

華東理工大學

计算机图形学

2023年9月

奉贤校区



華東理工大學



05

基本图形生成算法

Basic Graphics Generation

绘制流水线

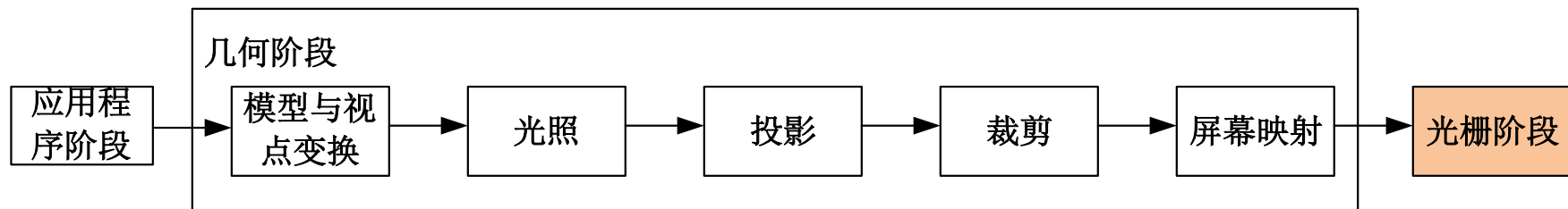
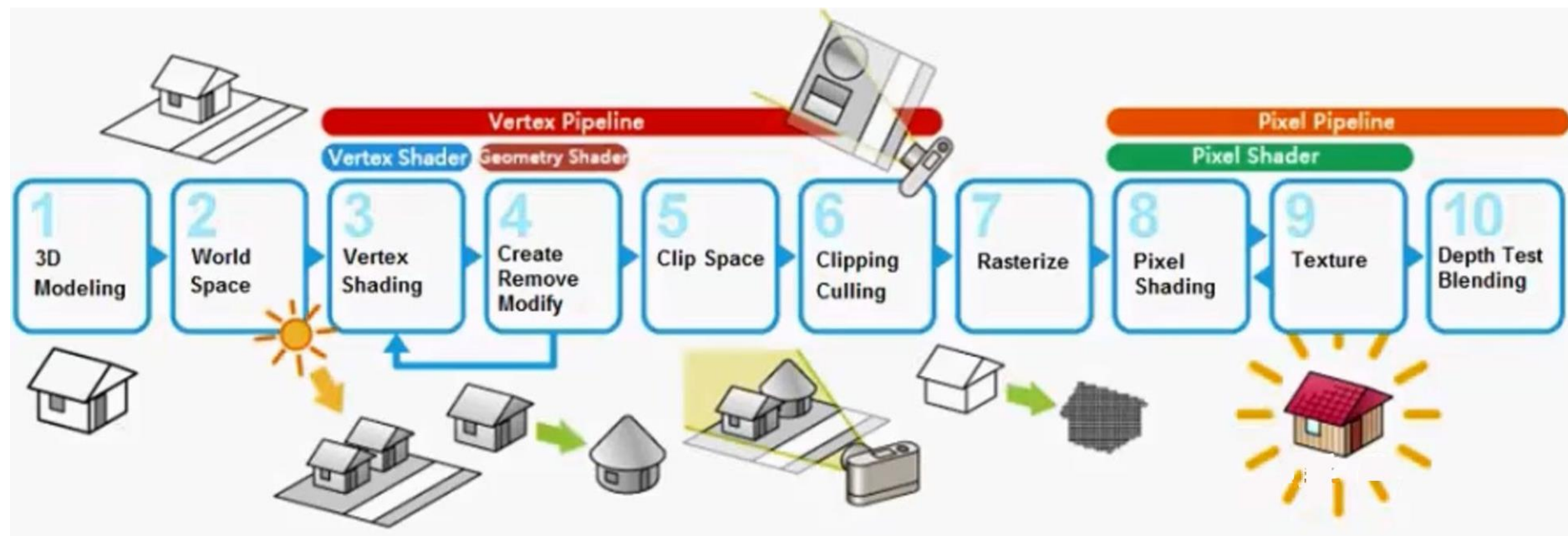


图2.22 绘制流水线的结构



- 快速
- 真实逼近

主要内容

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

□ 直线、圆

□ 多边形域

□ 字符串

□ 相关属性

图形生成的概念

- 图形的（显示）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。

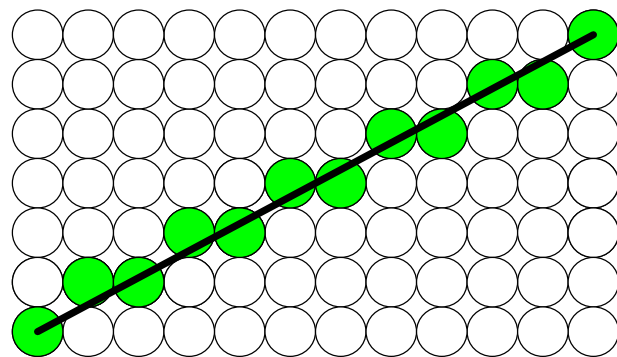


图5.1 用像素点集逼近直线

直线段的扫描转换

1

直线段的绘制要求

2

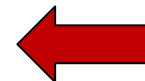
DDA算法

3

中点Bresenhan算法

4

改进的Bresenhan算法



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。
- 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。

