

Chap4 基本图形生成算法

- 线段的绘制
- 多边形绘制
- 反走样技术

图形生成的概念

- 图形的生成：是在指定的输出设备上，根据坐标描述构造二维几何图形。
- 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像素集的过程。

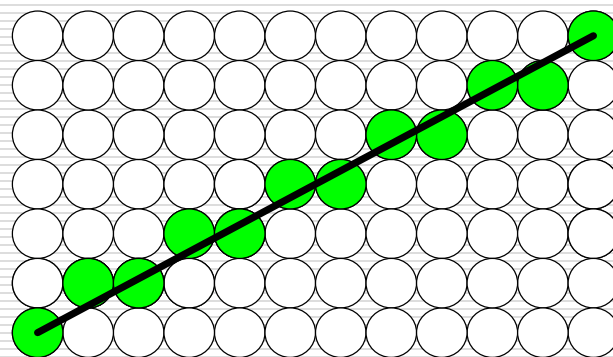


图4.1 用像素点集逼近直线

4.1 线段的绘制

- 直线段生成算法
- 圆的生成算法
- 线段的属性处理

4.1.1 直线的扫描转换

□ 直线的绘制要求

- 直线要直；
- 直线的端点要准确，无定向性无断裂；
- 直线的亮度、色泽要均匀；
- 画线的速度要快；
- 具有不同的色泽、亮度、线型等。

直线的扫描转换

- 解决的问题：给定直线两端点 $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$ ，画出该直线。
- 数值微分法
- 中点Bresenhan算法
- 改进的Bresenhan算法

数值微分法(DDA法)

直线的微分方程:

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

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

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

$$\varepsilon = 1/\max(|\Delta x|, |\Delta y|)$$

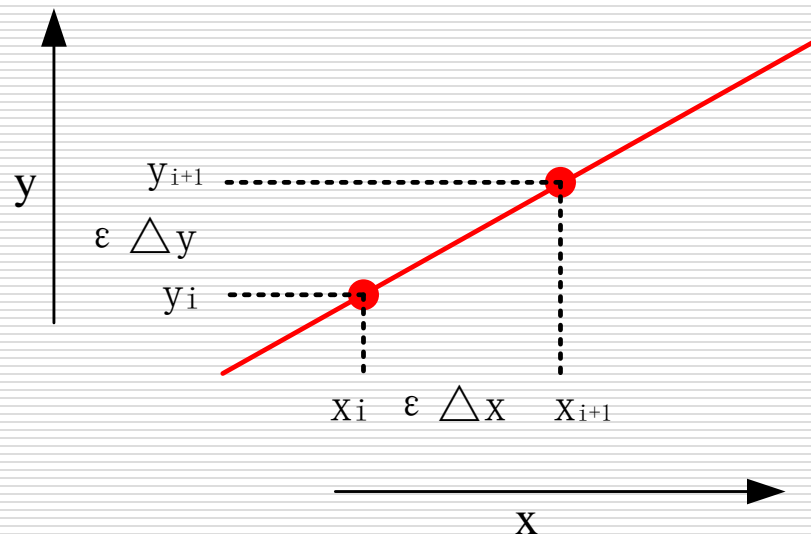


图4.2 DDA算法原理

□ $\max(|\Delta x|, |\Delta y|) = |\Delta x|$, 即 $|k| \leq 1$ 的情况:

$$x_{i+1} = x_i + \varepsilon \cdot \Delta x = x_i + \frac{1}{|\Delta x|} \cdot \Delta x = x_i \pm 1$$

$$y_{i+1} = y_i + \varepsilon \cdot \Delta y = y_i + \frac{1}{|\Delta x|} \cdot \Delta y = y_i \pm k$$

□ $\max(|\Delta x|, |\Delta y|) = |\Delta y|$, 此时 $|k| \geq 1$:

$$x_{i+1} = x_i + \varepsilon \cdot \Delta x = x_i + \frac{1}{|\Delta y|} \cdot \Delta x = x_i \pm \frac{1}{k}$$

$$y_{i+1} = y_i + \varepsilon \cdot \Delta y = y_i + \frac{1}{|\Delta y|} \cdot \Delta y = y_i \pm 1$$

数值微分法(DDA法)

□ 特点

- 增量算法
- 直观、易实现
- 不利于用硬件实现

中点 Bresenham 算法

□ 直线的方程

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

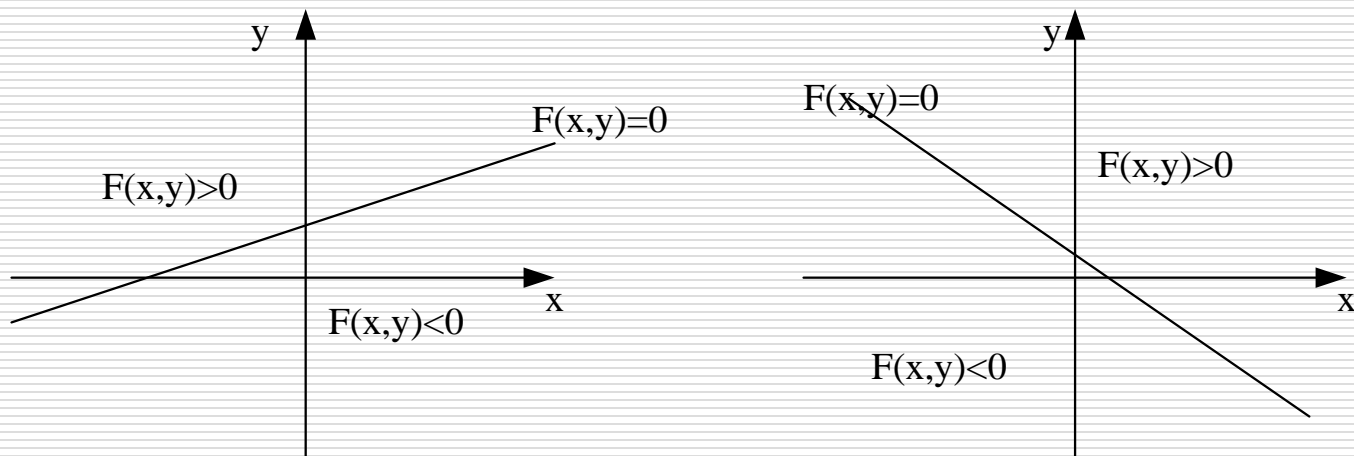


图4.3 直线将平面分为三个区域

中点 Bresenham 算法

- 算法原理：根据直线的斜率确定或选择变量在 x 或 y 方向上每次递增一个单位，而另一方向的增量为 1 或 0，它取决于实际直线与相邻像素点的距离，这一距离称为**误差项**。

假定 $0 \leq k \leq 1$,

即 $0 \leq \Delta y / \Delta x \leq 1$, x 是最大位移方向

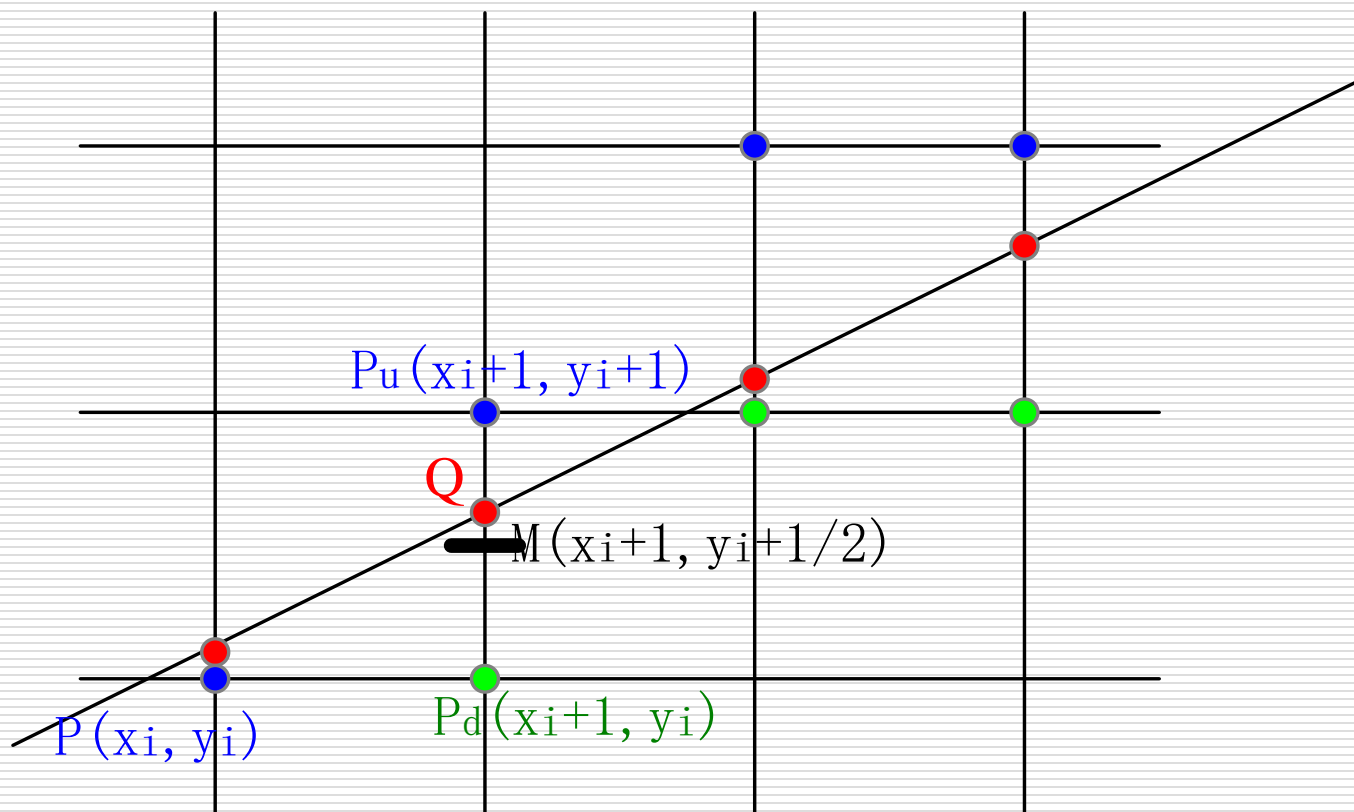


图4.4 Bresenham算法生成直线的原理

判别式:

$$d = F(x_M, y_M) = F(x_i + 1, y_i + 0.5) = y_i + 0.5 - k(x_i + 1) - b$$

则有:

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

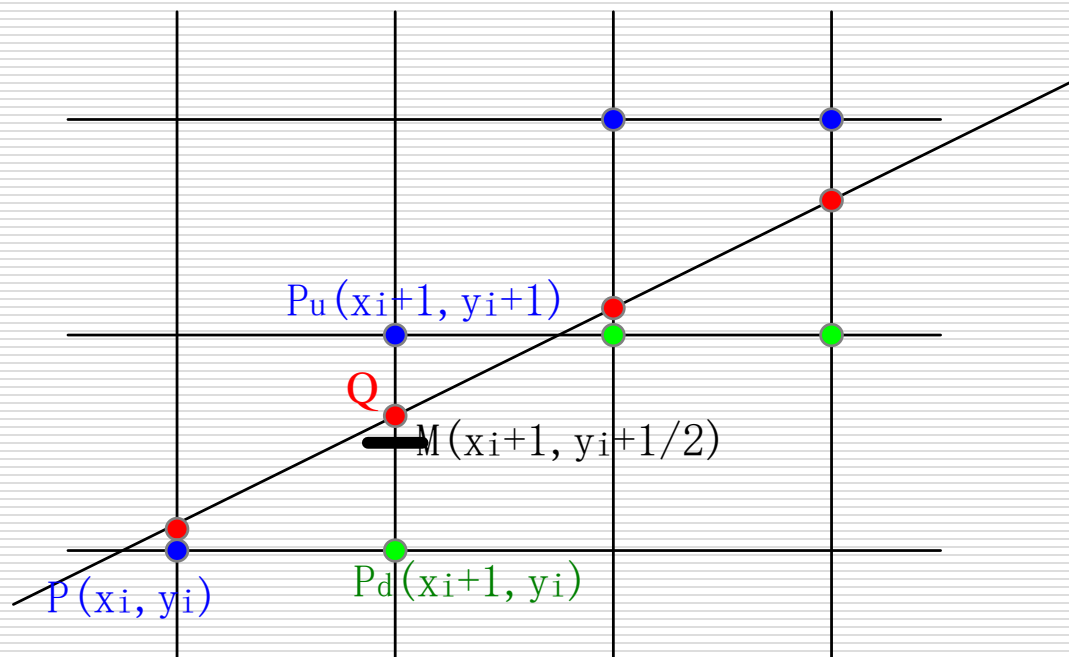


图4.5 Bresenham算法生成直线的原理

d的意义:

$$d = F(x_M, y_M) = F(x_i + 1, y_i + 0.5) = y_i + 0.5 - k(x_i + 1) - b$$

$$d = F(x_M, y_M) - F(x_Q, y_Q)$$

$$= y_M - kx_M - b - (y_Q - kx_Q - b)$$

$$= y_M - y_Q$$

$$d > 0 \quad y_M > y_Q$$

$$d < 0 \quad y_M < y_Q$$

$$d = 0 \quad y_M = y_Q$$

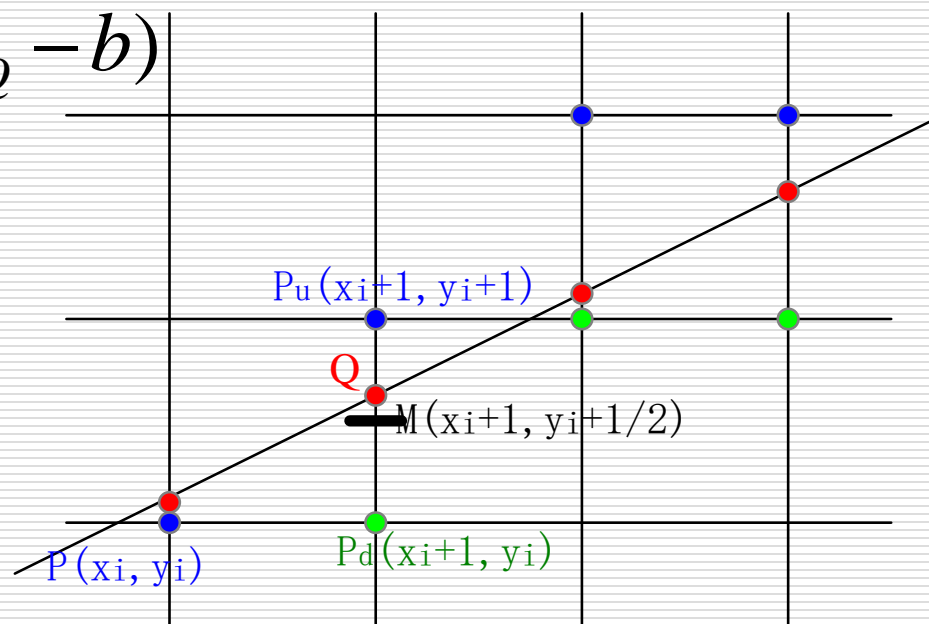


图4.6 Bresenham算法生成直线的原理

误差项的递推 ($d < 0$)

$$\begin{aligned} d_1 &= F(x_i + 1, y_i + 0.5) \\ &= y_i + 0.5 - k(x_i + 1) - b \end{aligned}$$

$$\begin{aligned} d_2 &= F(x_i + 2, y_i + 1.5) \\ &= y_i + 1.5 - k(x_i + 2) - b \end{aligned}$$

$$\begin{aligned} d_2 &= y_i + 0.5 - k(x_i + 1) - b + 1 - k \\ &= d_1 + 1 - k \end{aligned}$$

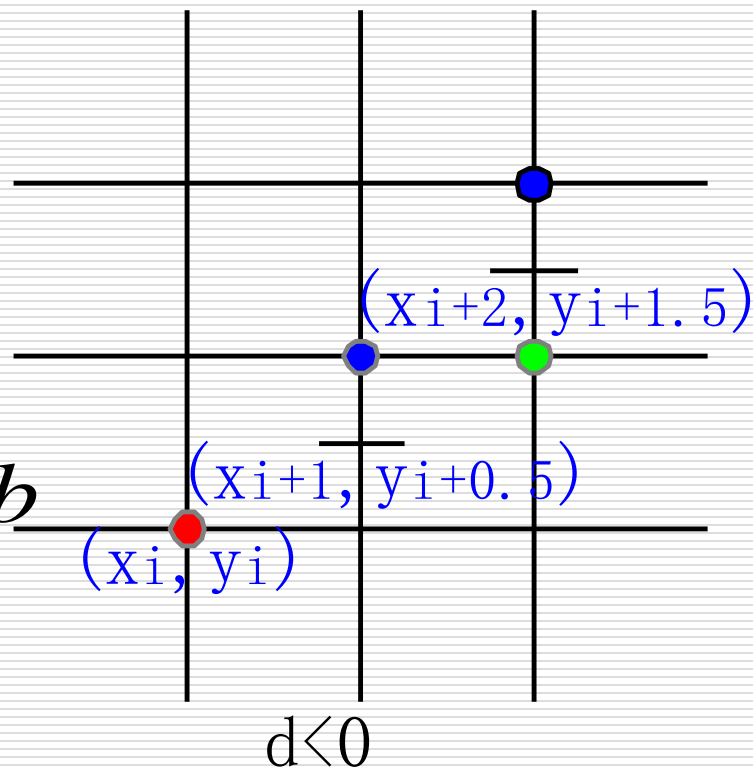


图4.7 误差项递推

误差项的递推 ($d \geq 0$)

$$\begin{aligned}d_1 &= F(x_i + 1, y_i + 0.5) \\&= y_i + 0.5 - k(x_i + 1) - b\end{aligned}$$

$$\begin{aligned}d_2 &= F(x_i + 2, y_i + 0.5) \\&= y_i + 0.5 - k(x_i + 2) - b\end{aligned}$$

$$\begin{aligned}d_2 &= y_i + 0.5 - k(x_i + 1) - b - k \\&= d_1 - k\end{aligned}$$

