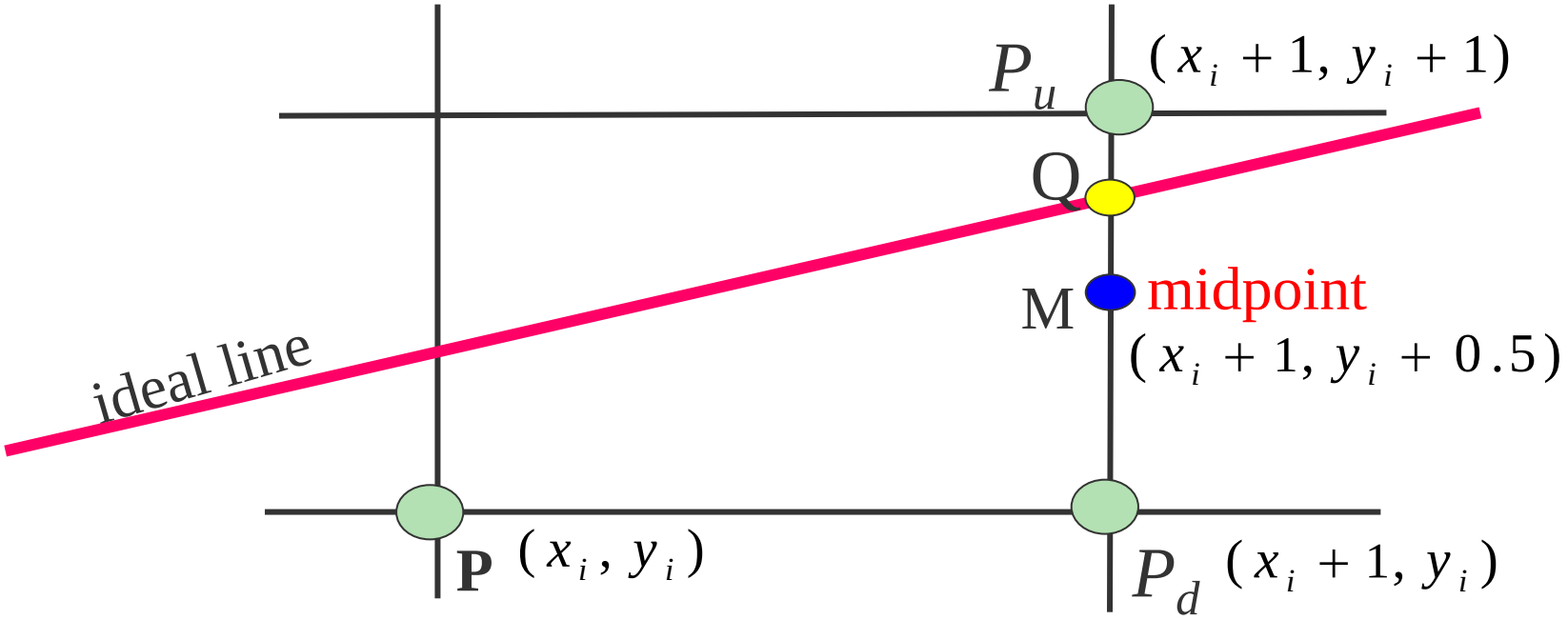


- 对于直线**上**的点：$F(x, y) = 0$
- 对于直线**上方**的点：$F(x, y) > 0$
- 对于直线**下方**的点：$F(x, y) < 0$