直线段的扫描转换

1

直线段的绘制要求

2

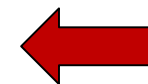
DDA算法

3

中点Bresenhan算法

4

改进的Bresenhan算法



数值微分法(DDA法)

直线的微分方程:

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

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

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

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

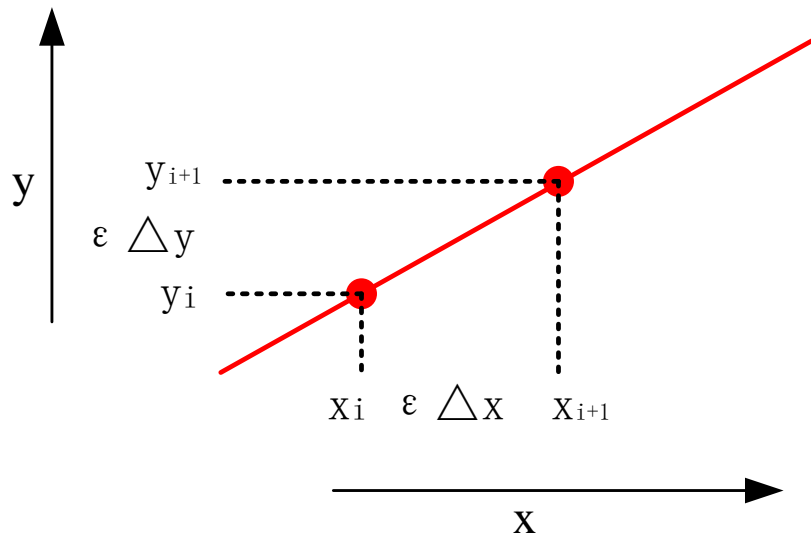


图5.2 DDA算法原理

• $\max(|\Delta x|, |\Delta y|) = |\Delta x|$, 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