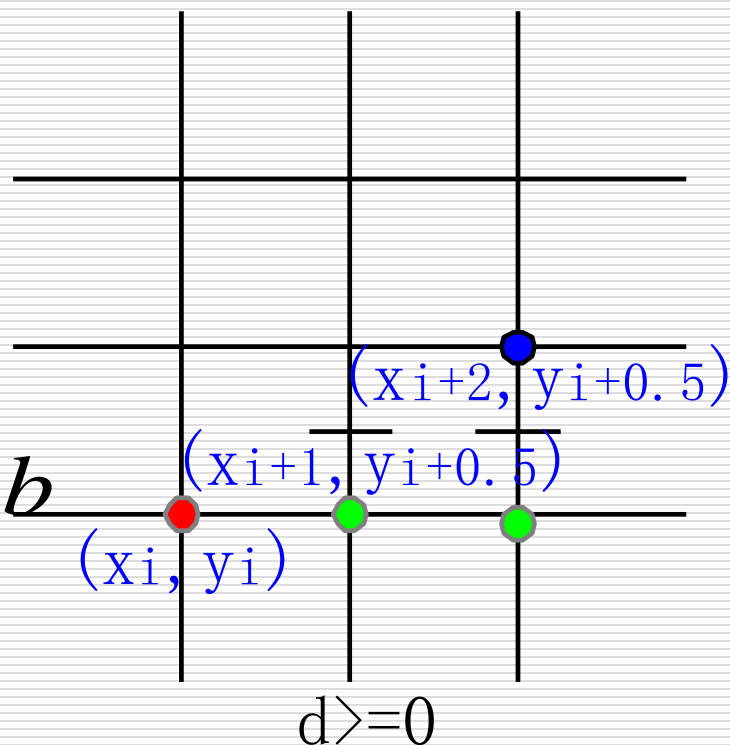


图4.8 误差项递推

中点 Bresenham 算法

初始值 d 的计算

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

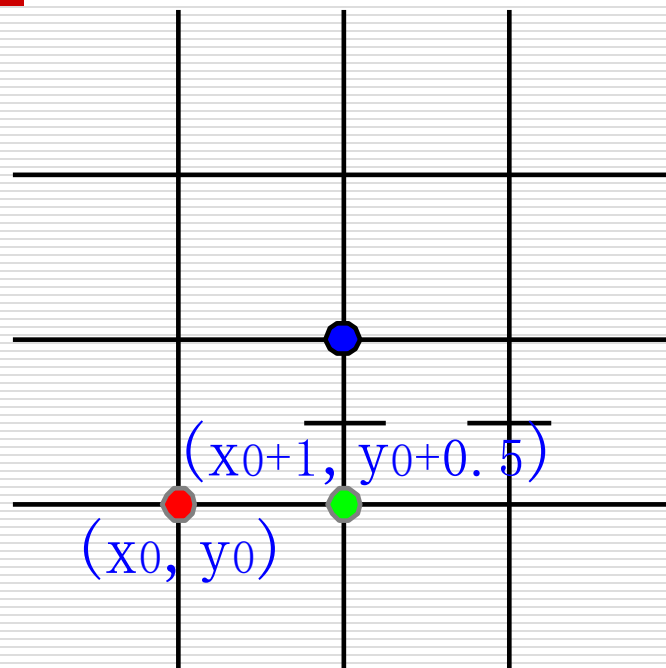


图4.9 计算初值

中点Bresenham算法

改进：用 $2d_{\Delta x}$ 代替 d ，令 $D=2d\Delta x$ 则：

$$D_0 = 2\Delta x d_0 = 2\Delta x(0.5 - k) = \Delta x - 2\Delta y$$

$$d < 0 \quad D < 0$$

$$D = 2\Delta x d = 2\Delta x(d + 1 - k)$$

$$= 2\Delta x d + 2\Delta x - 2\Delta y$$

$$= D + 2\Delta x - 2\Delta y$$

$$d \geq 0 \quad D \geq 0$$

$$D = 2\Delta x d = 2\Delta x(d - k) = D - 2\Delta y$$

中点Bresenham算法——算法步骤

- 输入直线的两端点 $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$ 。
- 计算初始值 Δx 、 Δy 、 $D = \Delta x - 2\Delta y$ 、 $x = x_0$ 、 $y = y_0$ 。
- 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 $D + 2\Delta x - 2\Delta y$ ；否则 (x, y) 更新为 $(x+1, y)$ ， D 更新为 $D - 2\Delta y$ 。
- 当直线没有画完时，重复上一步骤，否则结束。

改进的 Bresenham 算法

假定直线段的 $0 \leq k \leq 1$

$$x_{i+1} = x_i + \varepsilon \cdot \Delta x = x_i + \frac{1}{|\Delta x|} \cdot \Delta x = x_i + 1$$

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

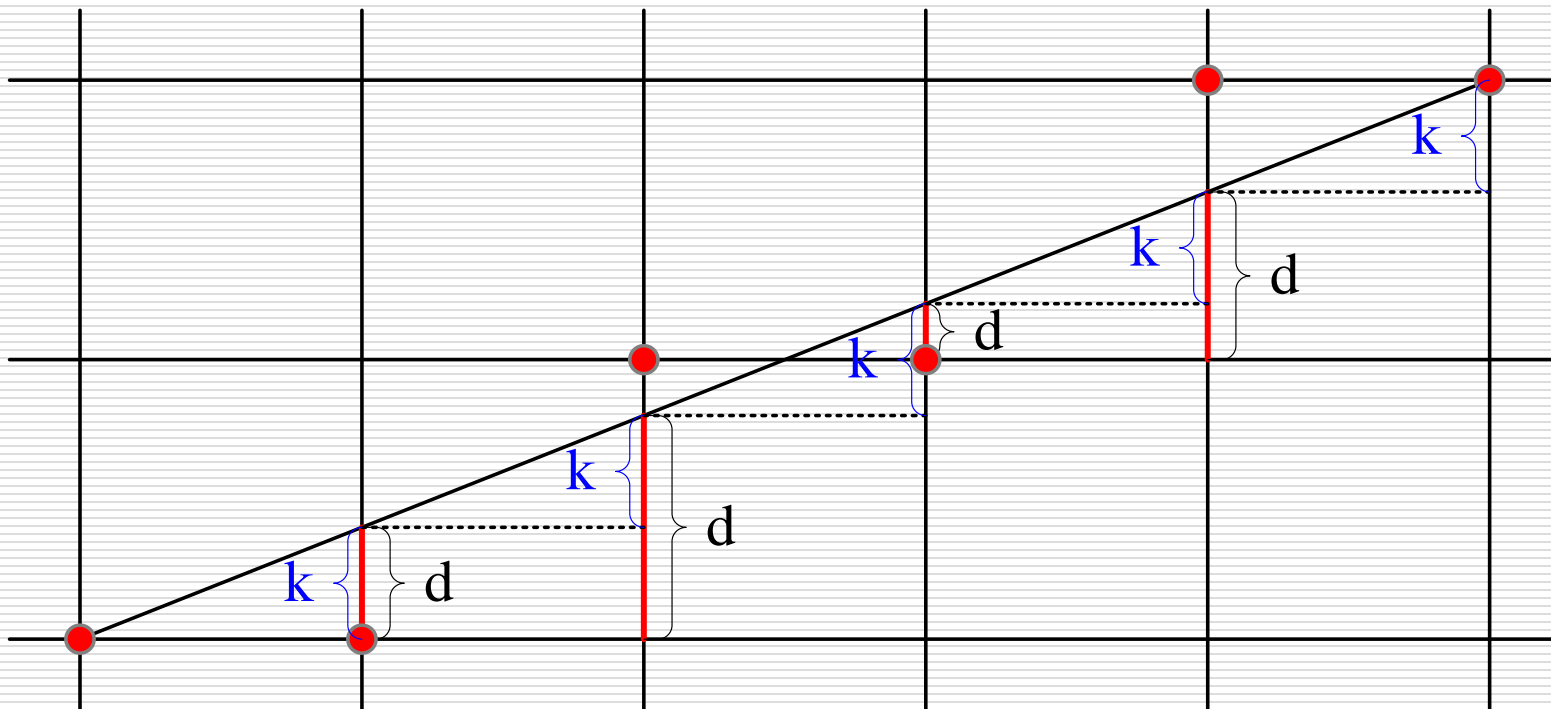


图4.10 改进的Brensemham算法绘制直线的原理

改进的 Bresenham 算法——原理

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

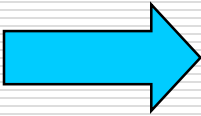
误差项的计算

- $d_{\text{初}} = 0$,
- 每走一步: $d = d + k$
- 一旦 y 方向上走了一步, $d = d - 1$

改进1: 令 $e=d-0.5$

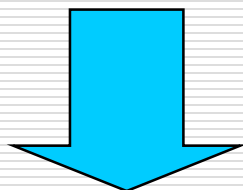
$$\left\{ \begin{array}{l} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (d > 0.5) \\ y_i & (d \leq 0.5) \end{cases} \end{array} \right. \longrightarrow \left\{ \begin{array}{l} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (e > 0) \\ y_i & (e \leq 0) \end{cases} \end{array} \right.$$

误差项的计算

- $d_{\text{初}} = 0$,
 - 每走一步: $d = d + k$
 - if ($e > 0$) then $d = d - 1$
- 
- $e_{\text{初}} = -0.5$,
 - 每走一步有 $e = e + k$ 。
 - if ($e > 0$) then $e = e - 1$

改进2：用 $E=2e\Delta x$ 来替换 e

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



- $E_{\text{初}} = -0.5 * 2\Delta x = -\Delta x$
- 每走一步有 $E = (e + k) * 2\Delta x = E + 2\Delta y$
- if $(e > 0)$ then $E = (e - 1) * 2\Delta x = E - 2\Delta x$

改进的 Bresenham 算法——算法步骤

1. 输入直线的两 endpoint $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$ 。
2. 计算初始值 Δx 、 Δy 、 $e = -\Delta x$ 、 $x = x_0$ 、 $y = y_0$ 。
3. 绘制点 (x, y) 。
4. e 更新为 $e + 2\Delta y$ ，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 $e - 2\Delta x$ ；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。

4.1.2 圆的扫描转换

- 解决的问题：绘出圆心在 origin，半径为整数 R 的圆 $x^2 + y^2 = R^2$ 。

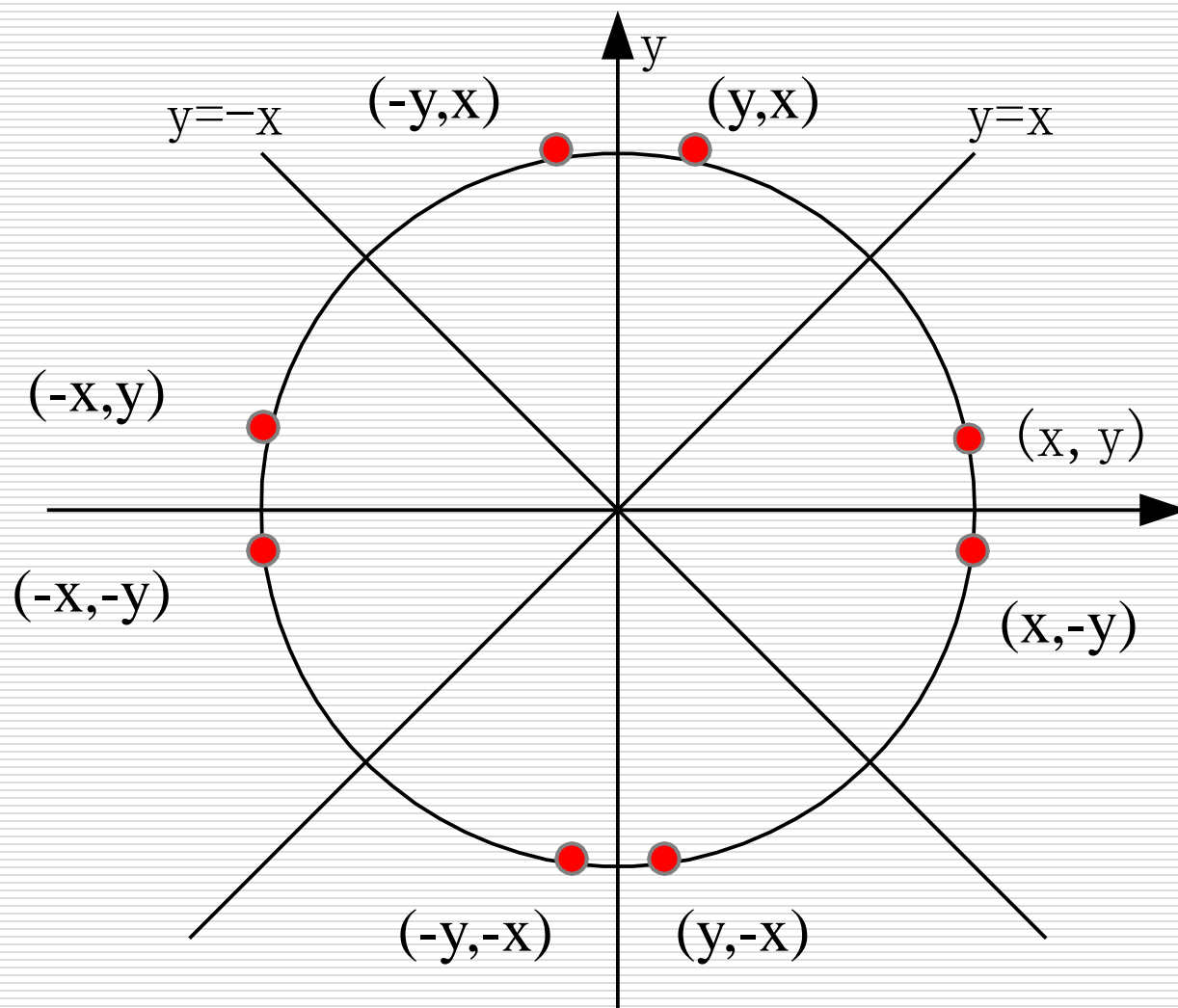


图4.11 八分法画圆

圆的扫描转换

解决问题:

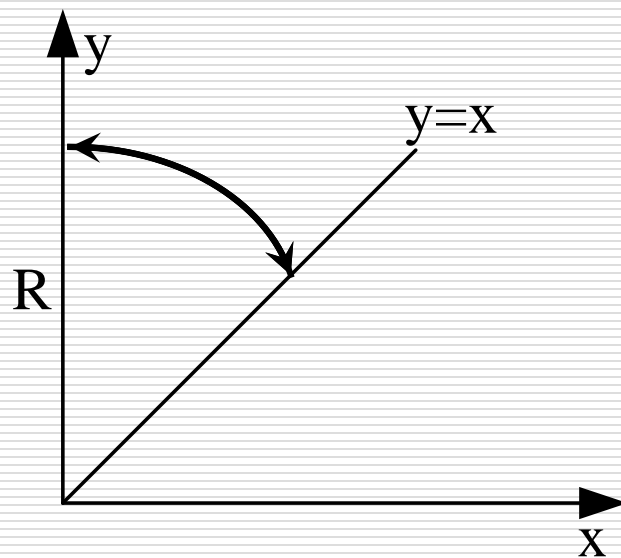


图4.12 1/8圆弧

圆的扫描转换

- 简单方程生成圆弧
- 中点**Bresenham**算法

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： $x^2 + y^2 = R^2$

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

$$y_{i+1} = \text{round}(\sqrt{R^2 - x_{i+1}^2}) \quad (5-7)$$

简单方程产生圆弧

圆的极坐标方程为：

$$x = R \cos \theta$$

$$y = R \sin \theta$$

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

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

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

中点Bresenham画圆

给定圆心在 origin，半径为整数 R 的圆，其方程为

$$x^2 + y^2 = R^2$$

构造函数 $F(x,y) = x^2 + y^2 - R^2$ 。

- 对于圆上的点，有 $F(x,y) = 0$;
- 对于圆外的点， $F(x,y) > 0$;
- 而对于圆内的点， $F(x,y) < 0$ 。

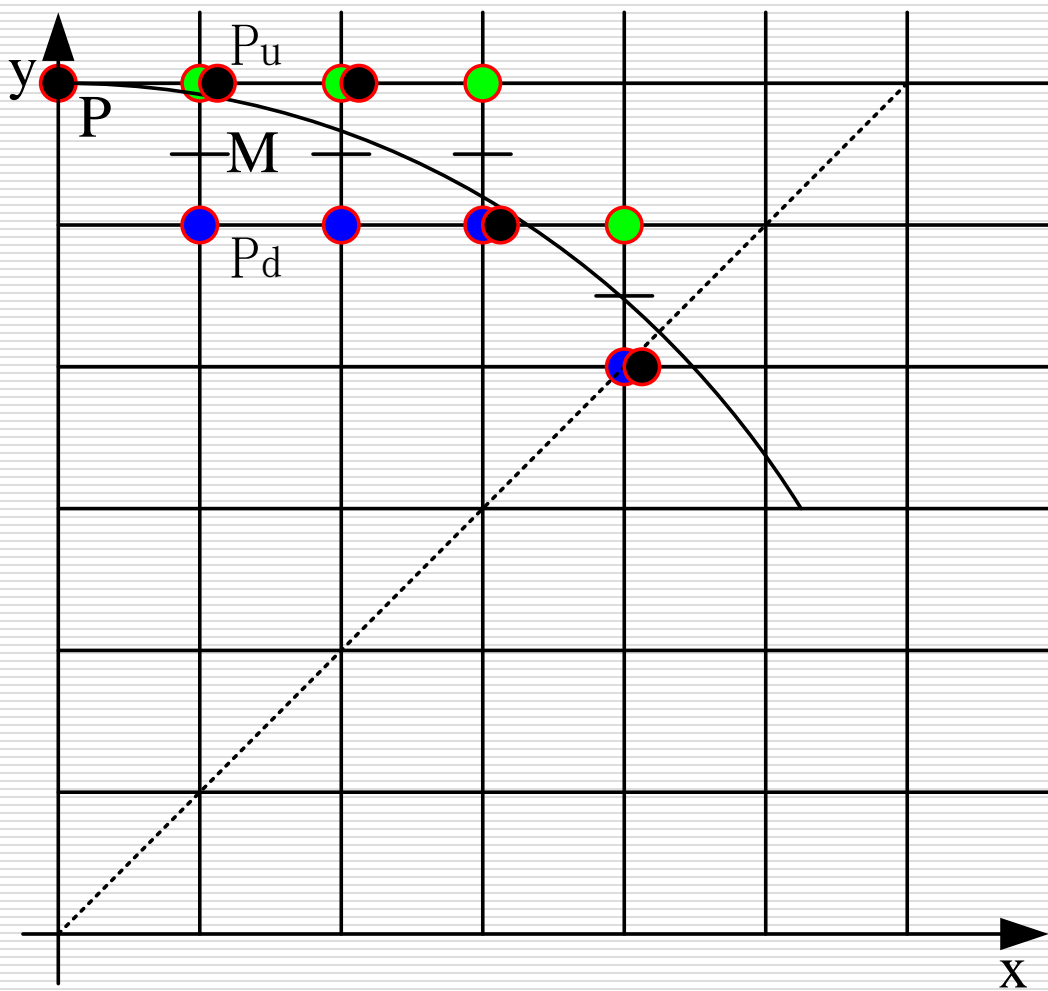


图4.13 中点Bresenham画圆的原理

中点Bresenham画圆

构造判别式:

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

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

误差项的递推 ($d \leq 0$)

$$\begin{aligned} d_2 &= F(x_i + 2, y_i - 0.5) \\ &= (x_i + 2)^2 + (y_i - 0.5)^2 - R^2 \end{aligned}$$

$$\begin{aligned} d_2 &= (x_i + 1 + 1)^2 + (y_i - 0.5)^2 - R^2 \\ &= (x_i + 1)^2 + 2x_i + 3 + (y_i - 0.5)^2 - R^2 \\ &= d_1 + 2x_i + 3 \end{aligned}$$

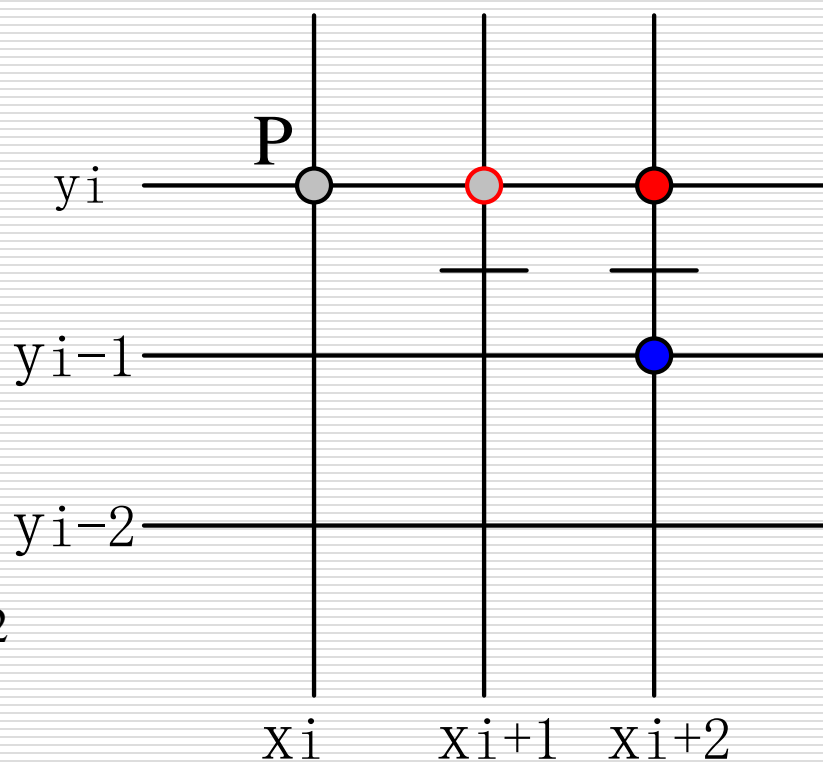


图4.14 $d \leq 0$ 的情况

误差项的递推 ($d > 0$)

$$\begin{aligned} d_2 &= F(x_i + 2, y_i - 1.5) \\ &= (x_i + 2)^2 + (y_i - 1.5)^2 - R^2 \end{aligned}$$

$$d_2 = (x_i + 1 + 1)^2 + (y_i - 0.5 - 1)^2 - R^2$$

$$= (x_i + 1)^2 + 2x_i + 3 + (y_i - 0.5)^2 - 2(y_i - 0.5) + 1 - R^2$$

$$= (x_i + 1)^2 + (y_i - 0.5)^2 + 2(x_i - y_i) + 5$$

$$= d_1 + 2(x_i - y_i) + 5$$

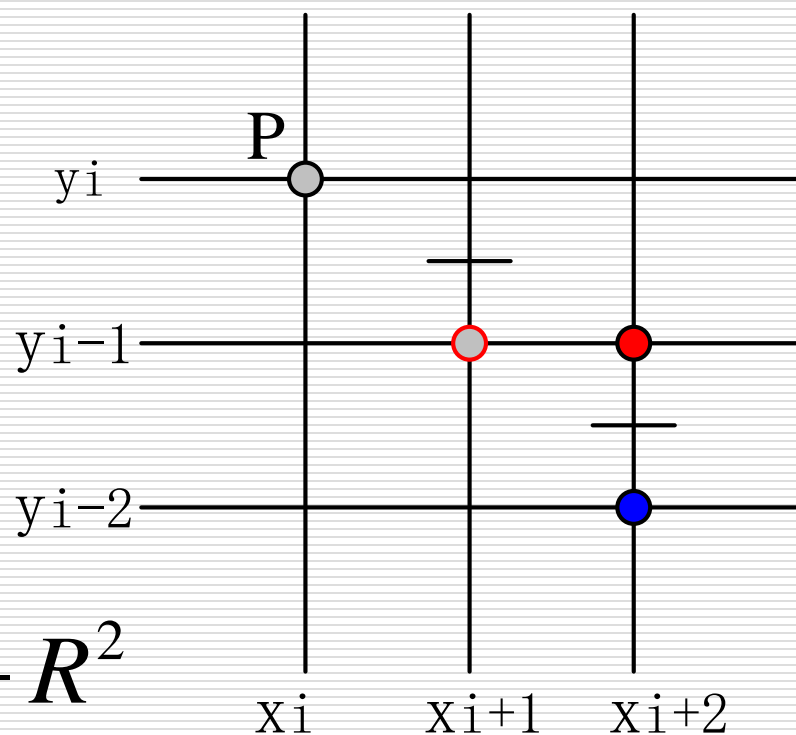


图4.15 $d > 0$ 的情况

中点Bresenham画圆

判别式的初始值

$$\begin{aligned}d_0 &= F(x_0 + 1, y_0 - 0.5) \\&= F(1, R - 0.5) \\&= 1 + (R - 0.5)^2 - R^2 \\&= 1.25 - R\end{aligned}$$

改进：用 **$d-0.25$** 代替 **d**

此时有：

$$d = d + 2x_i + 3 \quad d \leq -0.25$$

$$d = d + 2(x_i - y_i) + 5 \quad d > -0.25$$

$$d_0 = 1 - R$$

$$d \leq -0.25$$



$$d < 0$$

$$d > -0.25$$

$$d \geq 0$$

中点 Bresenham 画圆

□ 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。

4.1.3 线段的属性处理

- 图素或图段的外观由其属性决定。
- 图形软件中实现属性选择的方法是提供一张系统**当前属性值表**，通过调用标准函数提供属性值表的设置和修改。进行扫描转换时，系统使用属性值表中的当前属性值进行显示和输出。

线段属性——线型

□ 用像素模板(pixel mask)实现

- 问题：如何保持任何方向的划线长度近似地相等
- 根据线的斜率来调整像素数目

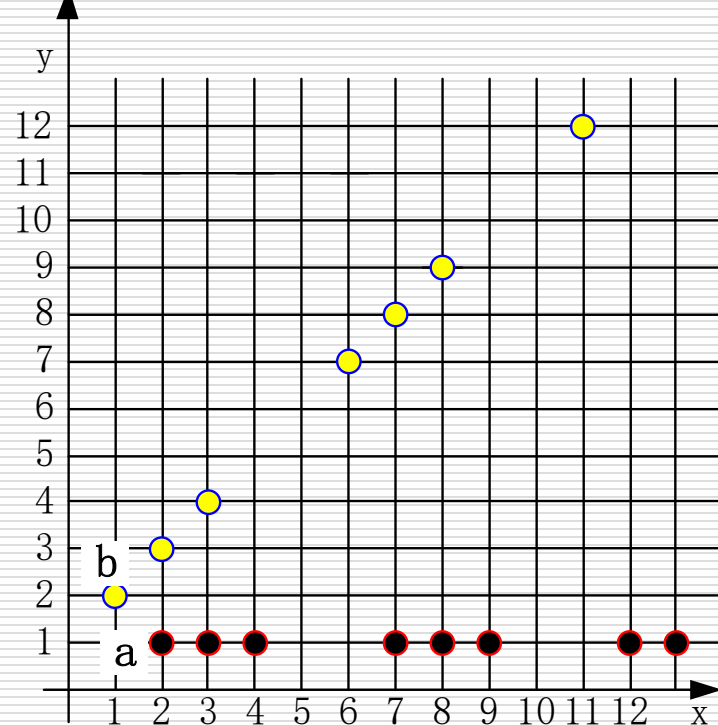


图4.16 相同数目像素显示的不等长划线

线段属性——线宽

□ 线刷子：垂直刷子、水平刷子

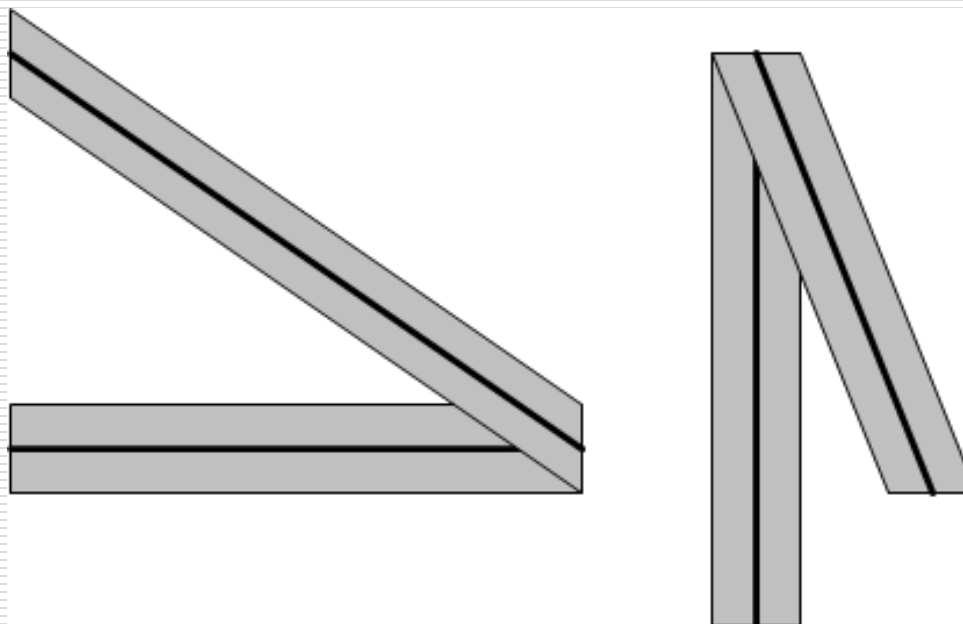


图4.17 线刷子

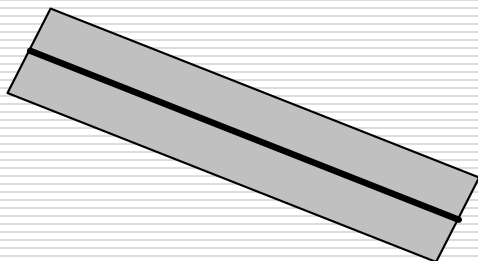
线段属性——线宽

□ 线刷子特点

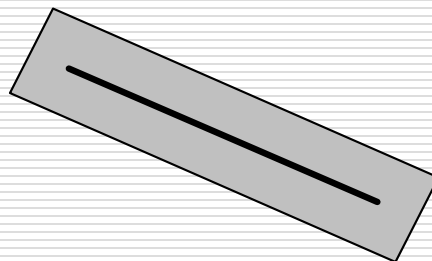
- 实现简单、效率高。
- 斜线与水平(或垂直)线不一样粗。
- 当线宽为偶数个像素时，线的中心将偏移半个像素。
- 利用线刷子生成线的始末端总是水平或垂直的，看起来不太自然。

线段属性——线宽

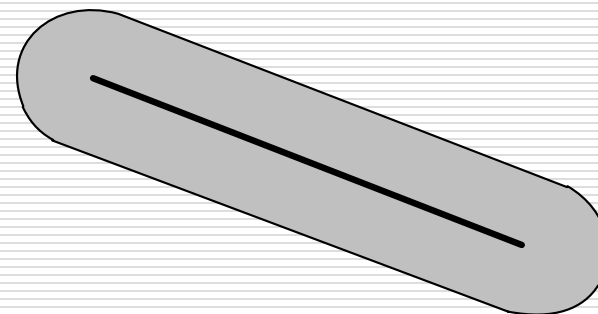
□ 线帽 (line cap)



(a) 方帽



(b) 凸方帽



(c) 圆帽

图4.18 线帽

线段属性——线宽

- 接近水平的线与接近垂直的线相交时，汇合处外角将有缺口

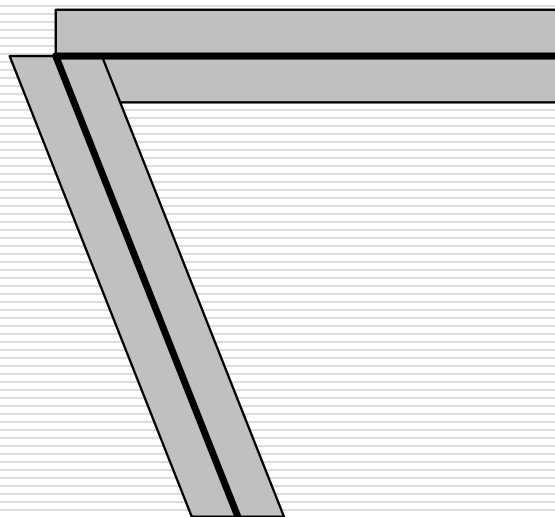
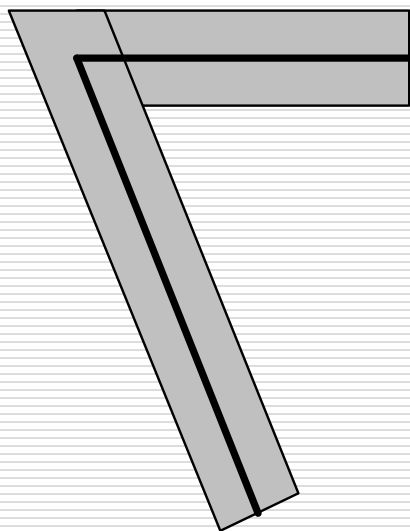


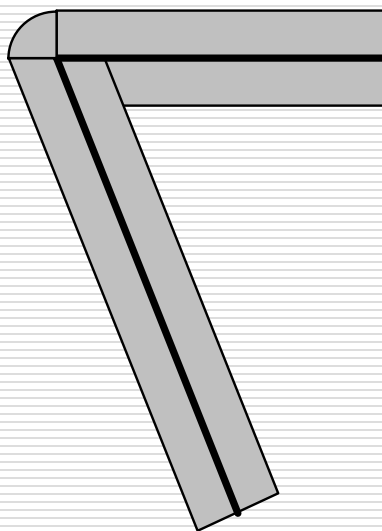
图4.19 线刷子的缺口

线段属性——线宽

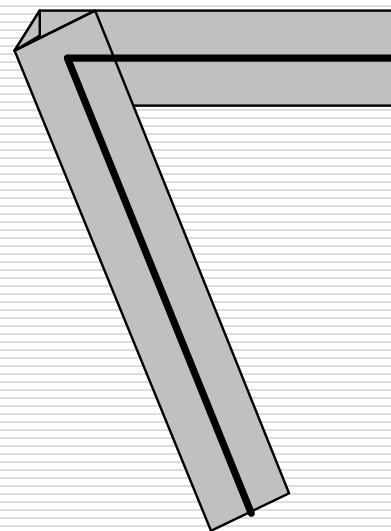
□ 斜角连接（**miter join**）、圆连接（**round join**）、斜切连接（**bevel join**）



(a) 斜角连接



(b) 圆连接



(c) 斜切连接

图4.20 线刷子缺口连接

线段属性——线宽

□ 方刷子

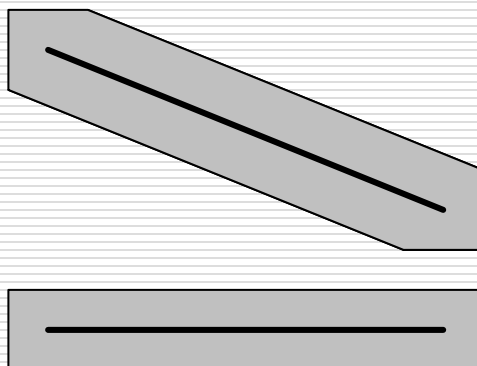


图4.21 方刷子

线段属性——线宽

□ 方刷子特点

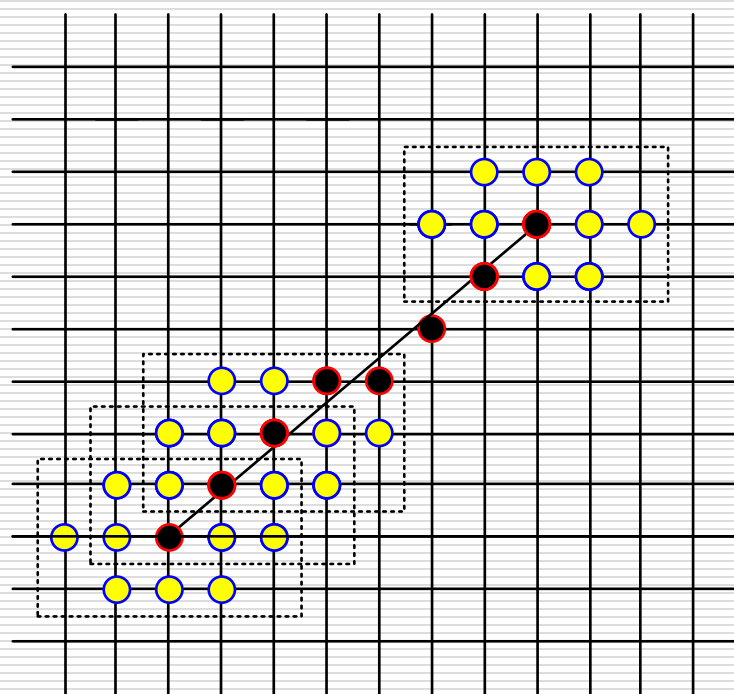
- 绘制的线条（斜线）比用线刷子所绘制的线条要粗一些
- 绘制的斜线与水平（或垂直）线不一样粗
- 绘制的线条自然地带有有一个“方线帽”