每次在最大位移方向上走一步，而另一个方向是走步还是不走步要取决于中点误差项的判断。

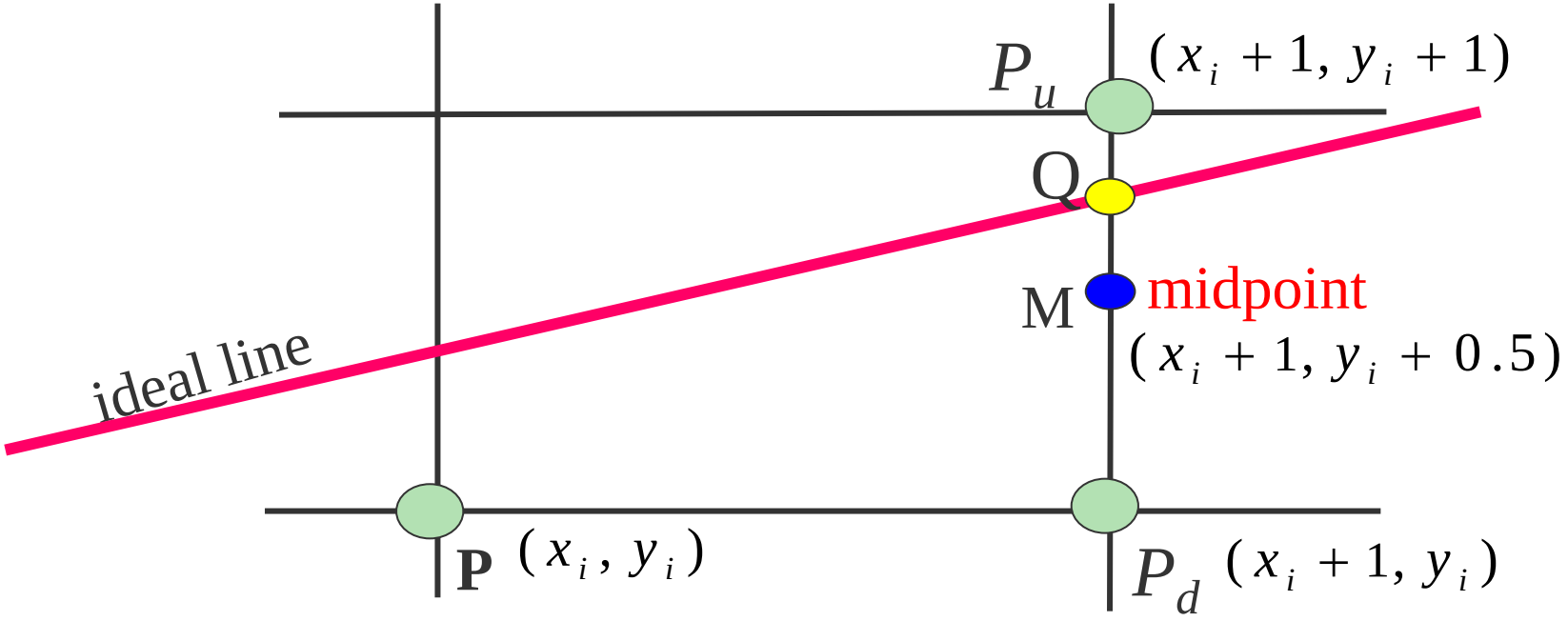
假定：0⩽|k|⩽1。因此，每次在x方向上加1，y方向上加1或不变需要判断。



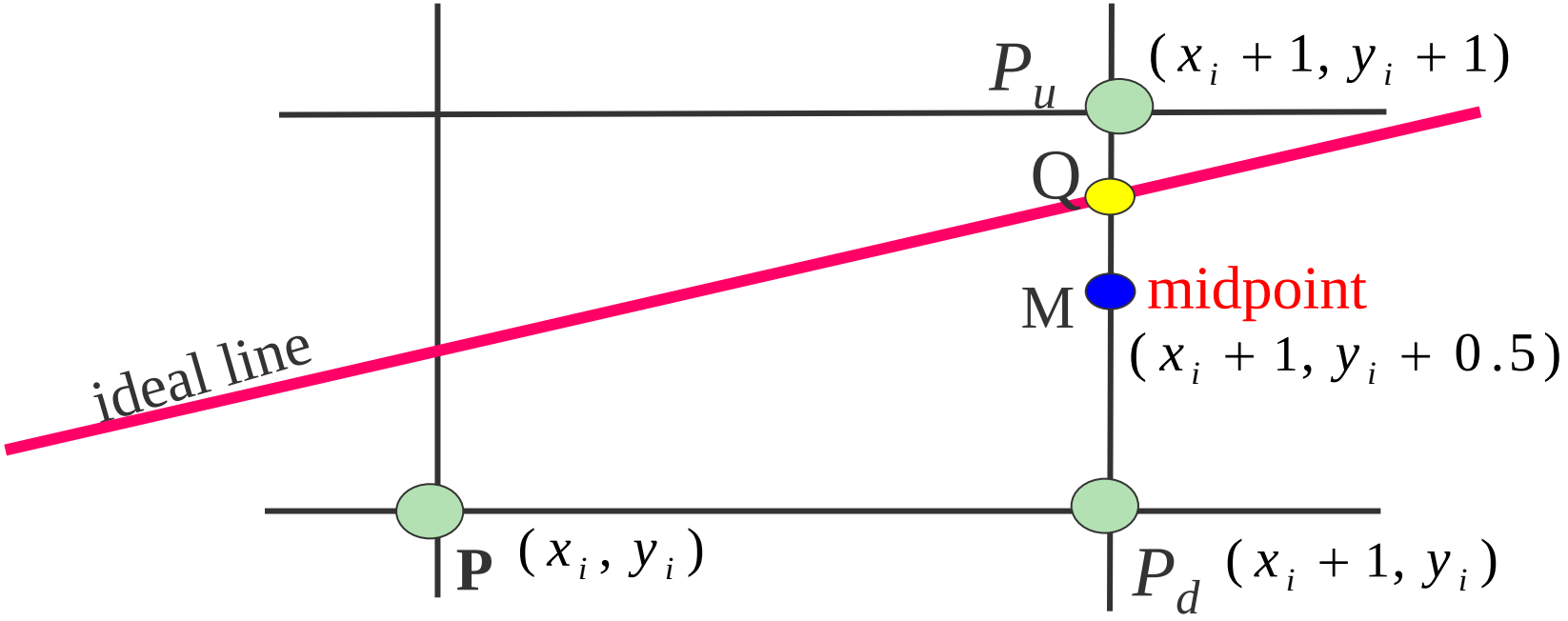


当M在Q的下方，则P_u离直线近，应为下一个象素点

当M在Q的上方，应取P_d 为下一点。



如何判断Q在M的上方还是下方？

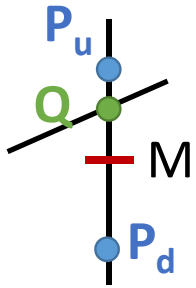


把M代入理想直线方程：

$$F(x_m, y_m) = Ax_m + By_m + C$$

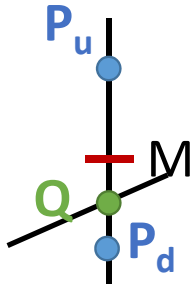
$$d_i = F(x_m, y_m) = F(x_i + 1, y_i + 0.5)$$
$$= A(x_i + 1) + B(y_i + 0.5) + C$$

当d<0时：



M在Q下方，应取P_u

当d>0时：



M在Q上方，应取P_d

当d=0时：

M在直线上，选P_u或P_d均可。

$$d_i = F(x_m, y_m) = F(x_i + 1, y_i + 0.5)$$
$$= A(x_i + 1) + B(y_i + 0.5) + C$$