直线段的扫描转换

1

直线段的绘制要求

2

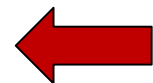
DDA算法

3

中点Bresenhan算法

4

改进的Bresenhan算法



中点Bresenham算法

- Jack Elton Bresenham 1962年 1965年 《IBM系统期刊》
- 直线的方程

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

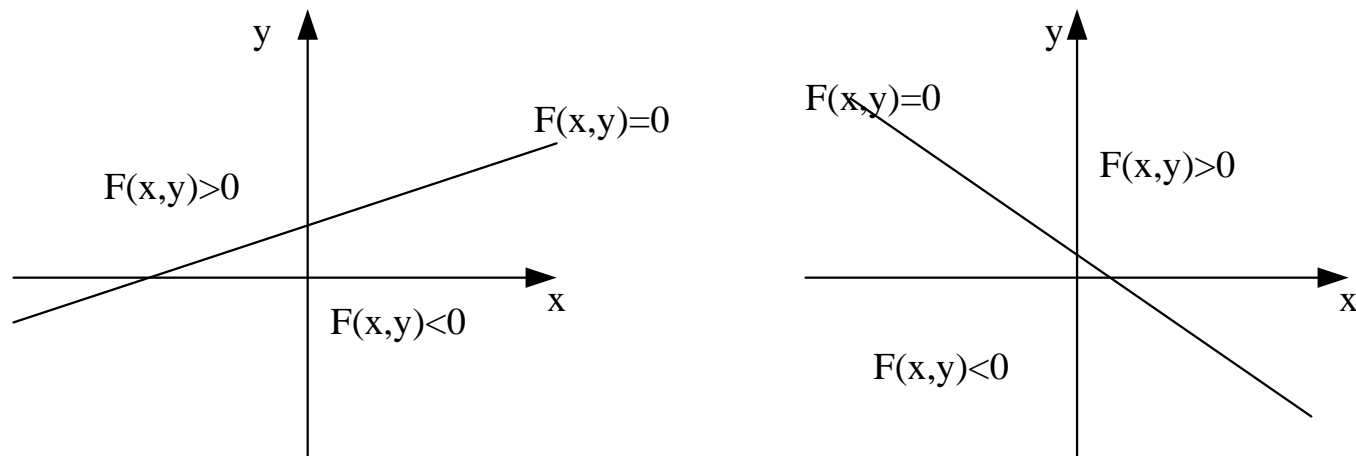


图5.3 直线将平面分为三个区域