线段属性——线宽

□ 模板刷子

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

(a) 像素模板



(b) 用该模板进行线宽处理

图4.22 利用像素模板进行线宽处理

4.2 多边形绘制

- 多边形的扫描转换
- 区域填充
- 多边形扫描转换与区域填充的比较
- 内外测试

4.2.1 多边形的扫描转换

- 顶点表示用多边形的顶点序列来刻画多边形。
- 点阵表示是用位于多边形内的像素的集合来刻画多边形。
- 扫描转换多边形：从多边形的顶点信息出发，求出位于其内部的各个像素，并将其颜色值写入帧缓存中相应单元的过程。

多边形的扫描转换

- X-扫描线算法
- 改进的有效边表算法
- 边缘填充算法

X-扫描线算法——原理

- 基本思想：按扫描线顺序，计算扫描线与多边形的相交区间，再用要求的颜色显示这些区间的所有像素。

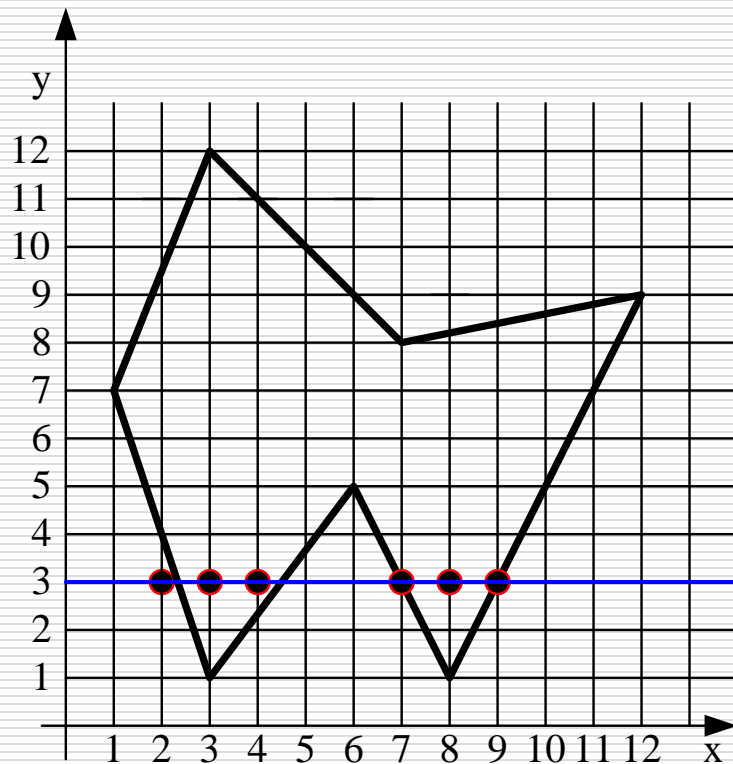


图5.23 x-扫描线算法填充多边形

X-扫描线算法——算法步骤

1. 确定多边形所占有的最大扫描线数，得到多边形顶点的最小和最大 y 值（ y_{\min} 和 y_{\max} ）。
2. 从 $y=y_{\min}$ 到 $y=y_{\max}$ ，每次用一条扫描线进行填充。
3. 对一条扫描线填充的过程可分为四个步骤：
求交；排序；交点配对；区间填色。

X-扫描线算法——取整规则

- 交点的取整规则：使生成的像素全部位于多边形之内。（用于直线等图元扫描转换的四舍五入原则可能导致部分像素位于多边形之外，从而不可用）。
- 假定非水平边与扫描线 $y=e$ 相交，交点的横坐标为 x ，规则如下：

□规则1： x 为小数，即交点落于扫描线上两个相邻像素之间时：

- 交点位于左边界之上，向右取整；
- 交点位于右边界之上，向左取整；

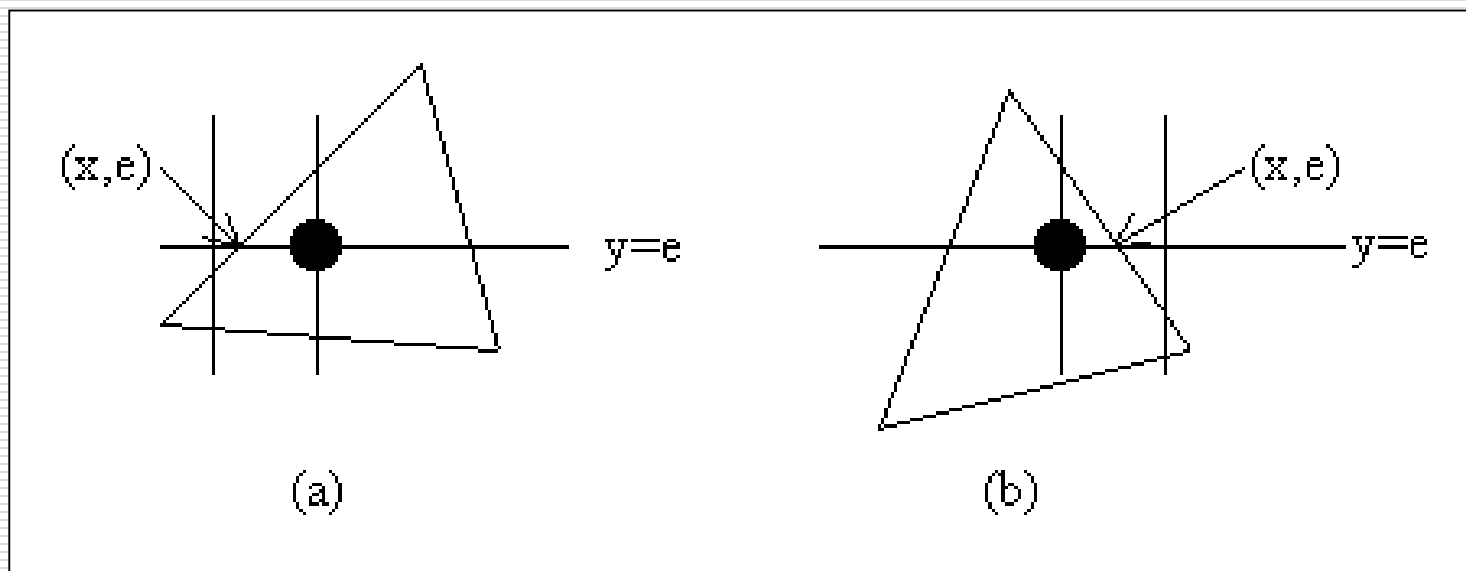


图5.24 取整规则1

□ 规则2：边界上像素的取舍问题，避免填充扩大化。
规定落在右边边界上的像素不予填充。（具体实现时，只要对扫描线与多边形的相交区间左闭右开）

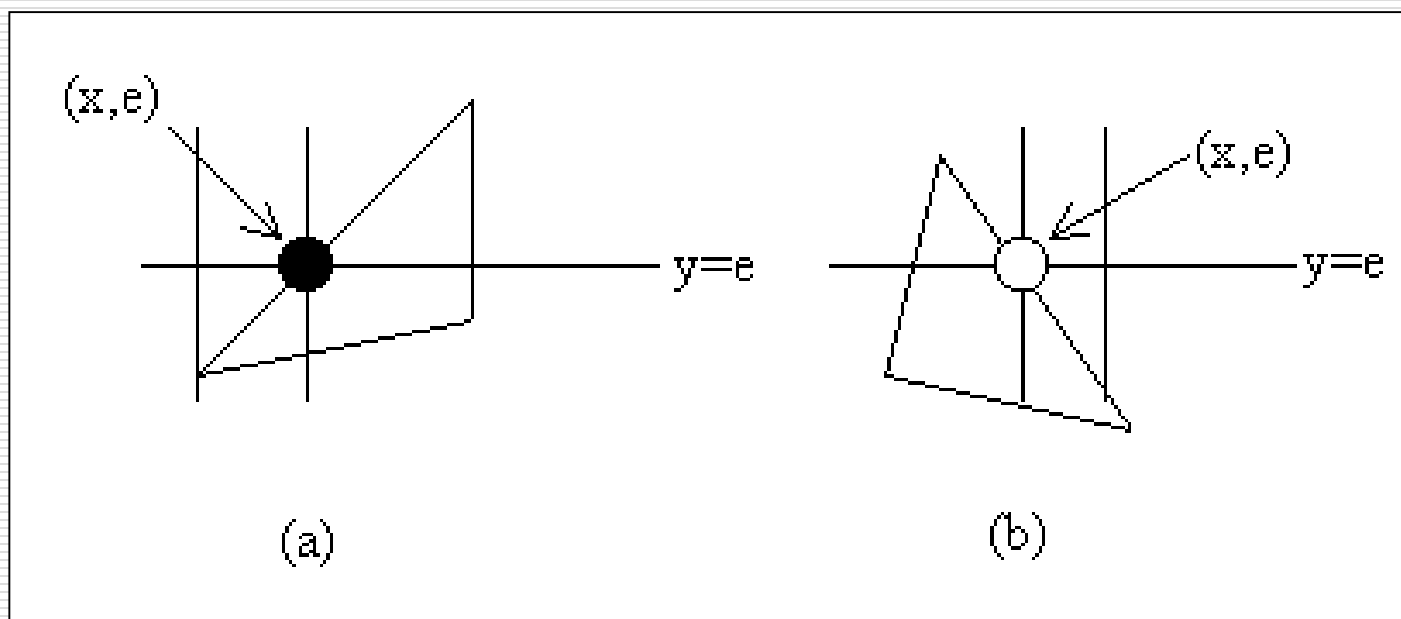


图5.25 取整规则2

□ **规则3**：当扫描线与多边形顶点相交时，交点的取舍，保证交点正确配对。

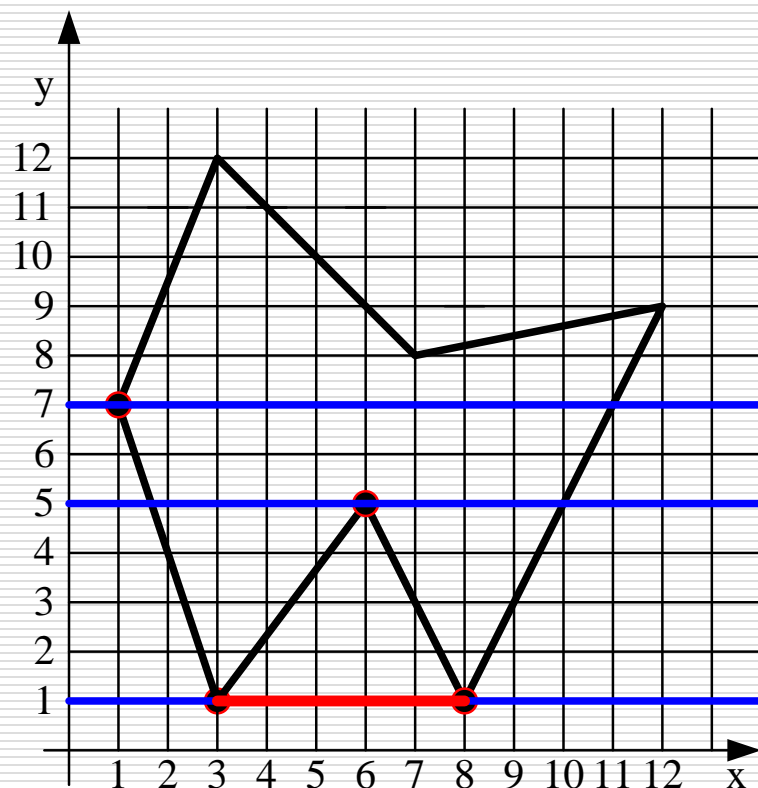


图5.26 取整规则3

X-扫描线算法——取整规则

解决方法:

当扫描线与多边形的顶点相交时,

- 若共享顶点的两条边分别落在扫描线的两边, 交点只算一个;
- 若共享顶点的两条边在扫描线的同一边, 这时交点作为零个或两个。

X-扫描线算法——取整规则

实际处理：只要检查顶点的两条边的另外两个端点的 y 值，两个 y 值中大于交点 y 值的个数是0, 1, 2，来决定取0, 1, 2个交点。

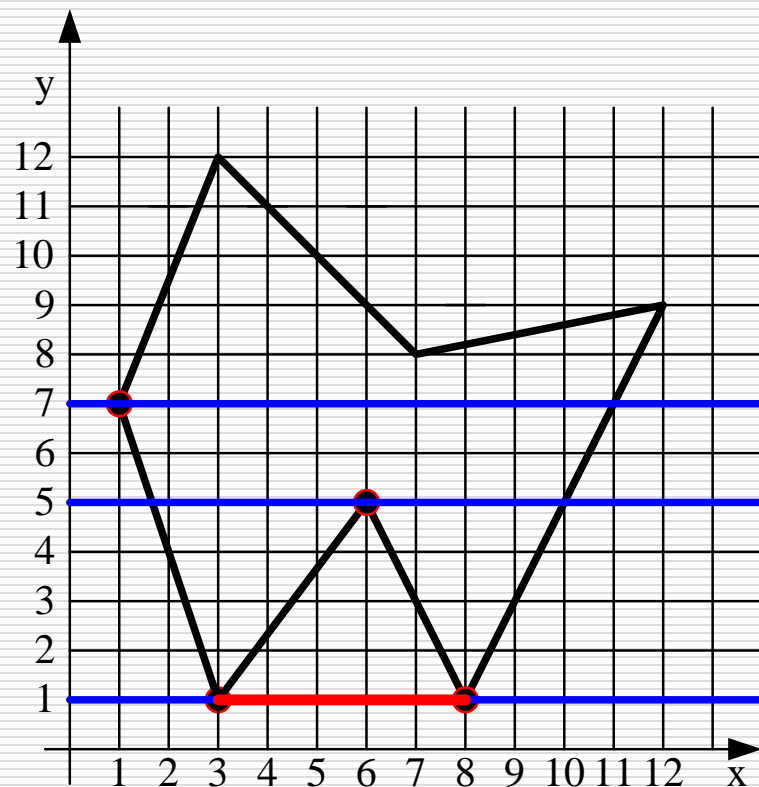


图5.27 取整规则3

X-扫描线算法——取整规则

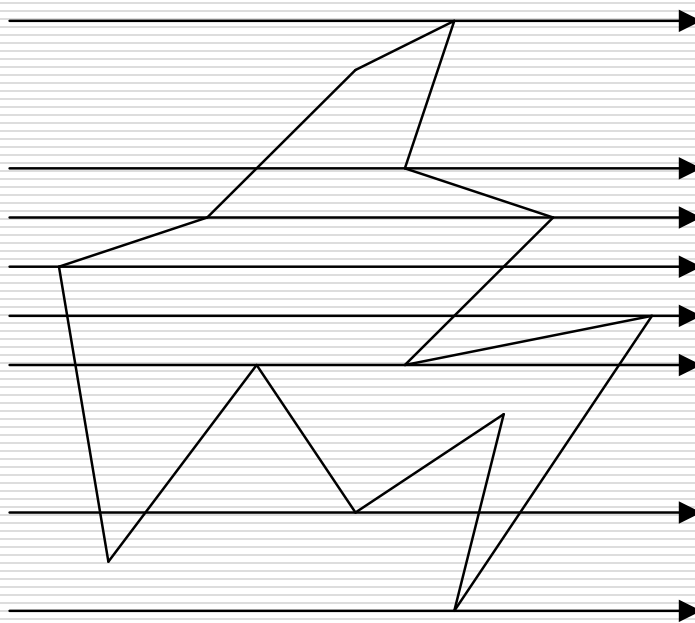


图5.28 与扫描线相交的多边形顶点的交点数

改进的有效边表算法 (Y连贯性算法)

改进原理:

- 处理一条扫描线时，仅对有效边求交。
- 利用扫描线的连贯性。
- 利用多边形边的连贯性。

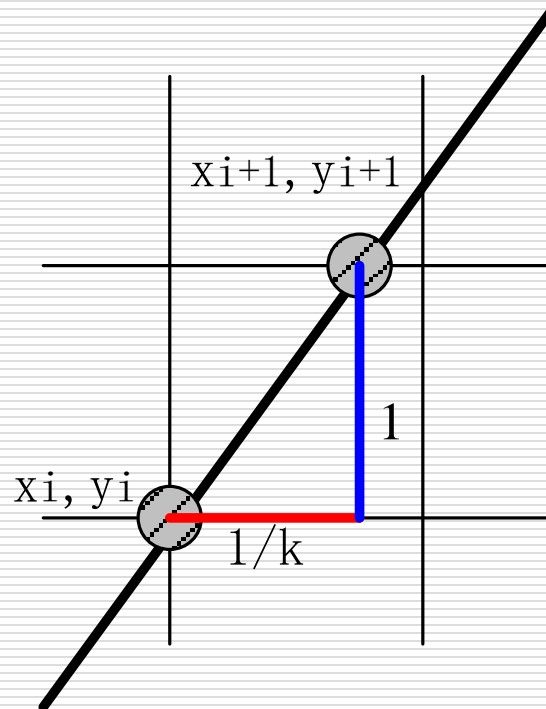


图5.29 与多边形边界相交的两条连续扫描线交点的相关性

改进的有效边表算法 (Y连贯性算法)

- 有效边 (**Active Edge**)：指与当前扫描线相交的多边形的边，也称为活性边。
- 有效边表 (**Active Edge Table, AET**)：把有效边按与扫描线交点 x 坐标递增的顺序存放在一个链表中，此链表称为有效边表。
- 有效边表的每个结点：

x y_{\max} $1/k$ next

改进的有效边表算法——构造边表

- 首先构造一个纵向链表，链表的长度为多边形所占有的最大扫描线数，链表的每个结点，称为一个桶，则对应多边形覆盖的每一条扫描线。
- 将每条边的信息链入与该边最小 y 坐标（ y_{\min} ）相对应的桶处。也就是说，若某边的较低端点为 y_{\min} ，则该边就放在相应的扫描线桶中。