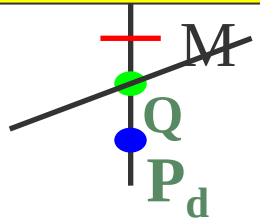
当d＜

$$y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

P_u

这就是中点画线法的基本原理

当d＞0 时：



M在Q上方，应取P_d

当d＝ 0时： M在直线上，选p_d或p_u均可。

下面来分析一下中点画线算法的计算量？

$$y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

$$d_i = A(x_i + 1) + B(y_i + 0.5) + C$$

为了求出d值，需要两个乘法，四个加法

能否也采用增量计算，提高运算效率呢？

$$d_{i+1} = d_i + ?$$

增量

分析一下中点画线算法的计算量

$$y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

$$d_i = A(x_i + 1) + B(y_i + 0.5) + C$$

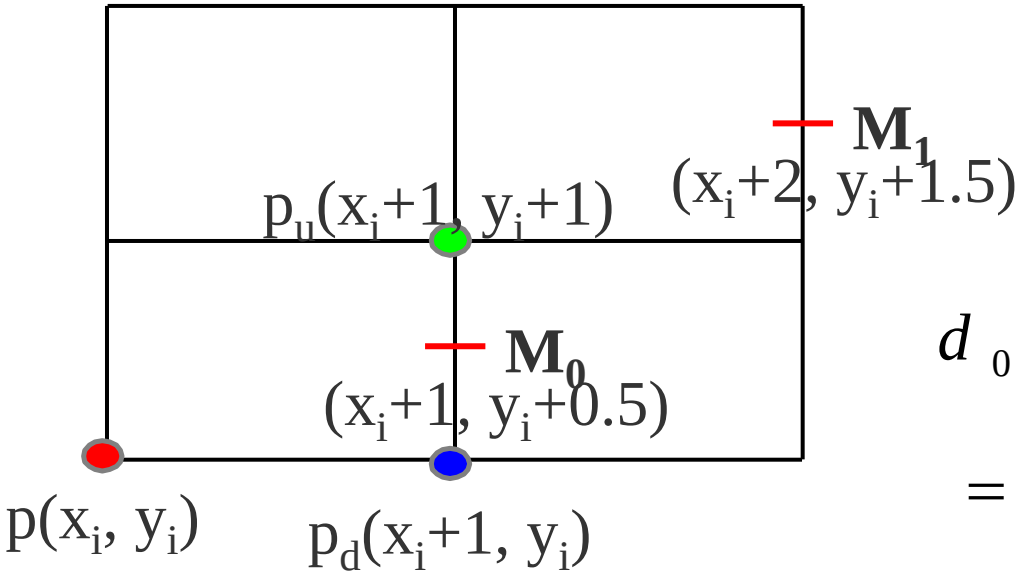
能否也采用增量计算，提高运算效率呢？

$$d_{i+1} = d_i + ?$$

增量

d是x, y的线性函数，采用增量计算是可行的

如何推导出d值的递推公式？



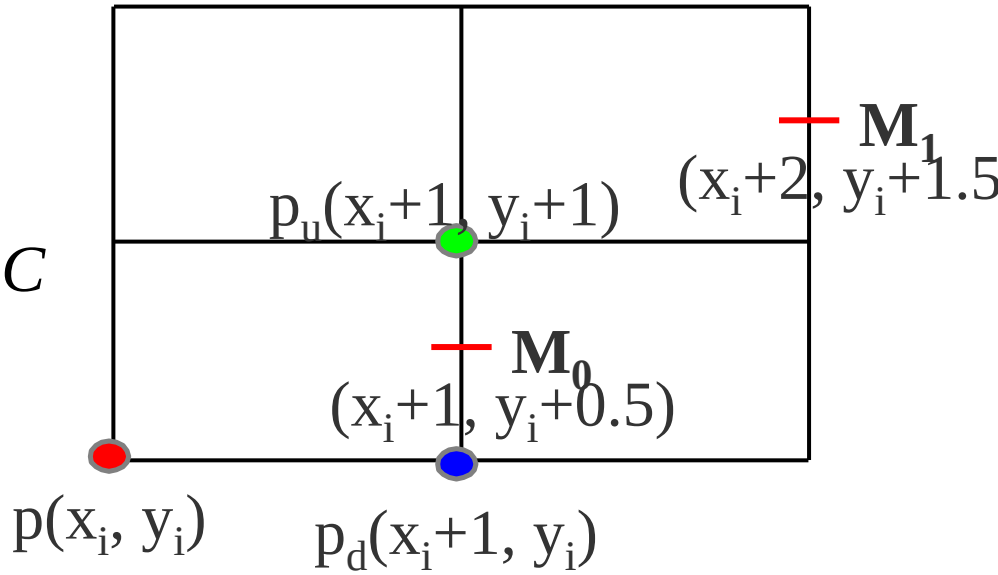
$$y = \begin{cases} y+1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

$$d_0 = F(x_{m0}, y_{m0})$$

$$= F(x_i + 1, y_i + 0.5)$$

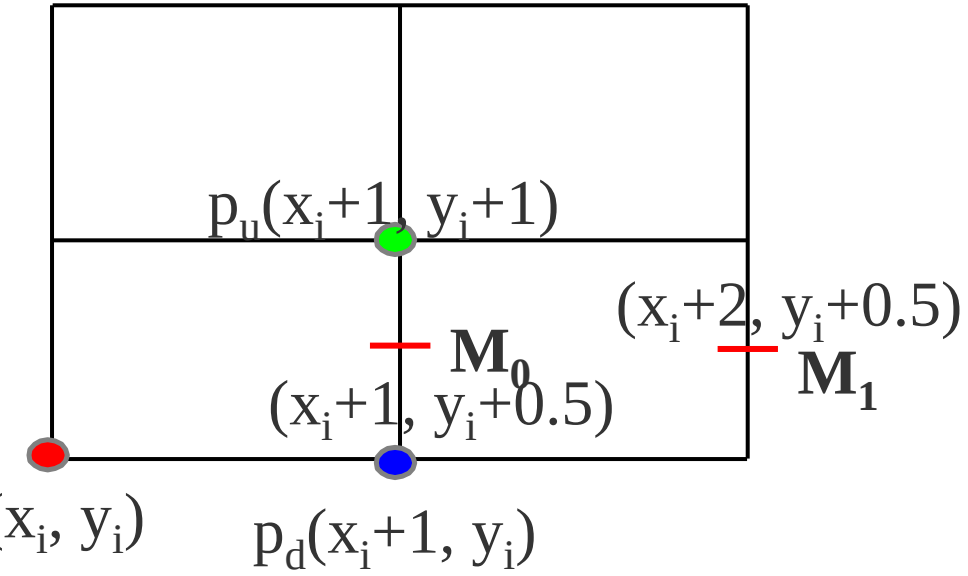
$$= A(x_i + 1) + B(y_i + 0.5) + C$$

$$d_0 = F(x_{m0}, y_{m0})$$
$$= F(x_i + 1, y_i + 0.5)$$
$$= A(x_i + 1) + B(y_i + 0.5) + C$$



