



直线段扫描转换

王坤峰 教授

信息科学与技术学院



内容

- 光栅图形
- 直线的扫描转换
 - 数值微分法
 - 中点画线法
 - Bresenham算法



光栅图形

■ 光栅显示器

- 可以看作是一个像素矩阵，在光栅(**Raster**)显示器上显示的任何一个图形，实际上都是一些具有一种或多种颜色的像素集合。

■ 光栅图形： 对一个具体的光栅显示器来说，像素个数是有限的，像素的颜色和灰度等级也是有限的，像素是有大小的，所以光栅图形只是近似的实际图形。

- 光栅显示器上显示的图形，称之为光栅图形。
- 如何使光栅图形最完美地逼近实际图形，便是光栅图形学要研究的内容。

■ 图形的扫描转换

- 确定最佳逼近图形的像素集合，并用指定的颜色和灰度设置像素的过程称为图形的扫描转换(Scan Conversion)或光栅化(Rasterization)。

■ 光栅化问题

- 对于一维图形，在不考虑线宽时，用一个像素宽的直线或曲线来显示图形。
- 二维图形的光栅化必须确定区域对应的像素集，将各个像素设置成指定的颜色和灰度，也称之为区域填充(Region Filling)。

- 图形光栅化后，显示在屏幕上的一个窗口里，超出窗口的部分不予显示。确定一个图形的哪些部分在窗口内，必须显示；哪些部分落在窗口之外，不予显示，这需要对图形进行**裁剪(Clipping)**。
- 在光栅图形中，非水平和垂直的直线用像素集合表示时，会呈锯齿状，这种现象称之为**走样(Aliasing)**；用于减少或消除走样的技术称为**反走样(Anti-Aliasing)**。



直线的扫描转换

■ 直线

- 数学上的直线是没有宽度、由无数个点构成的集合。
- 光栅显示器只能近似地显示直线。

■ 直线的扫描转换

- 当我们对直线进行光栅化时，需要在显示器有限个像素中，确定最佳逼近该直线的一组像素，并且按扫描线顺序，对这些像素进行写操作，这个过程称为直线的扫描转换。
 - 数值微分法 (DDA line algorithm)
 - 中点画线法 (Midpoint line algorithm)
 - Bresenham算法(Bresenham's line algorithm)

数值微分法

■ 分析

- 设过端点 $P_0(x_0, y_0)$ 、 $P_1(x_1, y_1)$ 的直线段为 $L(P_0, P_1)$,
- 斜率
$$k = \frac{y_1 - y_0}{x_1 - x_0}$$
- L 的起点 P_0 的横坐标 x_0 向 L 的终点 P_1 的横坐标 x_1 步进, 取步长=1(个像素), 用 L 的直线方程 $y=kx+b$ 计算相应的 y 坐标, 并取像素点 $(x, \text{round}(y))$ 作为当前点的坐标。

$$\begin{aligned}x_{i+1} &= x_i + \Delta x \\y_{i+1} &= kx_{i+1} + b \\&= kx_i + b + k\Delta x \\&= y_i + k\Delta x\end{aligned}$$

- 当 $\Delta x = 1$ 时 $y_{i+1} = y_i + k$ 。
- 也就是说，当 x 每递增1， y 递增 k (即直线斜率)。

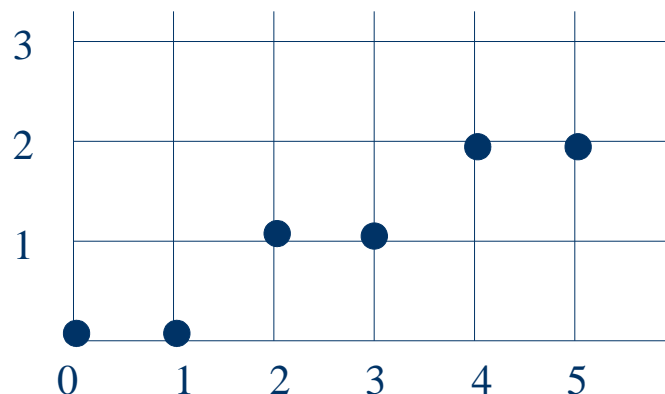
■ DDA画线算法程序

```
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;
  }
```

■ 举例

- 连接两点 $P_0(0,0)$ 和 $P_1(5,2)$ 的直线段

x	$\text{int}(y+0.5)$	$y+0.5$
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	$2.0+0.5$