改进的有效边表算法——构造边表

- 每条边的数据形成一个结点，内容包括：该扫描线与该边的初始交点 x （即较低端点的 x 值）， $1/k$ ，以及该边的最大 y 值 y_{\max} 。

$x|_{y_{\min}}$ y_{\max} $1/k$ NEXT

- 同一桶中若干条边按 $x|_{y_{\min}}$ 由小到大排序，若 $x|_{y_{\max}}$ 相等，则按照 $1/k$ 由小到大排序。

解决顶点交点计为1时的情形：

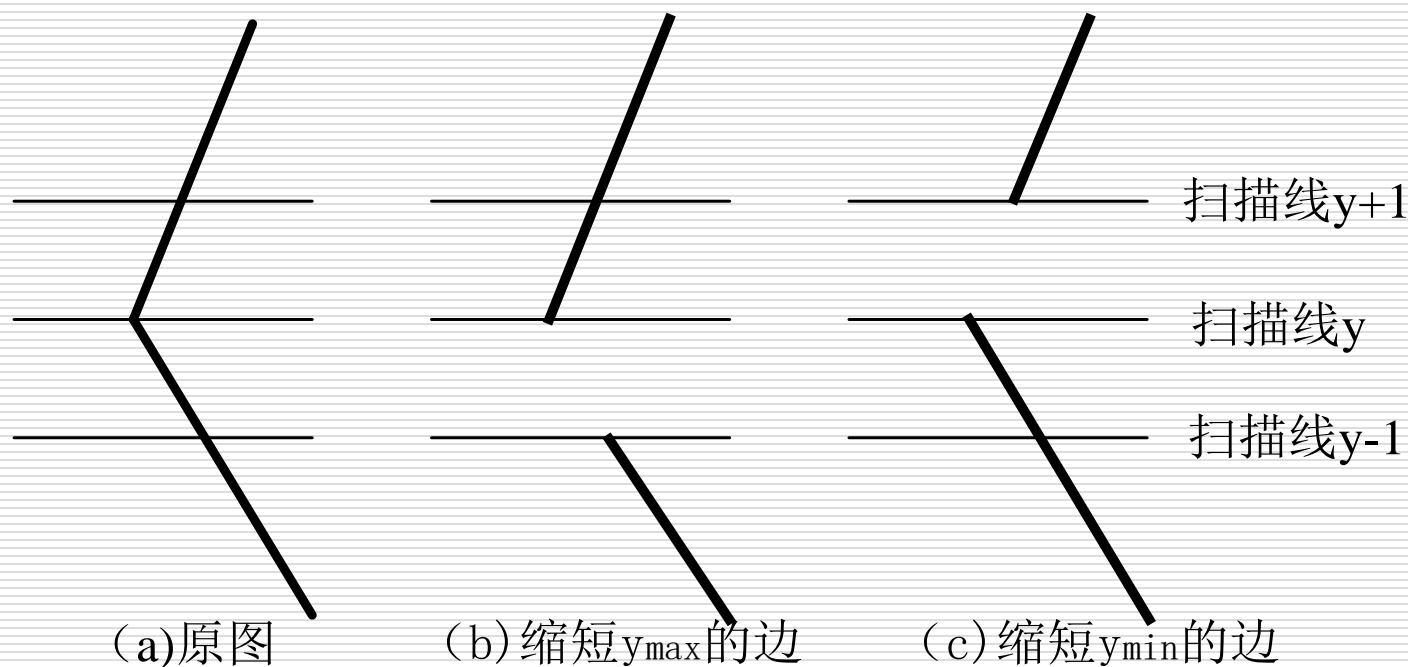


图5-30 将多边形的某些边缩短以分离那些应计为1个交点的顶点

p3p2	3	6	$-1/3$	
p3p4	3	5	$3/4$	
p5p4	8	5	$-1/2$	
p5p6	8	9	$1/2$	
p2p1	1	12	$2/5$	
p0p1	7	12	-1	
p0p6	7	9	5	

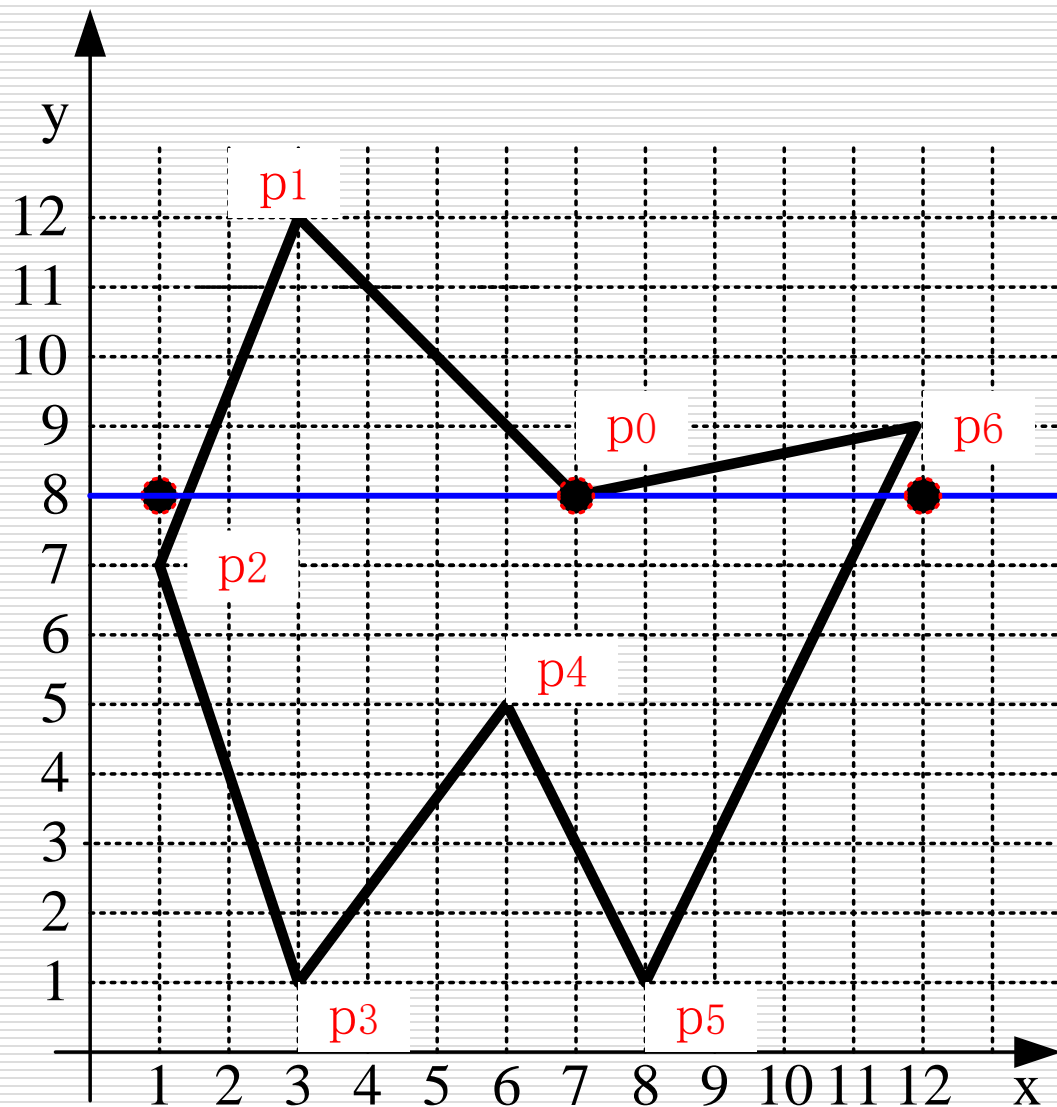


图5.31 多边形 $P_0P_1P_2P_3P_4P_5P_6$

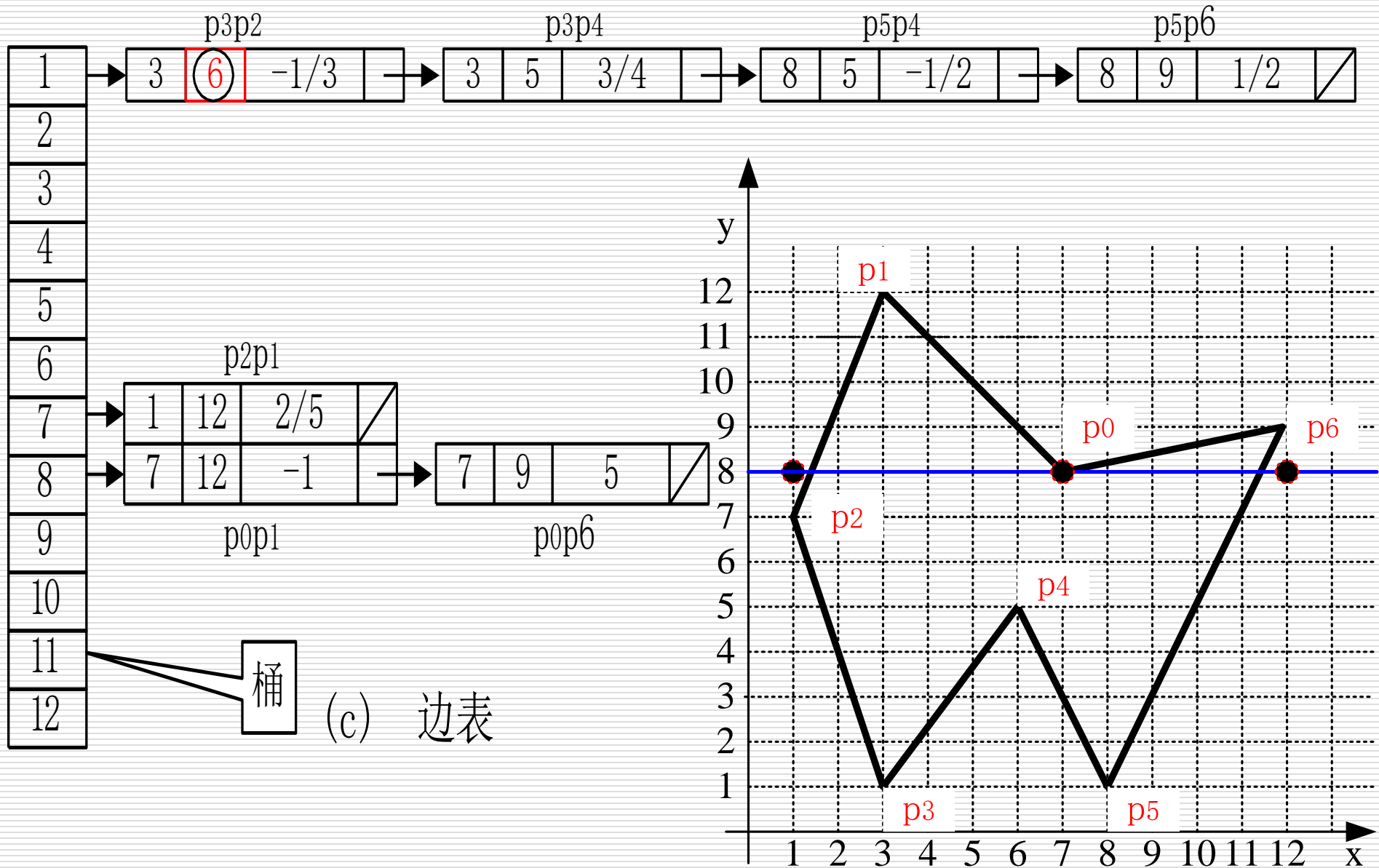


图5.32 多边形 $P_0P_1P_2P_3P_4P_5P_6$

改进的有效边表算法——算法步骤

- (1)初始化：构造边表，**AET**表置空；
- (2)将第一个不空的**ET**表中的边与**AET**表合并；
- (3)由**AET**表中取出交点对进行填充。填充之后删除 $y=y_{\max}$ 的边；
- (4) $y_{i+1}=y_i+1$,根据 $x_{i+1}=x_i+1/k$ 计算并修改**AET**表，同时合并**ET**表中 $y=y_{i+1}$ 桶中的边，按次序插入到**AET**表中，形成新的**AET**表；
- (5)**AET**表不为空则转(3)，否则结束。

边缘填充算法

□ 基本思想：按任意顺序处理多边形的每条边。

处理时，先求出该边与扫描线的交点，再对扫描线上交点右方的所有像素取反。

□ 算法简单，但对于复杂图型，每一像素可能被访问多次

栅栏填充算法

- ❑ 栅栏指的是一条过多边形顶点且与扫描线垂直的直线。它把多边形分为两半。
- ❑ 基本思想：按任意顺序处理多边形的每一条边，但处理每条边与扫描线的交点时，将交点与栅栏之间的像素取反。
- ❑ 这种算法尽管减少了被重复访问像素的数目，但仍有一些像素被重复访问。

边标志算法

- 基本思想：先用特殊的颜色在帧缓存中将多边形的边界勾画出来，然后将着色的像素点依 x 坐标递增的顺序配对，再把每一对象素构成的区间置为填充色。。
- 分为两个步骤：打标记；填充。
- 当用软件实现本算法时，速度与改进的有效边表算法相当，但本算法用硬件实现后速度会有很大提高。

4.2.2 区域填充

- 基本概念
- 区域的表示方法
- 区域的分类
- 区域填充算法

基本概念

- **区域填充**是指从区域内的某一个像素点（种子点）开始，由内向外将填充色扩展到整个区域内的过程。
- **区域**是指已经表示成点阵形式的填充图形，它是相互连通的一组像素的集合。

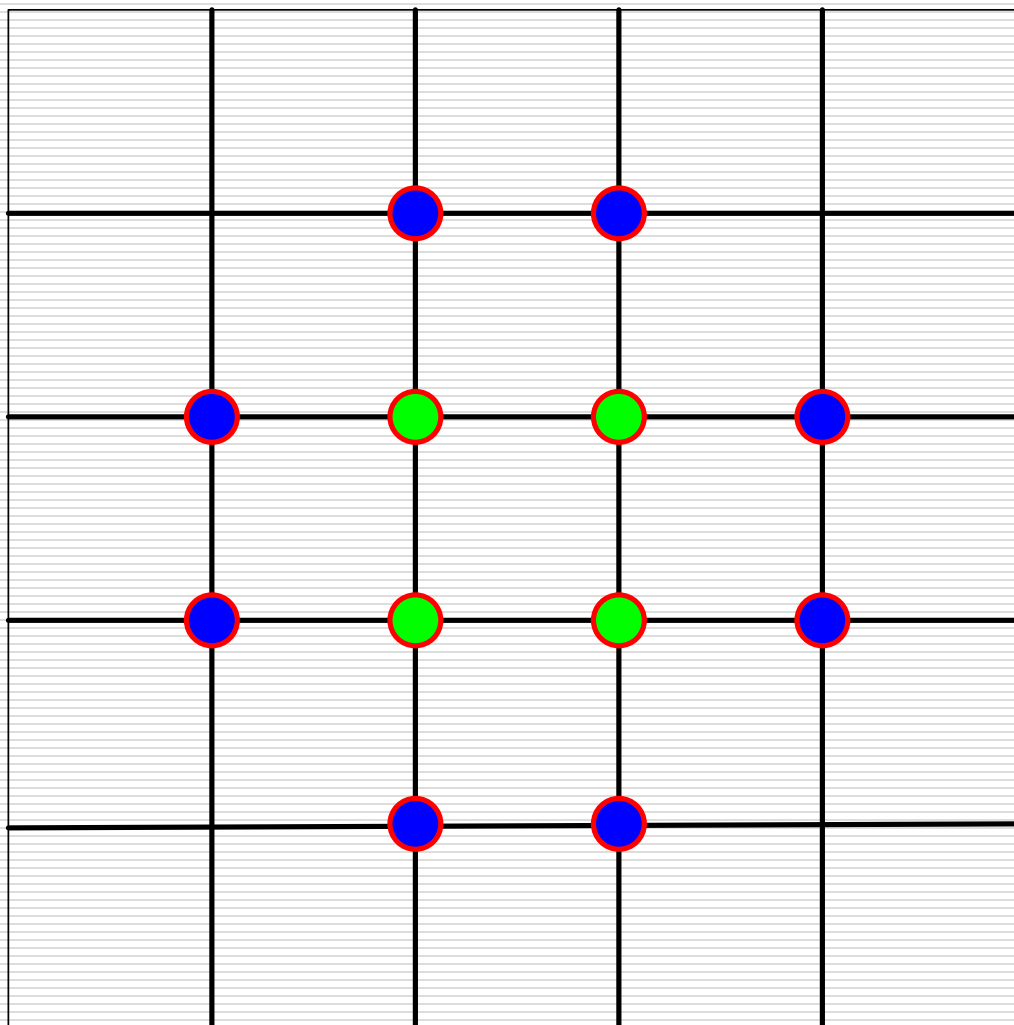


图5-33 区域的概念

区域的表示方法

- **边界表示法**：把位于给定区域的边界上的象素一一列举出来的方法。
- 边界表示法中，由于边界由特殊颜色指定，填充算法可以逐个象素地向外处理，直到遇到边界颜色为止，这种方法称为边界填充算法（**Boundary-fill Algorithm**）。