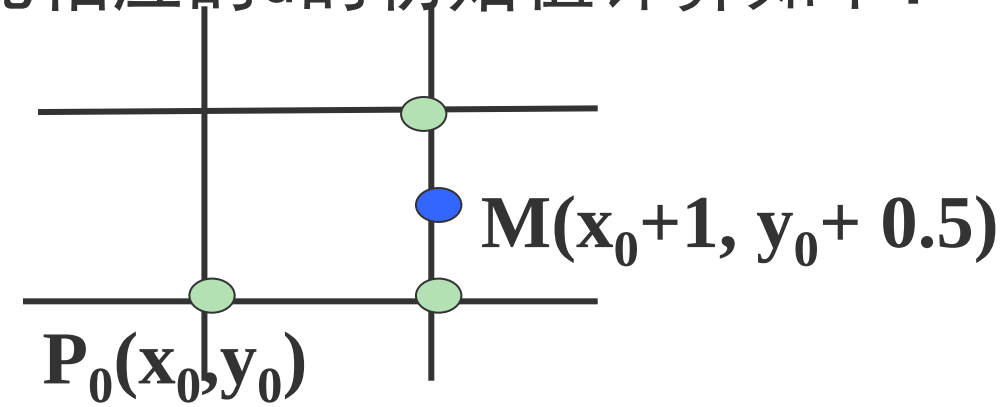
$$d_1 = F(x_{m1}, y_{m1})$$
$$= F(x_i + 2, y_i + 1.5)$$
$$= A(x_i + 2) + B(y_i + 1.5) + C$$
$$= A(x_i + 1) + B(y_i + 0.5) + C + A + B$$
$$= d_0 + A + B$$

$$y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$



$$\begin{aligned} d_1 &= F(x_i + 2, y_i + 0.5) \\ &= A(x_i + 2) + B(y_i + 0.5) + C \\ &= A(x_i + 1) + B(y_i + 0.5) + C + A \\ &= d_0 + A \end{aligned}$$

下面计算d的初始值d_0，直线的第一个像素$P_0(x_0, y_0)$在直线上，因此相应的d的初始值计算如下：



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

$$= A(x_0 + 1) + B(y_0 + 0.5) + C$$

$$= Ax_0 + By_0 + C + A + 0.5B$$

$$= A + 0.5B$$

$$d_{new} = \begin{cases} d_{old} + A + B & d < 0 \\ d_{old} + A & d \geq 0 \end{cases} \qquad d_0 = A + 0.5B$$

至此，中点算法至少可以和DDA算法一样好！

可以用2d代替d来摆脱浮点运算，写出仅包含整数运算的算法。

这样，中点生成直线的算法提高到整数加法，优于DDA算法。

小 结

（1） $Ax + By + C = 0$

（2）通过判中点的符号，最终可以只进行整数加法

这个算法是否还有改进的余地？

中点画线法

$$y = \begin{cases} y+1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

这个算法是否还有改进的余地？

能否提出一个算法，使这个算法不但能解决画直线，还能解决圆弧、抛物线甚至自由曲线的光栅化问题，使算法的覆盖域扩大。

直线绘制的三个著名的常用算法

1、数值微分法（DDA）

2、中点画线法

3、Bresenham算法

DDA把算法效率提高到每步只做一个加法。

中点算法进一步把效率提高到每步只做一个整数加法

Bresenham提供了一个更一般的算法。该算法不仅有好的效率，而且有更广泛的适用范围

E.Jack Bresenham

Biography [edit]

He retired from 27 years of service at IBM as a Senior Technical Staff Member in 1987. He taught for 16 years at Winthrop University and has nine patents[1]. He has three children: Janet, Linda, and David.

Bresenham's line algorithm, developed in 1962, is his most well-known innovation. It determines which points on a 2-dimensional raster should be plotted in order to form a straight line between two given points, and is commonly used to draw lines on a computer screen. It is one of the earliest algorithms discovered in the field of computer graphics. The Midpoint circle algorithm shares some similarities to his line algorithm and is known as *Bresenham's circle algorithm*.