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

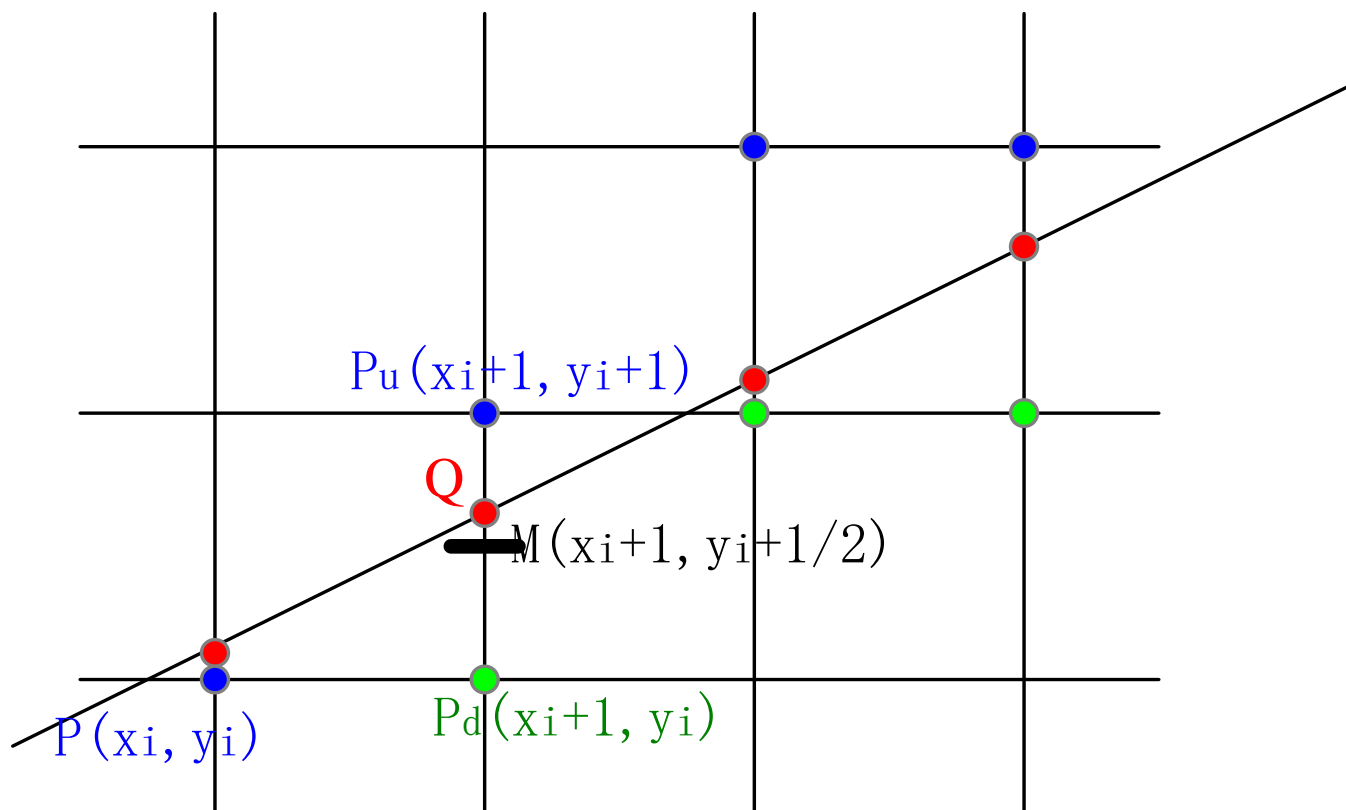


图5.4 Bresenham算法生成直线的原理

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：

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

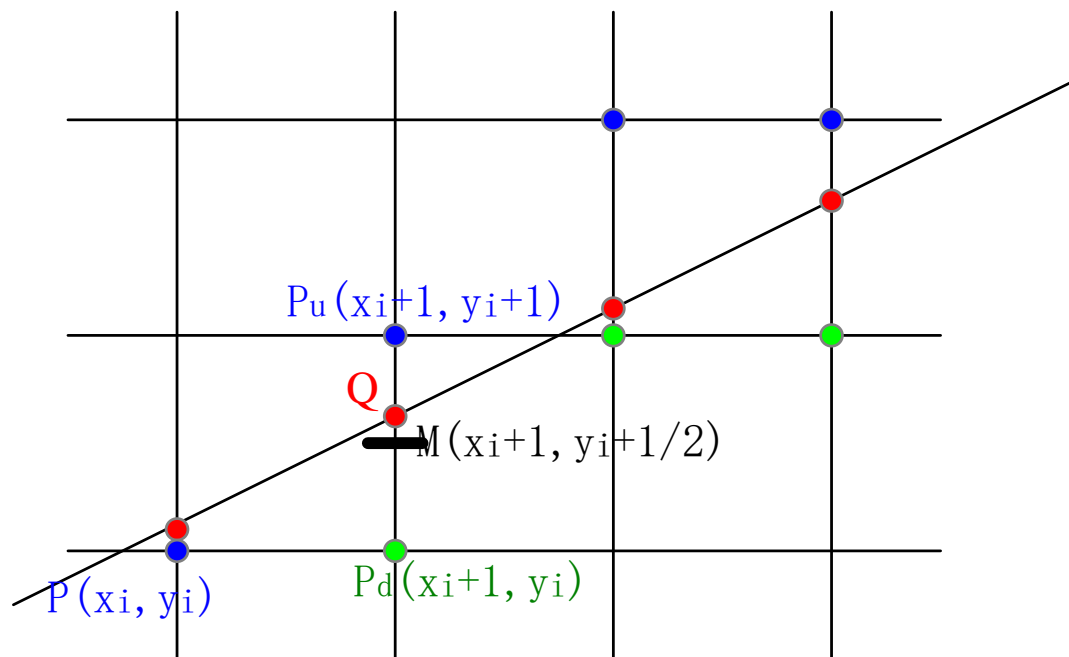


图5.5 Bresenham算法生成直线的原理

d的意义:

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

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

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

$$= y_M - y_Q$$

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

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

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

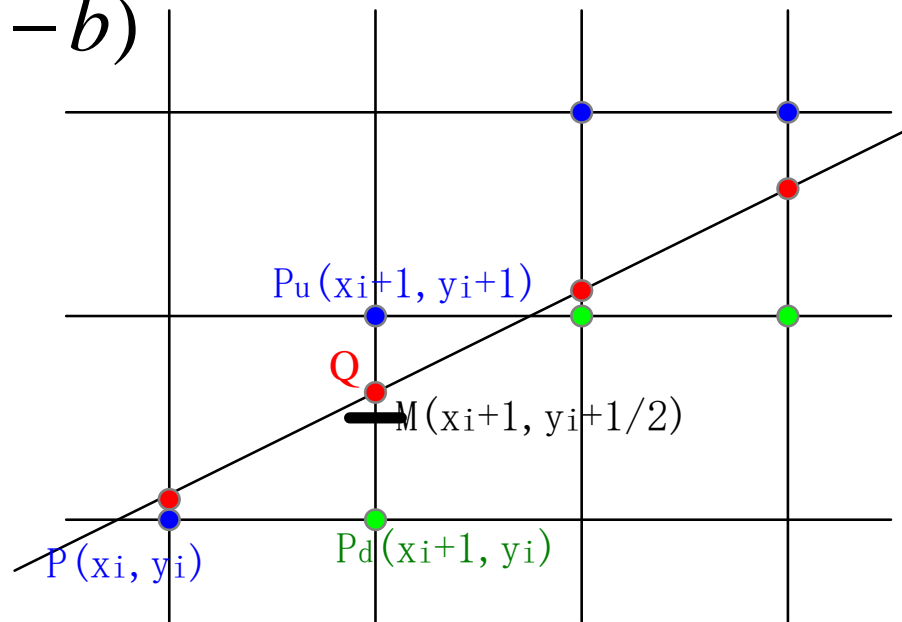


图5.6 Bresenham算法生成直线的原理

误差项的递推 ($d < 0$)

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

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

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

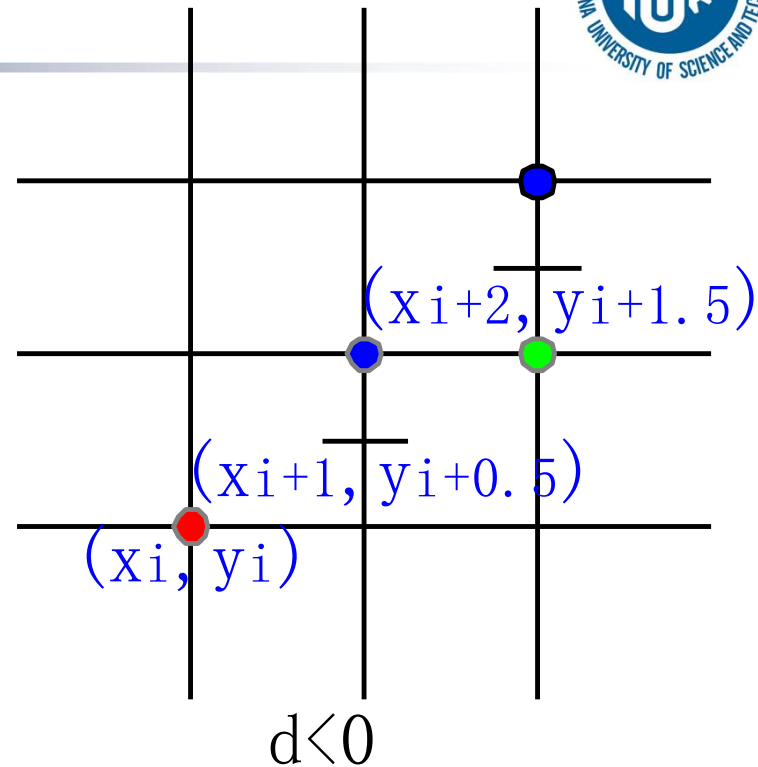


图5.7 误差项递推

误差项的递推 ($d \geq 0$)