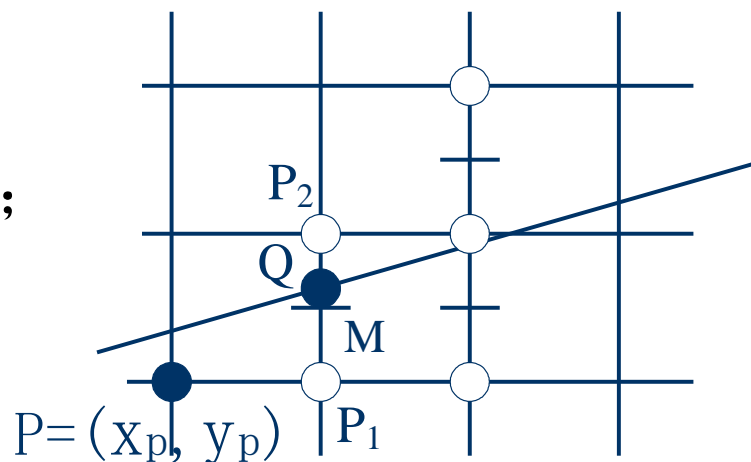
■ 算法分析

- 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， y 每增加1， x 相应增加 $1/k$ 。
- 在这个算法中， y 与 k 必须用浮点数表示，而且每一步都要对 y 进行四舍五入后取整，这使得它不利于硬件实现。

中点画线法

■ 分析

- 假定直线斜率 k 在 $0 \sim 1$ 之间，当前像素点为 (x_p, y_p) ，则下一个像素点有两种可选择点
 - $P_1(x_p+1, y_p)$ 或 $P_2(x_p+1, y_p+1)$ 。
- 设 P_1 与 P_2 的中点 $(x_p+1, y_p+0.5)$ 称为 M ， Q 为理想直线与 $x=x_p+1$ 垂线的交点。
- 当 M 在 Q 的下方时，则取 P_2 应为下一个像素点；
- 当 M 在 Q 的上方时，则取 P_1 为下一个像素点。



- 过点 (x_0, y_0) 、 (x_1, y_1) 的直线段 L 的方程式为
 - $F(x, y) = ax + by + c = 0$
 - 其中, $a = y_0 - y_1$, $b = x_1 - x_0$, $c = x_0 y_1 - x_1 y_0$,
- 欲判断中点 M 在 Q 点的上方还是下方, 只要把 M 代入 $F(x, y)$, 并判断它的符号即可。
- 为此, 我们构造判别式:
 - $d = F(M) = F(x_p + 1, y_p + 0.5) = a(x_p + 1) + b(y_p + 0.5) + c$
 - 当 $d < 0$ 时, M 在 $L(Q)$ 下方, 取 P_2 为下一个像素;
 - 当 $d > 0$ 时, M 在 $L(Q)$ 上方, 取 P_1 为下一个像素;
 - 当 $d = 0$ 时, 选 P_1 或 P_2 均可, 约定取 P_1 为下一个像素。

- d 是 x_p, y_p 的线性函数，可采用增量 (Incremental) 计算，提高运算效率。
- 若当前像素处于 $d \geq 0$ 情况，则取正右方像素 $P_1(x_p+1, y_p)$ ，要判断再下一个像素位置，应计算
 - $d_1 = F(x_p+2, y_p+0.5) = a(x_p+2) + b(y_p+0.5) + c = d + a$
 - 增量为 a
- 若 $d < 0$ 时，则取右上方像素 $P_2(x_p+1, y_p+1)$ 。要判断再下一像素，则要计算
 - $d_2 = F(x_p+2, y_p+1.5) = a(x_p+2) + b(y_p+1.5) + c = d + a + b$
 - 增量为 $a + b$

- 画线从 (x_0, y_0) 开始, d 的初值
 - $d_0 = F(x_0 + 1, y_0 + 0.5) = F(x_0, y_0) + a + 0.5b$
 - 因为 $F(x_0, y_0) = 0$, 所以 $d_0 = a + 0.5b$
- 由于我们使用的只是 d 的符号, 而且 d 的增量都是整数, 只是初始值包含小数。因此, 可以用 $2d$ 代替 d 来摆脱小数。

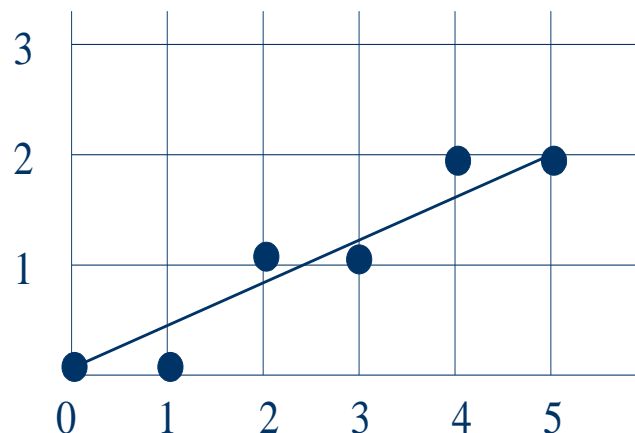
■ 中点画线算法程序:

```
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);
  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 */
} /* Midpoint_Line */
```