□ 内点表示：枚举出给定区域内所有像素的表示方法。以内点表示法为基础的区域填充算法称为泛填充算法（**Flood-fill Algorithm**）。

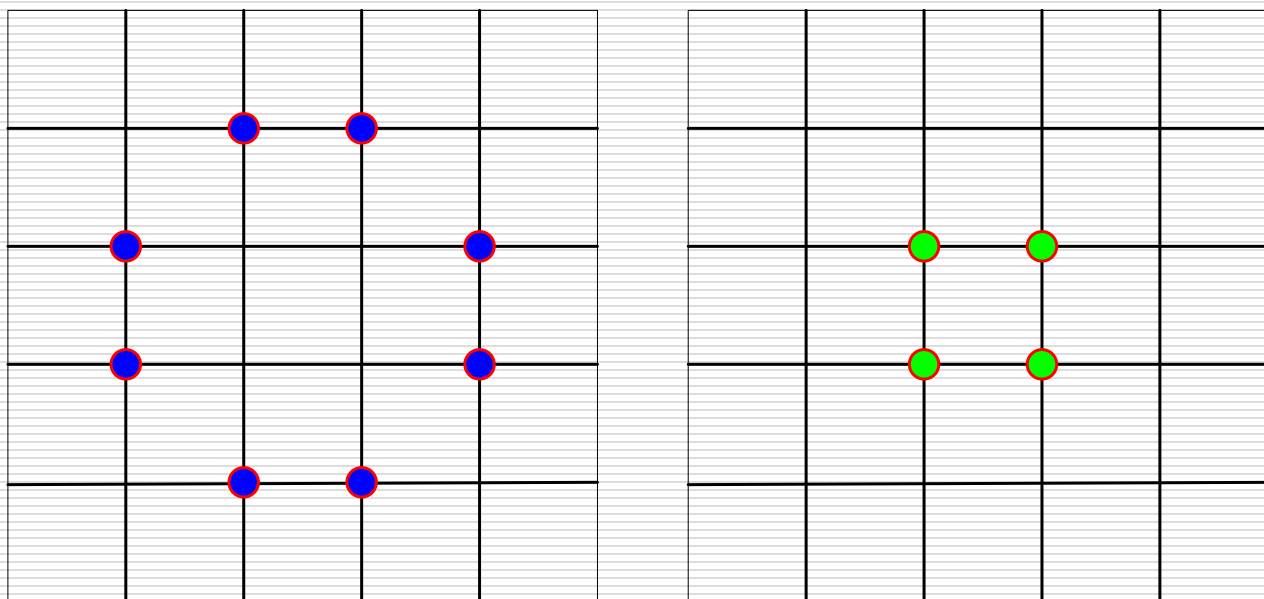
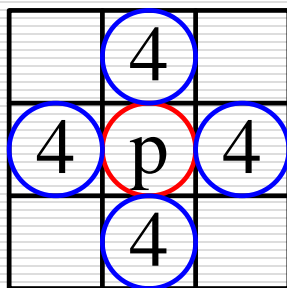


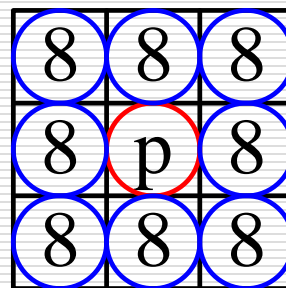
图5-34 区域的表示方法

区域的分类

□ 4-连通区域，8-连通区域



(a) 4-邻接点



(b) 8-邻接点

图5-35 4-邻接点与8-邻接点

区域的分类

- **4-连通区域**: 从区域上的一点出发, 通过访问已知点的**4-邻接点**, 在不越出区域的前提下, 遍历区域内的所有像素点。
- **8-连通区域**: 从区域上的一点出发, 通过访问已知点的**8-邻接点**, 在不越出区域的前提下, 遍历区域内的所有像素点。

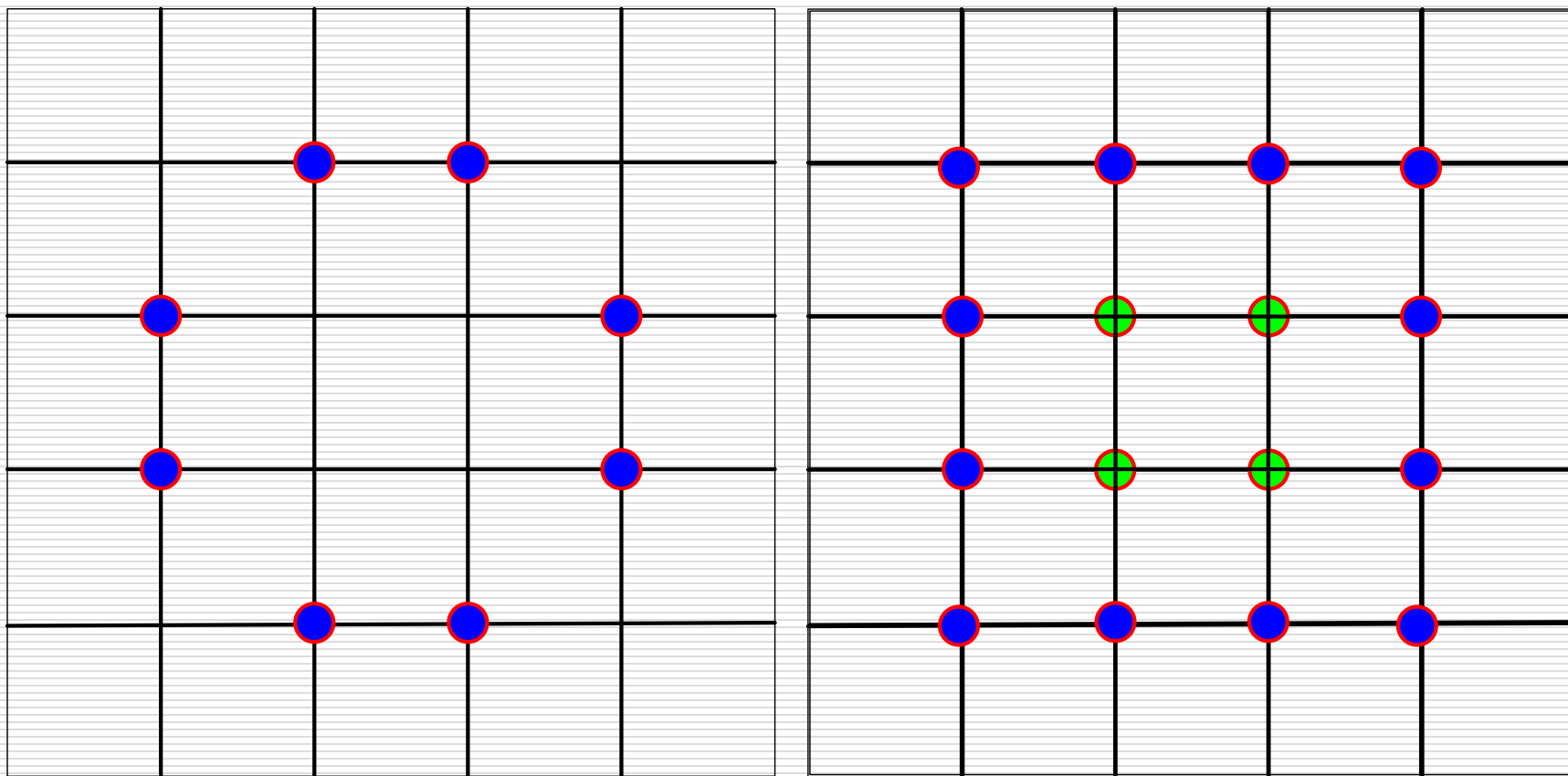


图5-36 4—连通与8—连通区域

4连通与8连通区域的区别

- 连通性： 4连通可看作8连通区域，但对边界有要求。
- 对边界的要求。

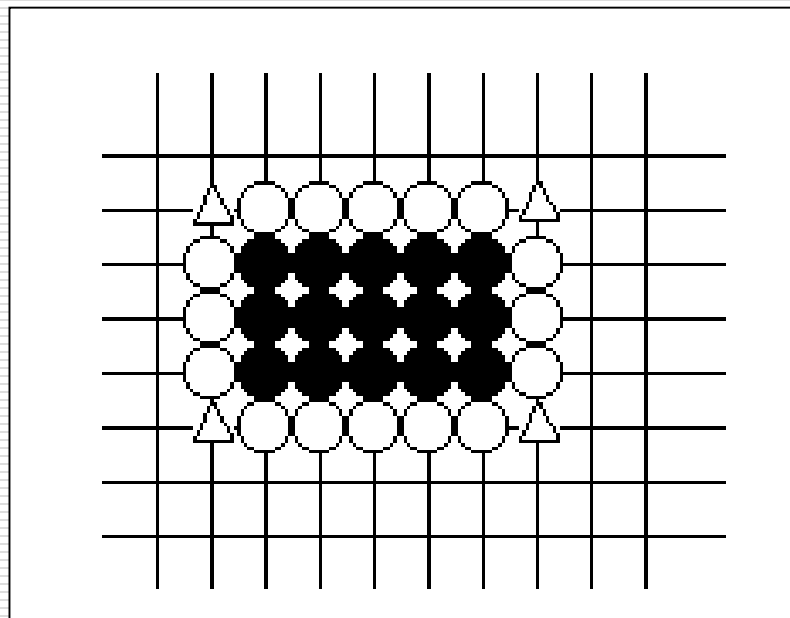


图5-37 4-连通与8-连通区域

区域填充算法

- 区域填充算法（边界填充算法和泛填充算法）
是根据区域内的一个已知像素点（种子点）出发，找到区域内其他像素点的过程，所以把这一类算法也成为**种子填充算法**。

区域填充算法——边界填充算法

- 算法的输入：种子点坐标 (x,y) ，填充色以及边界颜色。
- 利用堆栈实现简单的种子填充算法

算法从种子点开始检测相邻位置是否是边界颜色，若不是就用填充色着色，并检测该像素点的相邻位置，直到检测完区域边界颜色范围内的所有像素为止。

区域填充算法——边界填充算法

栈结构实现4-连通边界填充算法的算法步骤为：

种子像素入栈；当栈非空时重复执行如下三步操作：

(a) 栈顶像素出栈；

(b) 将出栈像素置成填充色；

(c) 检查出栈像素的4-邻接点，若其中某个像素点不是边界色且未置成多边形色，则把该像素入栈。

区域填充算法——边界填充算法

栈结构实现8-连通边界填充算法的算法步骤为：

种子像素入栈；当栈非空时重复执行如下三步操作：

(a) 栈顶像素出栈；

(b) 将出栈像素置成填充色；

(c) 检查出栈像素的8-邻接点，若其中某个像素点不是边界色且未置成多边形色，则把该像素入栈。

区域填充算法——边界填充算法

- ❑ 可以用于填充带有内孔的平面区域。
- ❑ 把太多的像素压入堆栈，降低了效率，同时需要较大的存储空间。
- ❑ 递归执行，算法简单，但效率不高，区域内每一像素都引起一次递归，进/出栈费时费内存。
- ❑ 通过沿扫描线填充水平像素段，来代替处理4-邻接点和8-邻接点。

区域填充算法——边界填充算法

- 扫描线种子填充算法：
扫描线通过在任意不间断扫描线区间中只取一个种子像素的方法使堆栈的尺寸极小化。不间断区间是指在一条扫描线上的一组相邻像素。

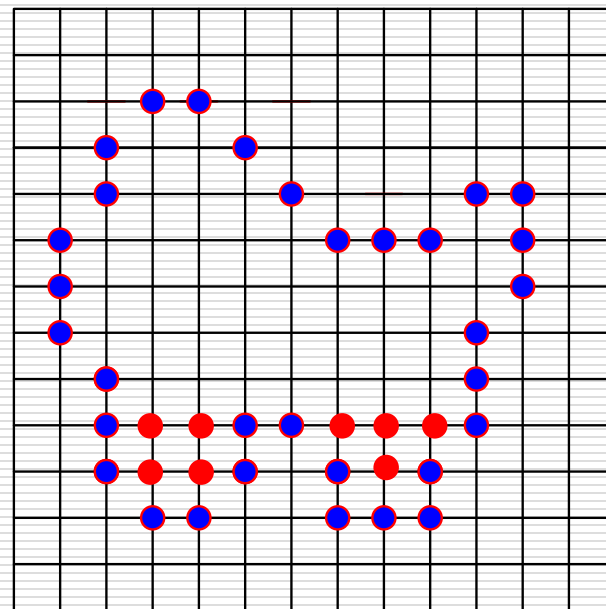


图5-38 扫描线种子填充算法

区域填充算法——边界填充算法

- 基本过程：当给定种子点时，首先填充种子点所在的扫描线上的位于给定区域的一个区段，然后确定与这一区段相通的上下两条扫描线上位于给定区域内的区段，并依次保存下来。反复这个过程，直到填充结束。

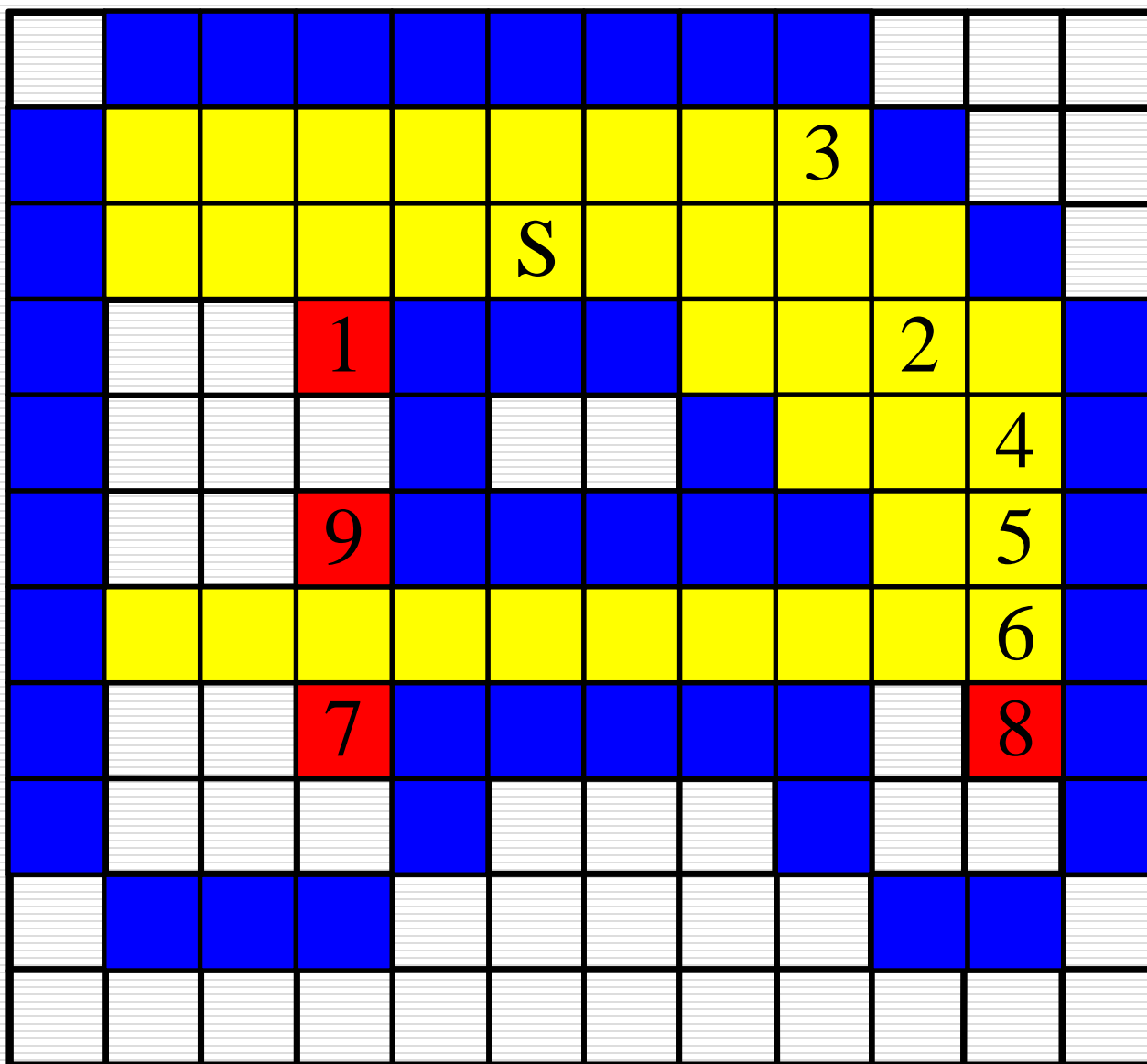


图5-39 扫描线种子填充算法过程

区域填充算法——泛填充算法

- 算法的输入：种子点坐标 (x,y) ，填充色和内部点的颜色。
- 算法原理：算法从指定的种子 (x,y) 开始，用所希望的填充颜色赋给所有当前为给定内部颜色的像素点。

区域填充算法——泛填充算法

- 种子像素入栈；栈非空时重复执行如下三步操作：
 - (1) 栈顶像素出栈；
 - (2) 将出栈像素置成填充色；
 - (3) 检查出栈像素的8-邻接点，若其中某个像素点不是给定内部点的颜色且未置成新的填充色，则把该像素入栈。

区域填充算法——泛填充算法

- 当以边界表示时，**4-连通边界填充算法**只能填充**4-连通区域**，**8-连通边界填充算法**也只能填充**8-连通区域**。
- 当以内点表示时，**8-连通泛填充算法**可以填充**8-连通区域**也可以填充**4-连通区域**，当然**4-连通泛填充算法**还是只能填充**4-连通区域**。

多边形扫描转换与区域填充方法比较

相同点：

- 都是光栅图形面着色，用于真实感图形显示。
- 可相互转换。

多边形扫描转换与区域填充方法比较

- **基本思想不同**：前者将顶点表示转换成点阵表示，而后者只改变区域内填充颜色，没有改变表示方法。
- **对边界的要求不同**：前者只要求扫描线与多边形边界交点个数为偶数。后者则要求区域封闭，防止递归填充跨界。
- **基本的条件不同**：前者是从边界顶点信息出发，而后者从区域内种子点开始算法。