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

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

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

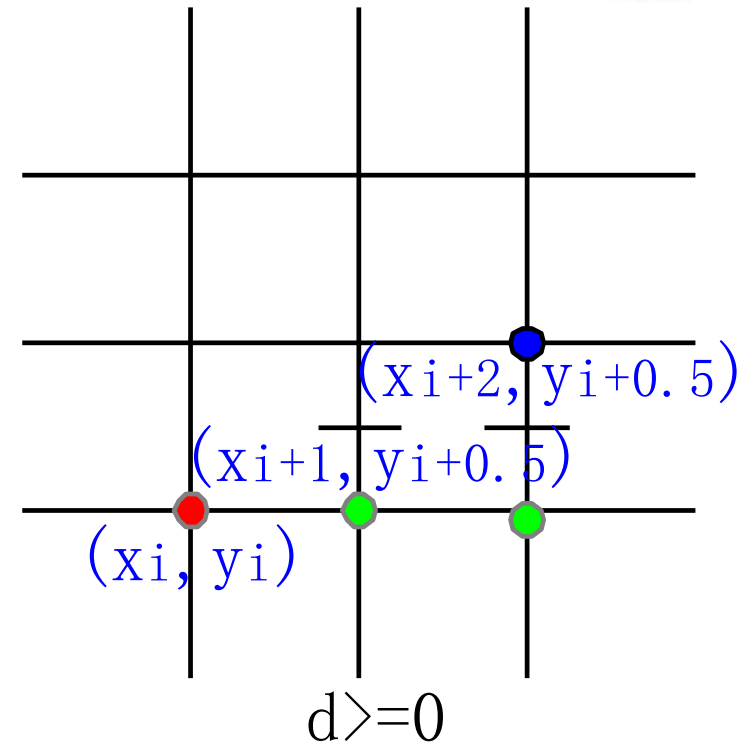


图5.8 误差项递推

中点Bresenham算法

初始值d的计算

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

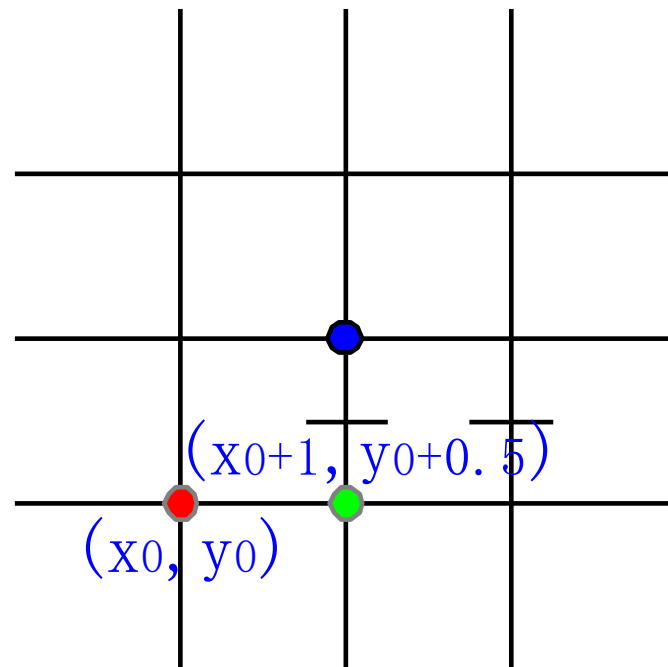


图5.9 计算初值

中点Bresenham算法——改进

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。

程序演示

直线段的扫描转换

1

直线段的绘制要求

2

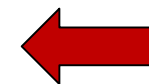
DDA算法

3

中点Bresenhan算法

4

改进的Bresenhan算法



改进的Bresenham算法

假定直线段的 $0 \leq k \leq 1$

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

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

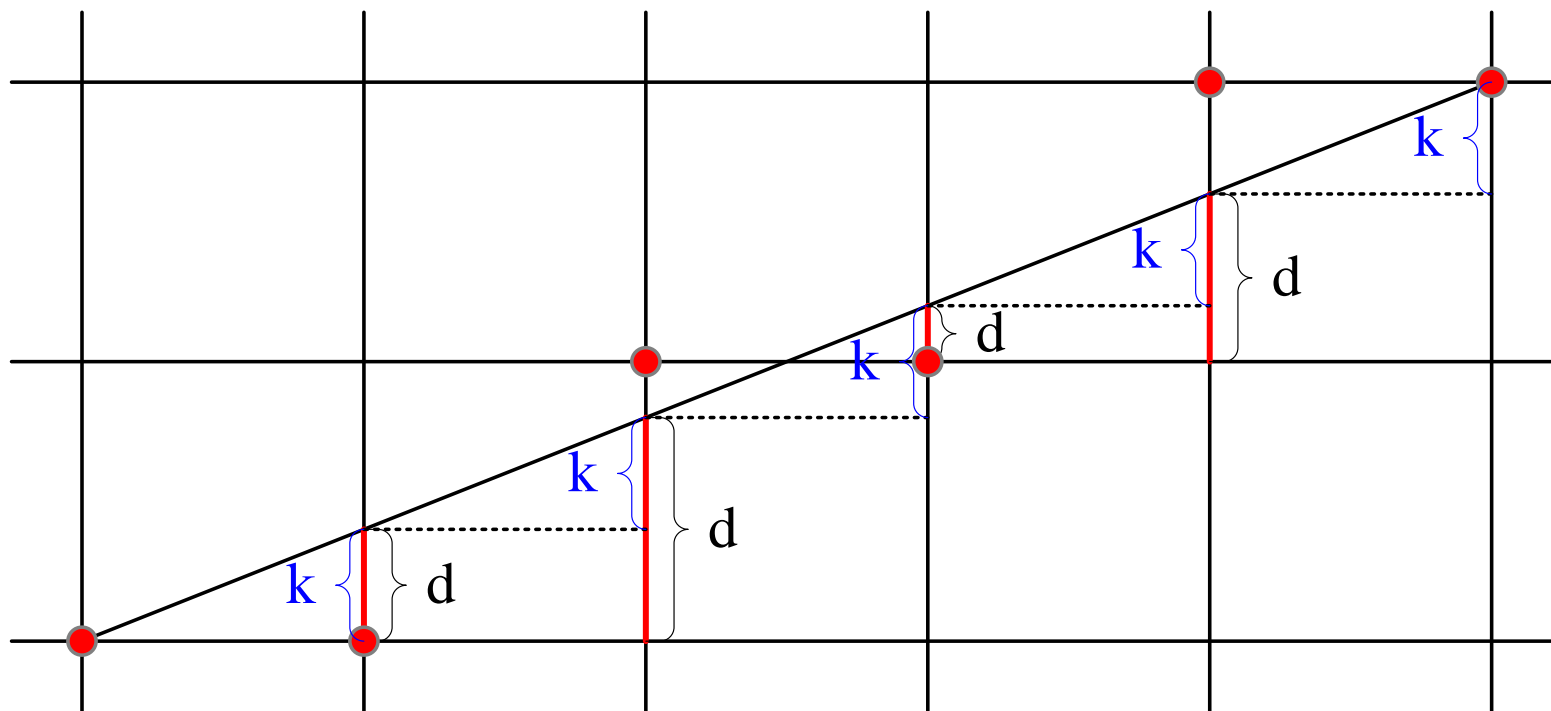
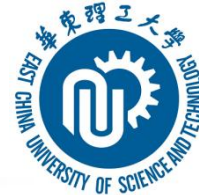


图5.10 改进的Brensemham算法绘制直线的原理

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: $d=d+k$

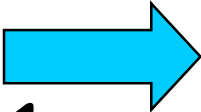
一旦y方向上走了一步, $d=d-1$

改进1：消除小数

令 $e = d - 0.5$

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

误差项的计算

- | | | |
|--|---|--|
| <ul style="list-style-type: none"> • $d_{\text{初}} = 0,$ • 每走一步: $d = d + k$ • if $(e > 0)$ then $d = d - 1$ |  | <ul style="list-style-type: none"> • $e_{\text{初}} = -0.5,$ • 每走一步有 $e = e + k.$ • if $(e > 0)$ then $e = e - 1$ |
|--|---|--|