■ 举例

- 用中点画线方法扫描转换连接两点 $P_0(0, 0)$ 和 $P_1(5, 2)$ 的直线段。

x	y	d
0	0	1
1	0	-3
2	1	3
3	1	-1
4	2	5



Bresenham算法

■ 分析

- 设直线方程为

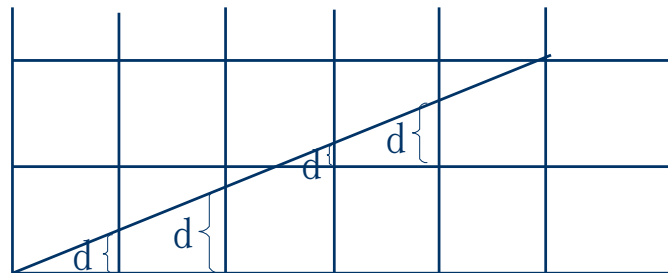
- $y=kx+b$

- 则有 $y_{i+1}=y_i+k(x_{i+1}-x_i)=y_i+k$

- 下一个像素的列坐标为 x_i+1 ，而行坐标要么为 y_i ，要么递增1为 y_i+1 。

- 误差项 d 的初值 $d_0=0$ ， x 坐标每增加1， d 的值相应递增直线的斜率值 k ，即 $d=d+k$ 。

- 是否增1取决于误差项 d 的值。



- 一旦 $d \geq 1$ ，就把它减去1，这样保证 d 在0、1之间。
- 当 $d \geq 0.5$ 时，直线与垂线 $x=x_i+1$ 交点最接近于当前像素 (x_i, y_i) 的右上方像素 (x_i+1, y_i+1)
- 当 $d < 0.5$ 时，更接近于右方像素 (x_i+1, y_i) 。
- 为方便计算，令 $e = d - 0.5$ ， e 的初值为 -0.5 ，增量为 k 。当 $e \geq 0$ 时，取当前像素 (x_i, y_i) 的右上方像素 (x_i+1, y_i+1) ；而当 $e < 0$ 时，取 (x_i, y_i) 右方像素 (x_i+1, y_i) 。

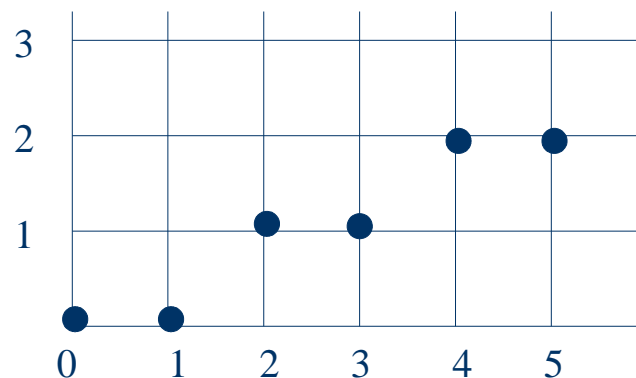
■ Bresenham画线算法程序:

```
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++)
  { drawpixel (x, y, color);
    x++; e=e+k;
    if (e>=0)
      {y++; e=e-1;}
  }
}
```

■ 举例

- 用Bresenham方法扫描转换连接两点 $P_0(0,0)$ 和 $P_1(5,2)$ 的直线段。

x	y	e
0	0	-0.5
1	0	-0.1
2	1	-0.7
3	1	-0.3
4	2	-0.9
5	2	-0.5



■ 改进

- 前面的Bresenham算法在计算直线斜率与误差项时用到小数与除法。
- 可以改用整数以避免除法。
- 由于算法中只用到误差项的符号，因此可将误差项替换为：

$$2*e*dx$$

■ 改进后的Bresenham算法程序

```
void InterBresenhamline (int x0,int y0,int x1, int y1,int color)
{ dx = x1-x0,; dy = y1- y0,; e=-dx;
  x=x0;  y=y0;
  for (i=0; i<=dx; i++)
  {drawpixel (x, y, color);
   x++;  e=e+2*dy;
   if (e>=0)
       { y++; e=e-2*dx;}
  }
}
```