4.2.3 内外测试

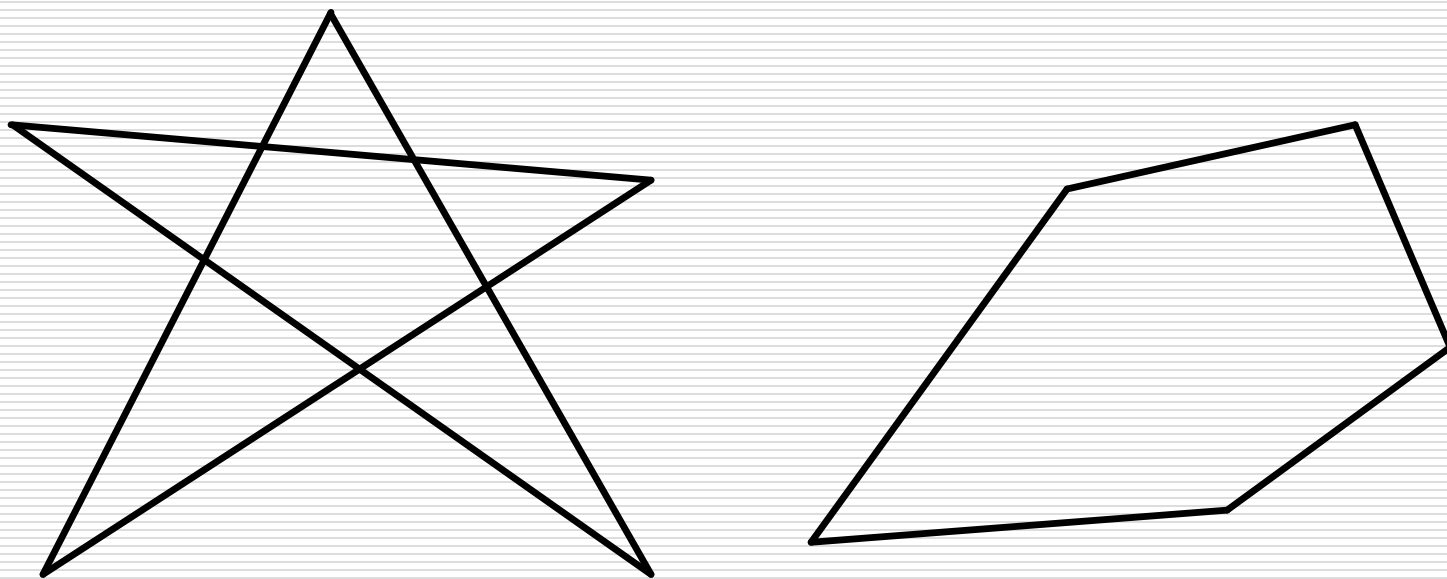
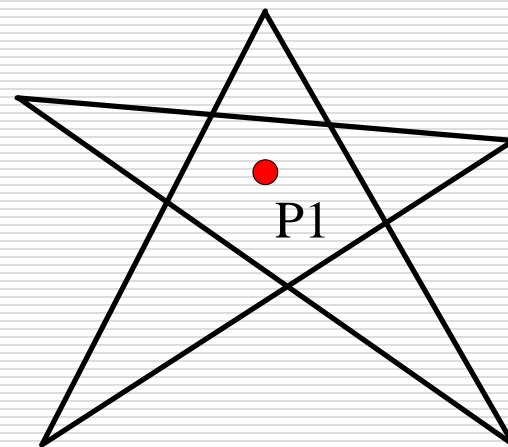


图5-40 不自交的多边形与自相交的多边形

内外测试

□ 奇-偶规则 (Odd-even Rule)

从任意位置 p 作一条射线，若与该射线相交的多边形边的数目为奇数，则 p 是多边形内部点，否则是外部点。



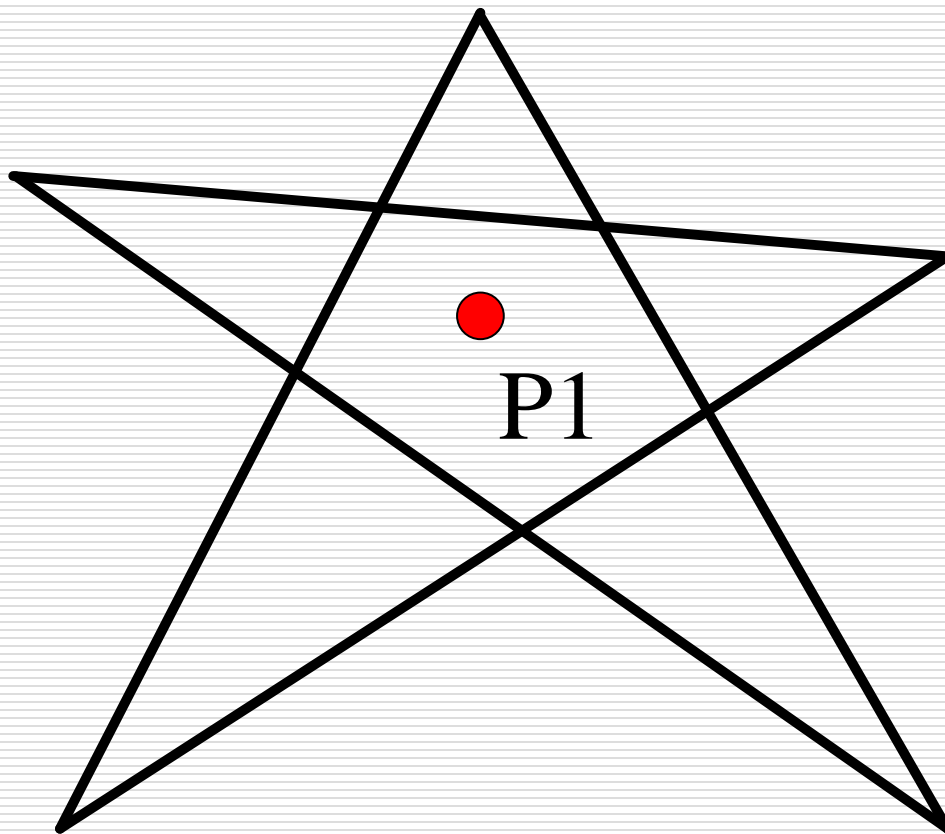
内外测试

非零环绕数规则 (Nonzero Winding Number Rule)

- 首先使多边形的边变为矢量。
- 将环绕数初始化为零。
- 再从任意位置 p 作一条射线。当从 p 点沿射线方向移动时，对在每个方向上穿过射线的边计数，每当多边形的边从右到左穿过射线时，环绕数加 1 ，从左到右时，环绕数减 1 。
- 处理完多边形的所有相关边之后，若环绕数为非零，则 p 为内部点，否则， p 是外部点。

内外测试

□ 两种规则的比较



4.2.4 区域填充属性

- 区域填充属性选择包括颜色、图案和透明度
 - 用模板定义填充图案
 - 修改填充算法
 - 透明方式，则对应位置为**1**时用前景色写像素，否则不改变该像素的值。
 - 不透明方式，则视对应位置为**1**或**0**来决定是用前景色还是背景色去写像素。

区域填充属性

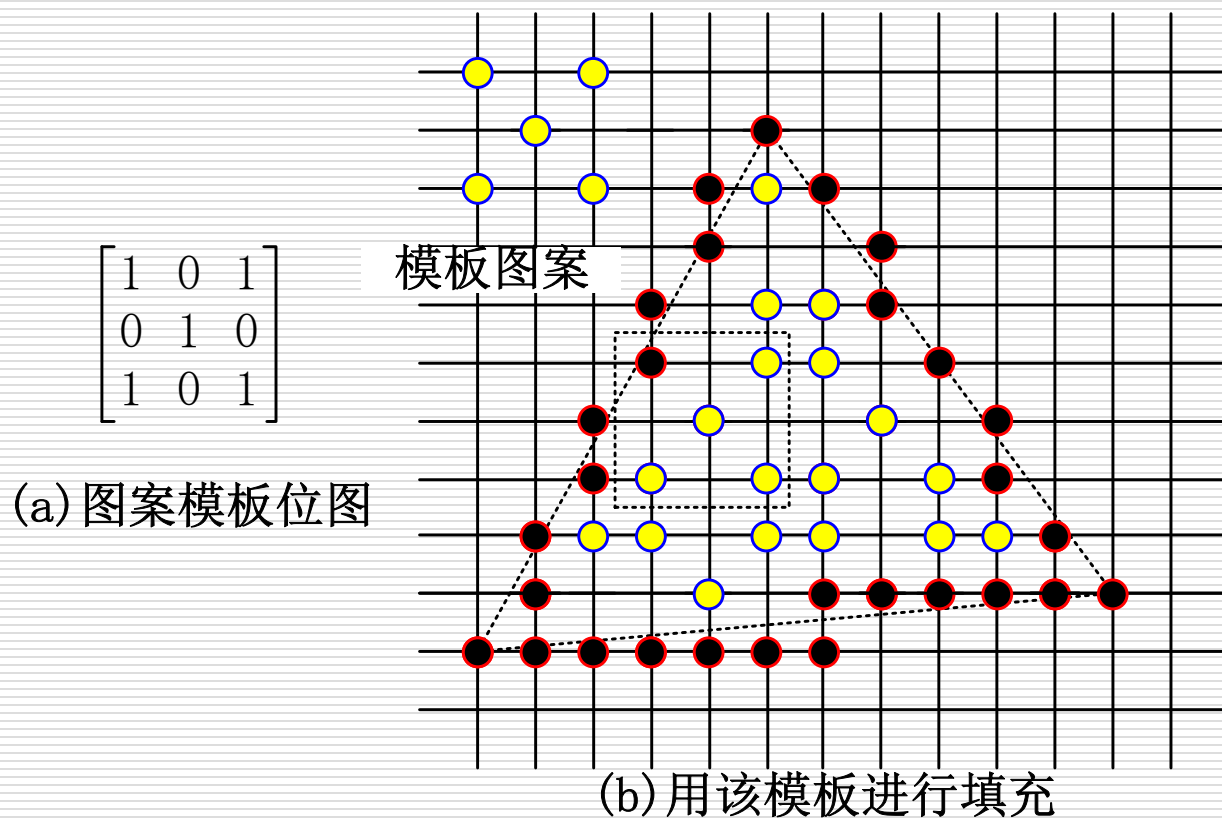


图4.41 利用图案模板进行三角形的填充

4.4 反走样

□ 用离散量表示连续量引起的失真，就叫做走样（**Aliasing**）。

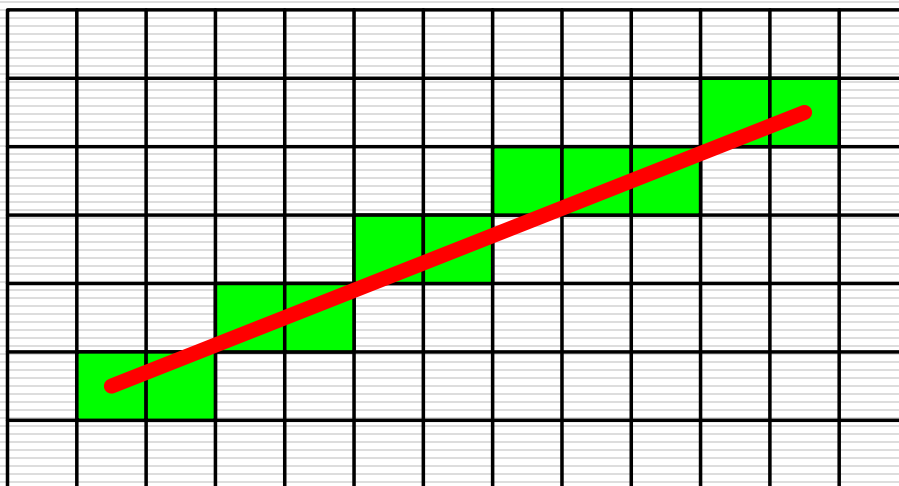


图5-42 绘制直线时的走样现象

反走样

产生原因：

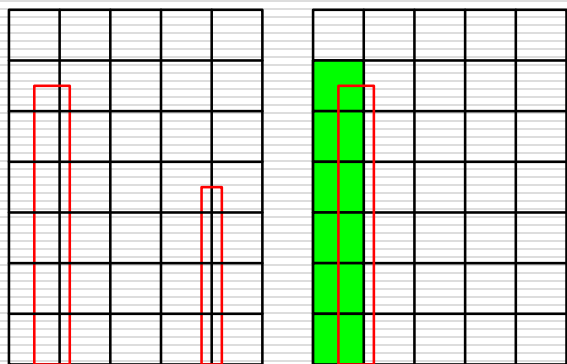
数学意义上的图形是由无限多个连续的、面积为零的点构成；但在光栅显示器上，用有限多个离散的，具有一定面积的像素来近似地表示他们。

反走样

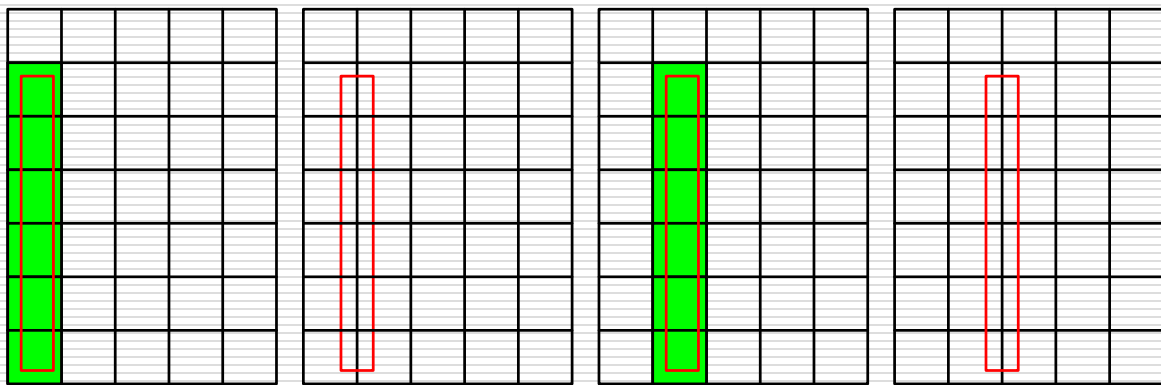
走样现象：

- 一是光栅图形产生的阶梯形。
- 一是图形中包含相对微小的物体时，这些物体在静态图形中容易被丢弃或忽略，在动画序列中时隐时现，产生闪烁。

例子



(a)需显示的矩形 (b)显示结果



(a)显示 (b)不显示 (c)显示 (d)不显示

图5-43 丢失细节与运动图形的闪烁

□ 用于减少或消除这种效果的技术，称为反走样（**antialiasing**，图形保真）。

□ 一种简单方法：

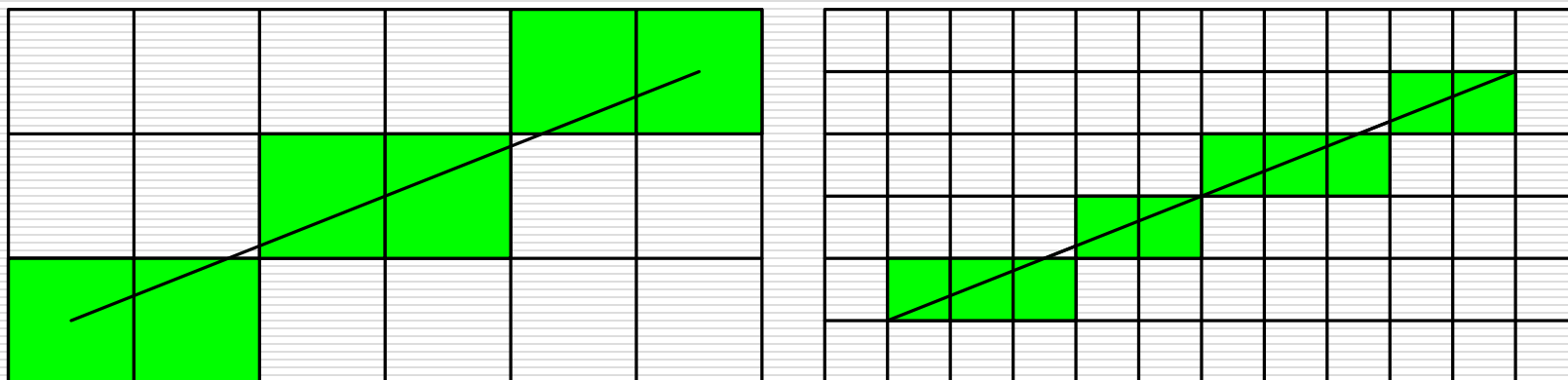


图5-44 分辨率提高一倍，阶梯程度减小一倍

□ 过取样（**super sampling**），或后滤波

□ 区域取样（**area sampling**），或前滤波

反走样——过取样 (super sampling)

□ 过取样：在高于显示分辨率的较高分辨率下用点取样方法计算，然后对几个像素的属性进行平均得到较低分辨率下的像素属性。

□ 简单过取样

在 x , y 方向把分辨率都提高一倍, 使每个像素对应4个子像素, 然后扫描转换求得各子像素的颜色亮度, 再对4个像素的颜色亮度进行平均, 得到较低分辨率下的像素颜色亮度。

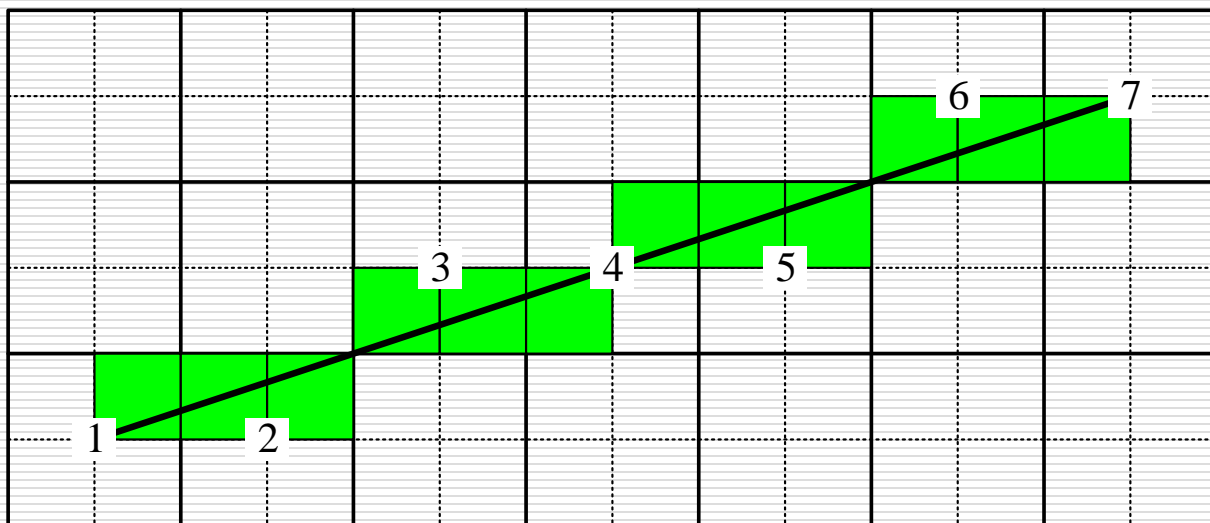


图5-45 简单的过取样方式

□ **重叠过取样**：为了得到更好的效果，在对一个像素点进行着色处理时，不仅仅只对其本身的子像素进行采样，同时对其周围的多个像素的子像素进行采样，来计算该点的颜色属性。