改进2：避免除法

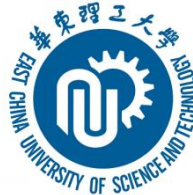
用 $E=2e\Delta x$ 来替换 e

- $e_{\text{初}} = -0.5$
- 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。

圆的扫描转换

1

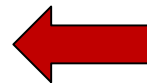
八分法画圆

2

简单方程生成圆弧

3

中点Bresenhan算法画圆



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 $x^2+y^2=R^2$ 。

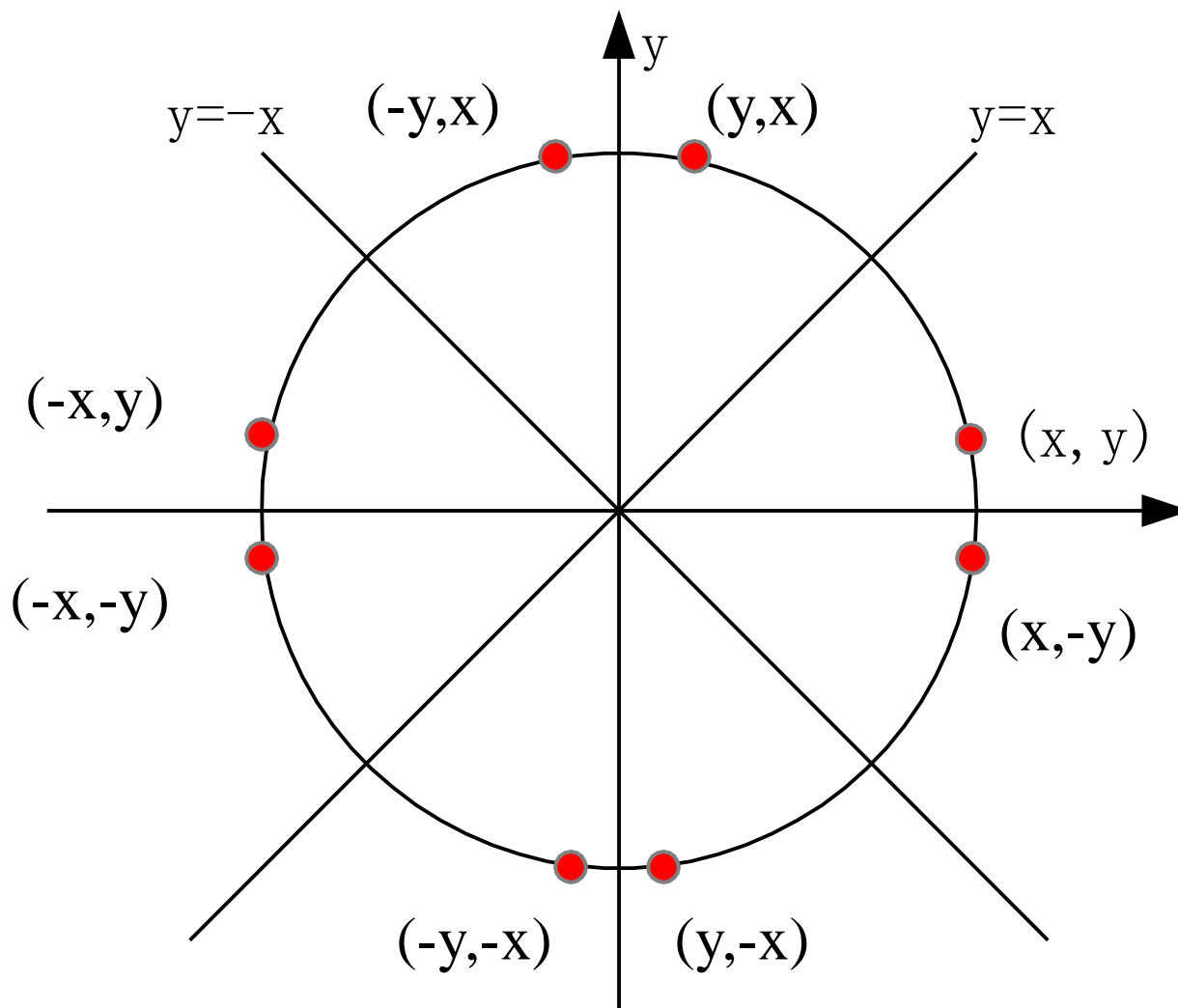


图5.11 八分法画圆

圆的扫描转换

解决问题：

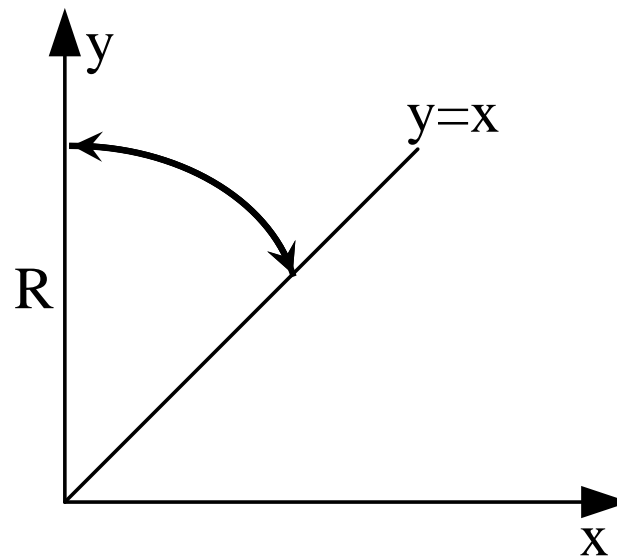


图5.12 1/8圆弧

圆的扫描转换

1

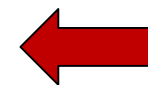
八分法画圆

2

简单方程生成圆弧

3

中点Bresenhan算法画圆



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：

$$x = R \cos \theta$$

$$y = R \sin \theta$$