- Ph.D., Stanford University, 1964
- MSIE, Stanford University, 1960
- BSEE, University of New Mexico, 1959

communication of the ACM

Graphics and Image Processing

W. Newman* Editor

A Linear Algorithm for Incremental Digital Display of Circular Arcs

Jack Bresenham
IBM System Communications Division

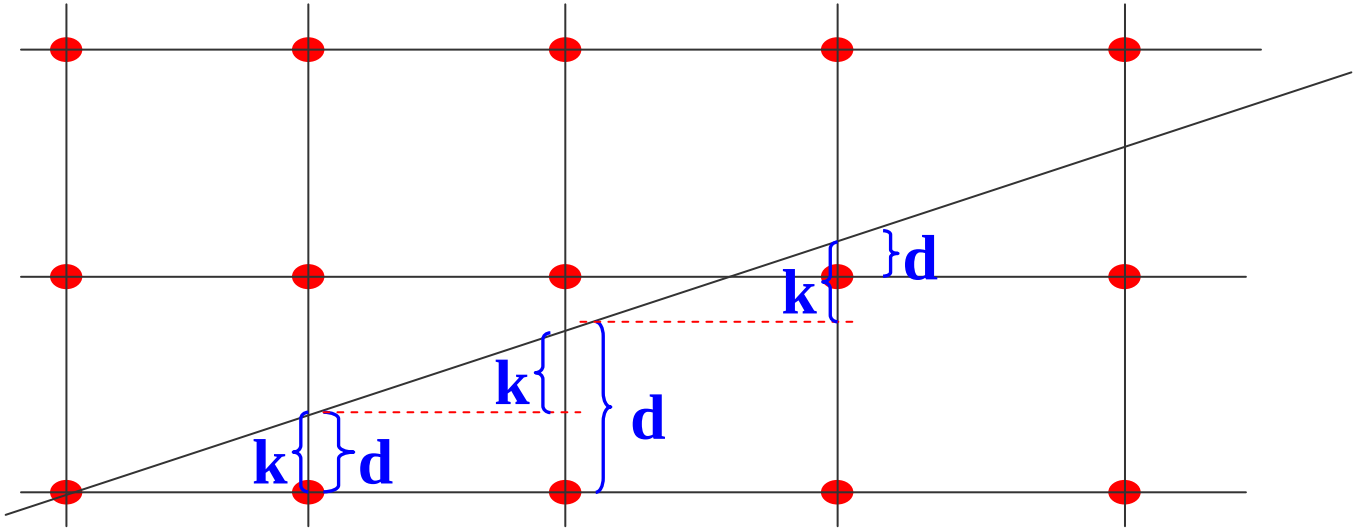
Circular arcs can be drawn on an incremental display device such as a cathode ray tube, digital plotter, or matrix printer using only sign testing and elementary addition and subtraction. This paper describes methodology for producing dot or step patterns closest to the true circle.

This paper describes an algorithm for circular arc mesh point selection using incremental display devices such as a cathode ray tube or digital plotter. Error criteria are explicitly specified and both squared and radial error minimization considered. The repetitive incremental stepping loop for point selection requires only simple addition/subtraction and sign testing; neither quadratic nor trigonometric evaluations are required. When a circle's center point and radius are integers, only integer calculations are required.