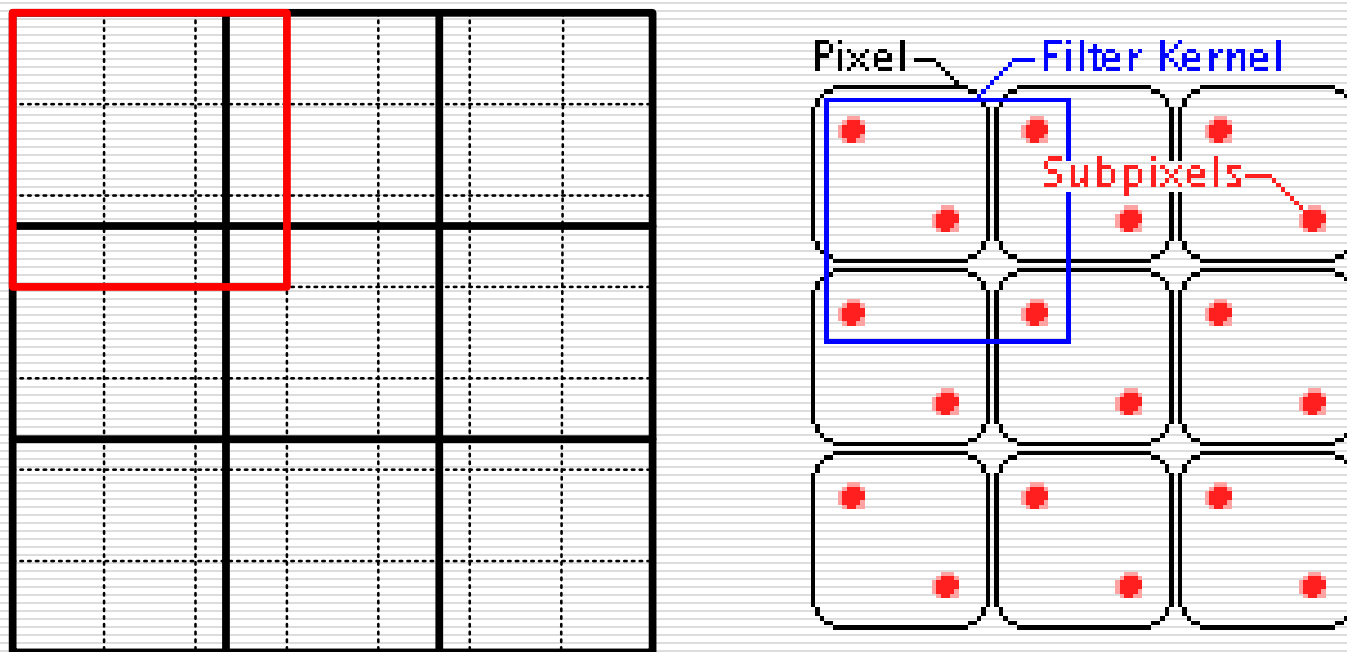


图5-46 重叠过取样

- 基于加权模板的过取样：前面在确定像素的亮度时，仅仅是对所有子像素的亮度进行简单的平均。更常见的做法是给接近像素中心的子像素赋予较大的权值，即对所有子像素的亮度进行加权平均。

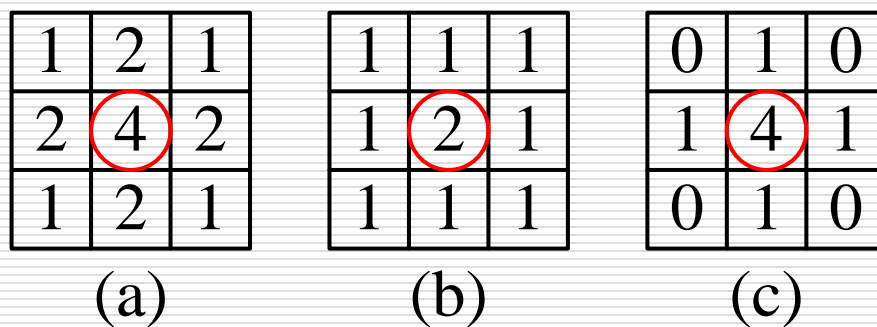


图5-47 常用的加权模板

优点:

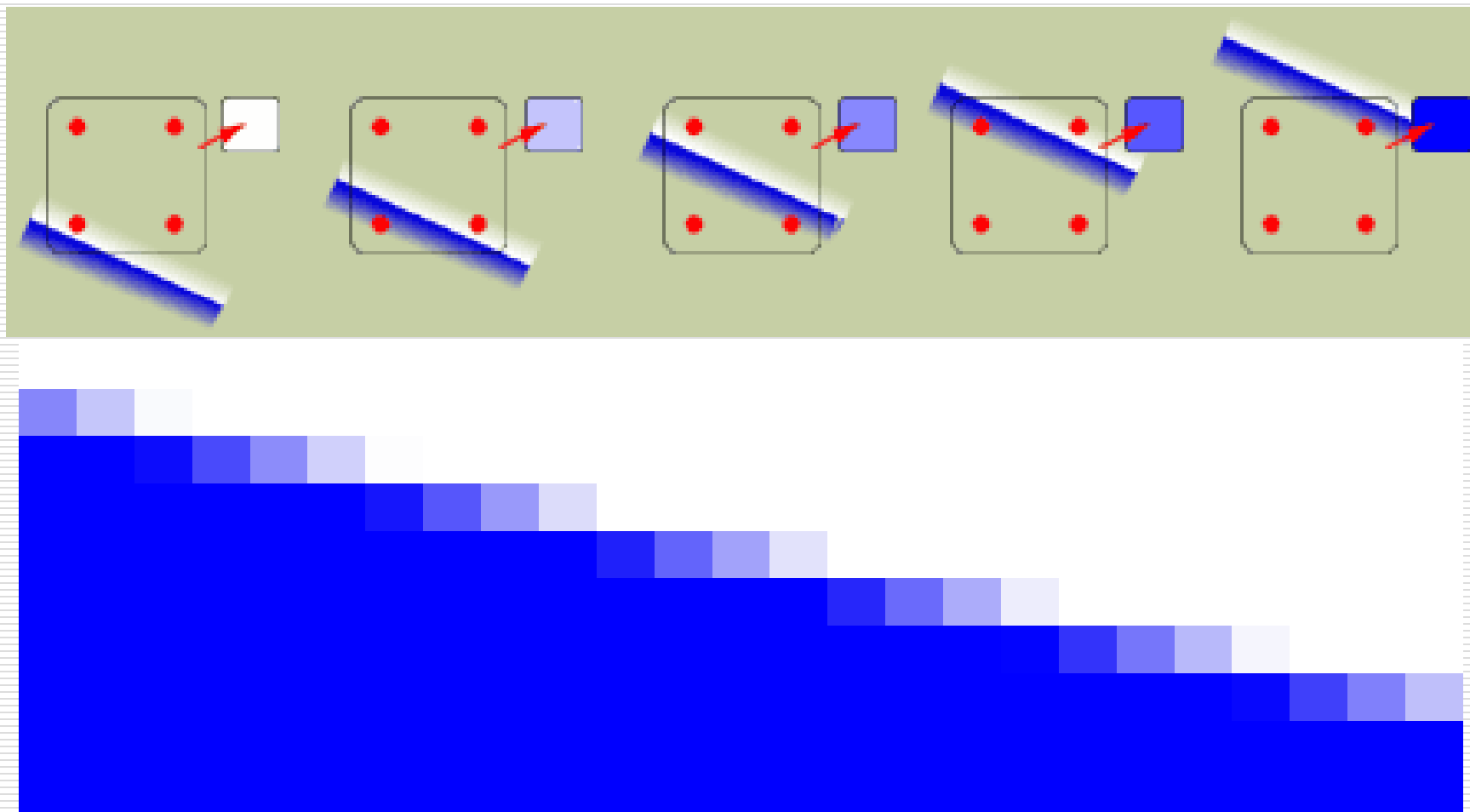


图5-48 过取样的效果

缺点：特殊角度的处理问题

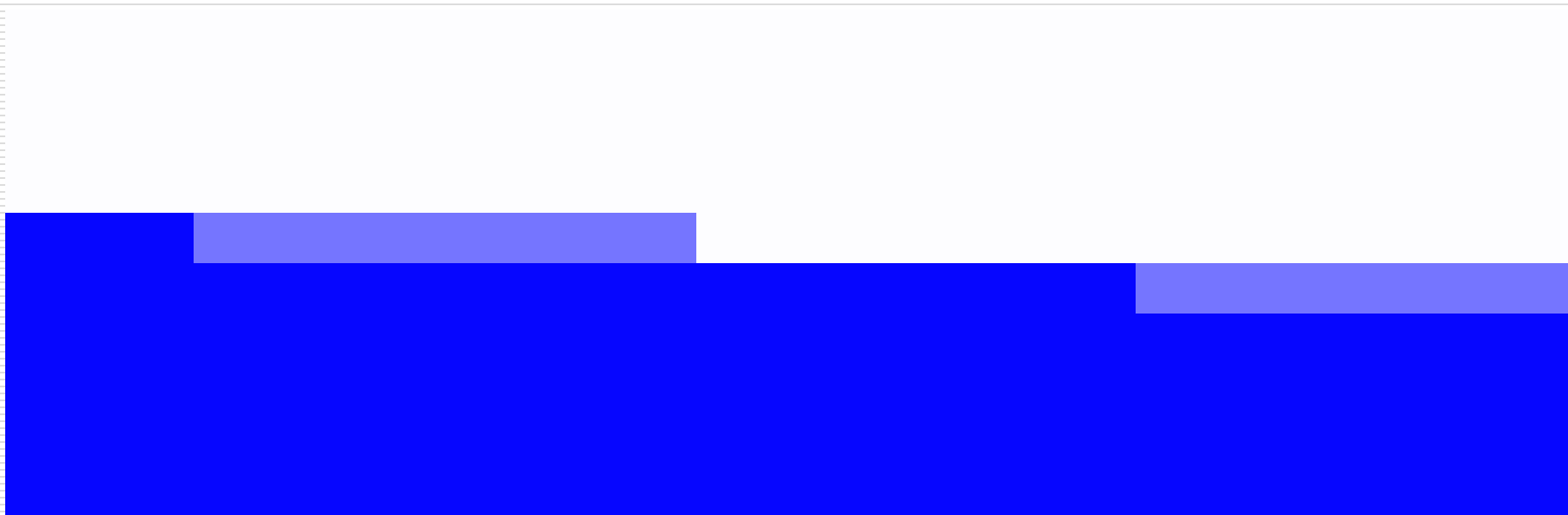


图5-49 特殊角度的问题

反走样——简单的区域取样

- 在整个像素区域内进行采样，这种技术称为区域取样。又由于像素的亮度是作为一个整体被确定的，不需要划分子像素，故也被称为前置滤波。

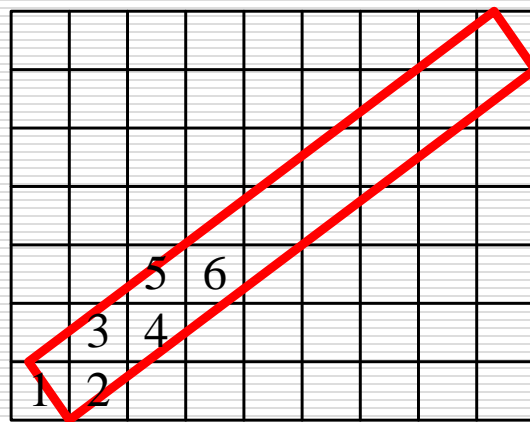


图5-50 有宽度的直线段

反走样——简单的区域取样

如何计算直线段与像素相交区域的面积？

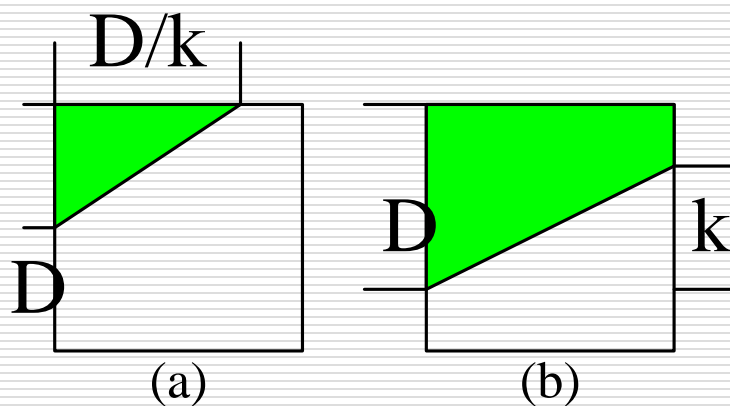


图5-51 重叠区域面积的计算

反走样——简单的区域取样

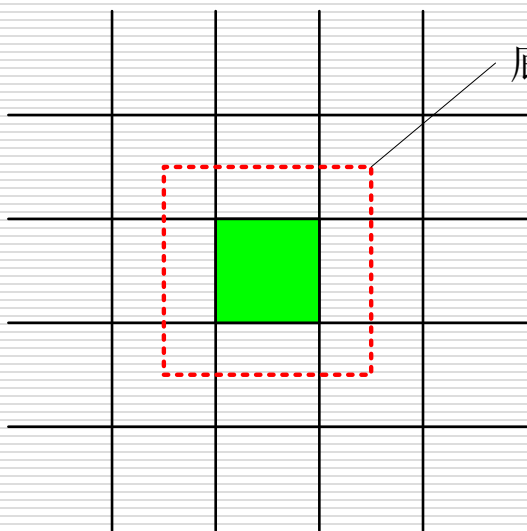
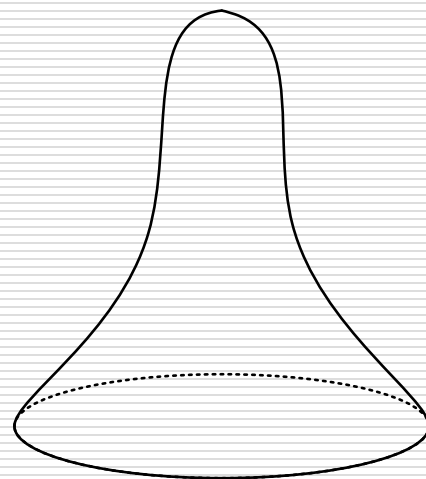
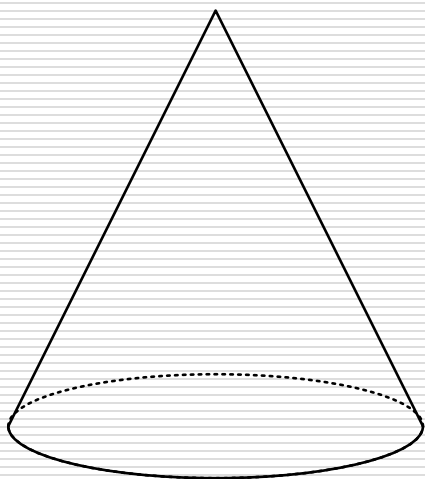
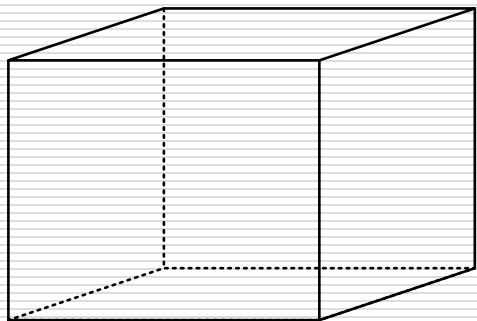
- 可以利用一种求相交区域的近似面积的离散计算方法：
 - (1)将屏幕像素分割成 n 个更小的子像素，
 - (2)计算中心落在直线段内的子像素的个数 m ，
 - (3) m/n 为线段与像素相交区域面积的近似值。
- 直线段对一个像素亮度的贡献与两者重叠区域的面积成正比。
- 相同面积的重叠区域对像素的贡献相同。

反走样——加权区域取样

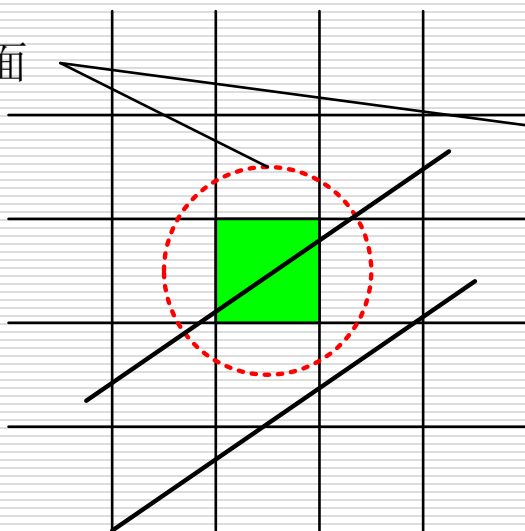
- 过取样中，我们对所有子像素的亮度进行简单平均或加权平均来确定像素的亮度。
- 在区域取样中，我们使用覆盖像素的连续的加权函数（**Weighting Function**）或滤波函数（**Filtering Function**）来确定像素的亮度。

加权区域取样原理

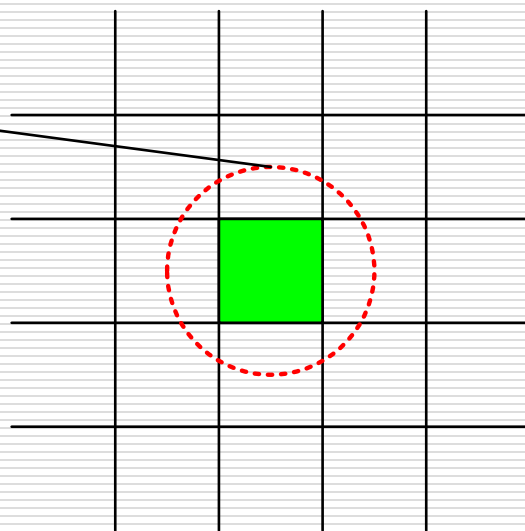
加权函数 $W(x,y)$ 是定义在二维显示平面上的函数。对于位置为 (x,y) 的小区域 dA 来说，函数值 $W(x,y)$ （也称为在 (x,y) 处的高度）表示小区域 dA 的权值。将加权函数在整个二维显示图形上积分，得到具有一定体积的滤波器（**Filter**），该滤波器的体积为1。将加权函数在显示图形上进行积分，得到滤波器的一个子体，该子体的体积介于0到1之间。用它来表示像素的亮度。



(a)立方体滤波



(b)圆锥滤波



(c)高斯滤波

图5-52 常用的滤波函数

反走样——加权区域取样

特点:

- 接近理想直线的像素将被分配更多的灰度值;
- 相邻两个像素的滤波器相交, 有利于缩小直线条上相邻像素的灰度差。