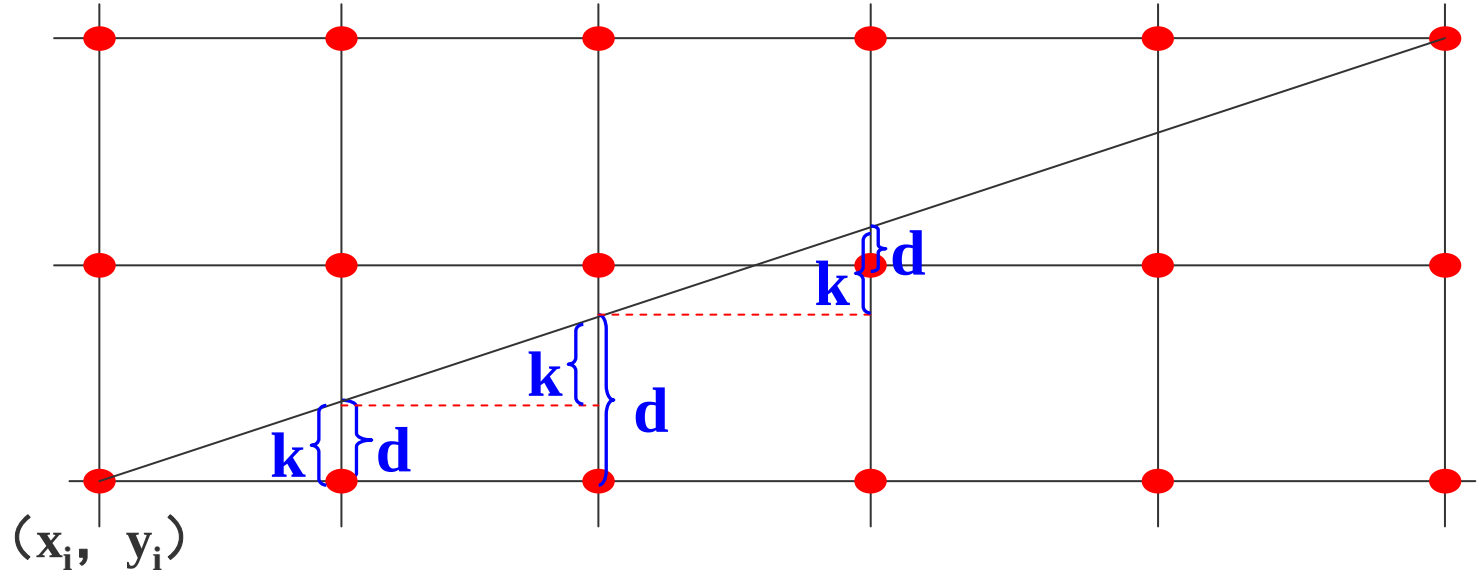
The circle algorithm complements an earlier line algorithm described in [1, 2]. The algorithm's minimum error point selection is appropriate for use in numerical control, drafting, or photo mask preparation applications where closeness of fit is a necessity. Its simplicity and use only of elementary addition/subtraction allow its use in small computers, programmable terminals, or direct hardware implementations where compactness and speed are desirable.

The display devices under consideration are capable of executing, in response to an appropriate pulse, any one of the eight linear movements shown in Figure 1. Thus incremental movement is from a point on a mesh to any of its eight adjacent points on the mesh.

Bresenham算法的基本思想



该算法的思想是通过各行、各列像素中心构造一组虚拟网格线，按照直线起点到终点的顺序，计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列象素中与此交点最近的象素。

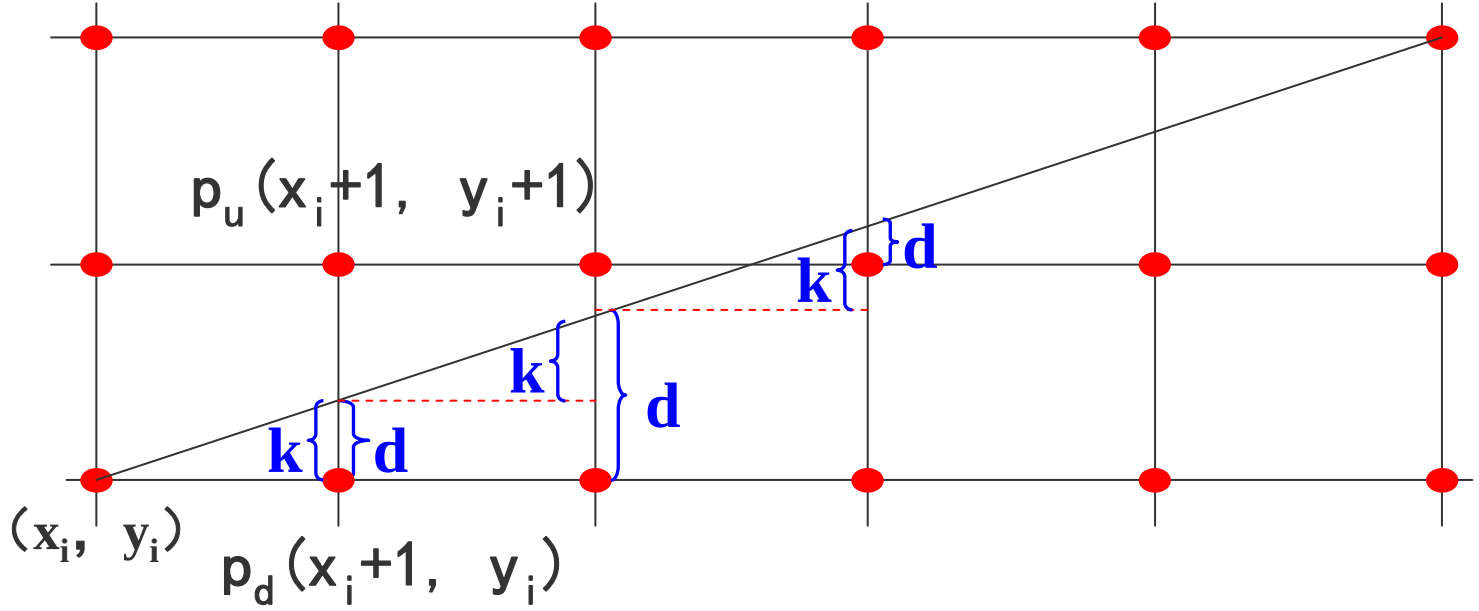


假设每次x+1，y的递增（减）量为0或1，它取决于实际直线与最近光栅网格点的距离，这个距离的最大误差为0.5。

误差项d的初值$d_0=0$

$$d=d+k$$

一旦d≥1，就把它减去1，保证d的相对性，且在0、1之间。



$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (d>0.5) \\ y_i & (d\leqslant 0.5) \end{cases} \end{cases}$$

关键是把这个算法的效率也搞到整数加法，否则就是失败。如何提高到整数加法？

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (d>0.5) \\ y_i & (d\leqslant 0.5) \end{cases} \end{cases}$$

如何把这个算法的效率也提高到整数加法？

改进1：令 e = d−0.5

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

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

e ＞ 0，y方向递增1；e<0，y方向不递增

e = 0 时，可任取上、下光栅点显示

- $e_{初} = -0.5$
- 每走一步有e = e+k
- if (e>0) then e=e-1

$e_{初} = -0.5$ $k = \frac{dy}{dx}$

改进2：由于算法中只用到误差项的符号，于是可以用e*2*△x来替换e。

- $e_{初}$ = -△x，
- 每走一步有：e = e+2△y。
- if (e>0) then e = e-2△x

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、$e=-\triangle x$、$x=x_0$、$y=y_0$。
3. 绘制点(x, y)。
4. e更新为$e+2\triangle y$，判断e的符号。若e>0，则(x, y)更新为(x+1, y+1)，同时将e更新为$e-2\triangle x$；否则(x, y)更新为(x+1, y)。
5. 当直线没有画完时，重复步骤3和4。否则结束。

Bresenham算法很像DDA算法，都是加斜率

但DDA算法是每次求出一个新的y以后取整来画；而Bresenham算法是判符号来决定上下两个点。所以该算法集中了DDA和中点两个算法的优点，而且应用范围广泛

小 结

1、计算机科学问题的核心就是算法

把一个含有乘法和一个加法的普通直线算法，是如何通过改进和完善其性能，最终变成整数加法的一个精彩过程

2、领会算法中所蕴含的创新思想

改进和完善算法的过程中所体现出来的一些闪光的思想是我们所要认识和领会的

3、科学研究无止境，学术面前人人平等

学会研究性学习，对已有的算法提出质疑找出其不足。

1	Bresenham直线生成算法的改进	贾银亮; 张焕春; 经亚枝	中国图象图形学报	2008-01-15	期刊	21	470		
2	基于Bresenham画线算法的图像快速-高精度旋转算法	石慎; 张艳宁; 郗润平; 郑江滨	计算机辅助设计与图形学学报	2007-11-15	期刊	13	211		
3	基于Bresenham算法的四步画直线算法	林笠	暨南大学学报(自然科学与医学版)	2003-10-30	期刊	16	477		
4	改进的Bresenham直线生成算法	刘晶; 李俊; 孙涵	计算机应用与软件	2008-10-15	期刊	11	202		
5	改进的直线Bresenham算法	李高平; 檀结庆	合肥工业大学学报(自然科学版)	2003-10-28	期刊	10	343		
6	Bresenham画圆算法的改进	王志喜; 王润云	计算机工程	2004-06-20	期刊	11	435		

1. 标题: Bresenham Algorithm: Implementation and Analysis in Raster Shape
作者: Gaol, F.L.
来源出版物: Journal of Computers 卷: 8 期: 1 页: 69-78 DOI: 10.4304/jcp.8.1.69-78 出版年: Jan. 2013
被引频次: 0 (来自所有数据库)
[查看摘要]

2. 标题: Image Enhancement with Rotating Kernel Transformation Filter Generated by Bresenham's Algorithm
作者: Seung-Won Shin; Kyeong-Seop Kim; SeMin Lee; 等.
来源出版物: Transactions of the Korean Institute of Electrical Engineers 卷: 61 期: 6 页: 872-8 DOI: 10.5370/KIEE.2012.61.6.872 出版年: June 2012
被引频次: 0 (来自所有数据库)