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

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

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

圆的扫描转换

1

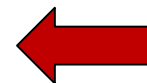
八分法画圆

2

简单方程生成圆弧

3

中点Bresenhan算法画圆



中点Bresenham画圆

给定圆心在原点，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， $F(x,y)<0$ 。

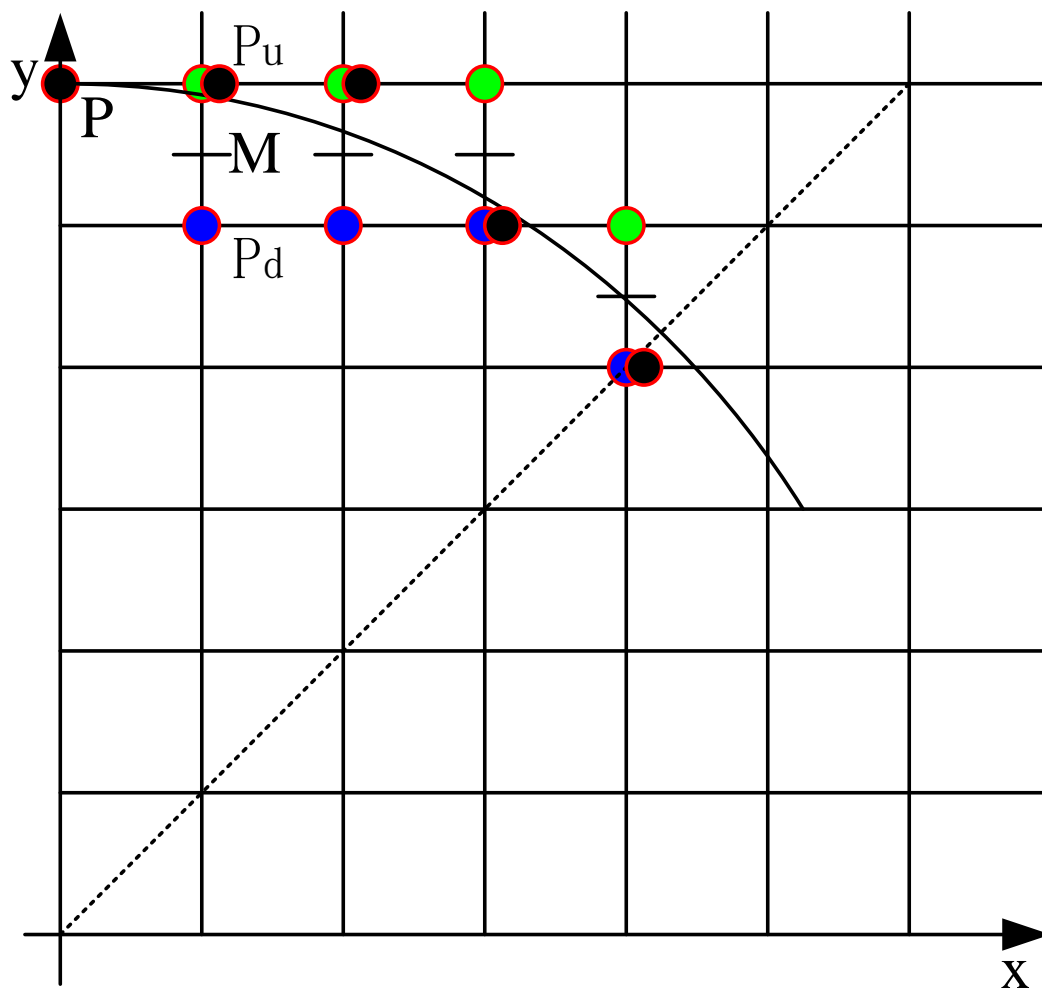


图5.13 中点Bresenham画圆的原理

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 ($d \leq 0$)

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

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

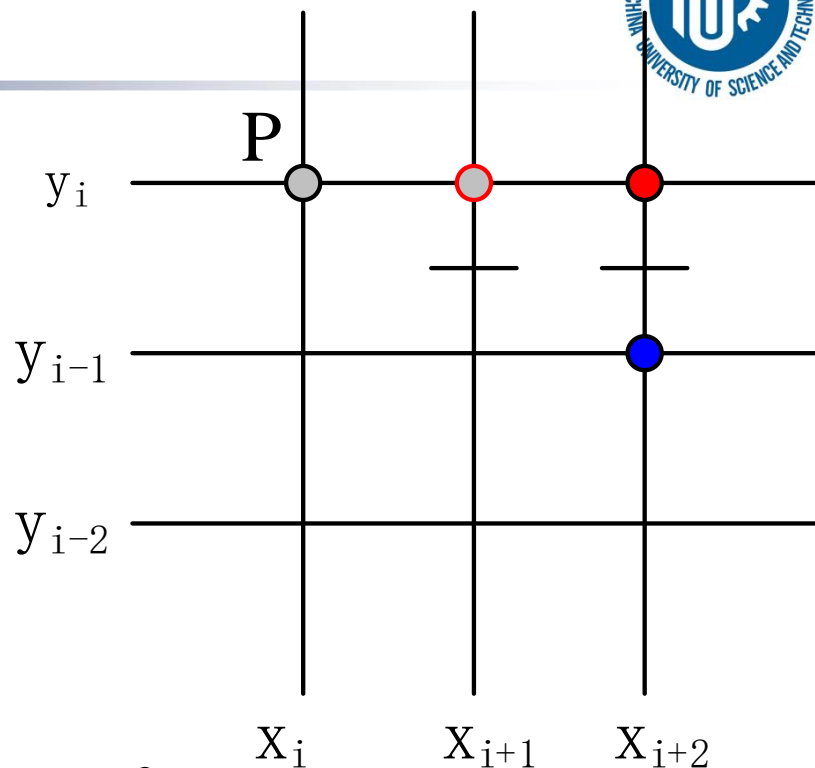


图5.14 $d \leq 0$ 的情况

误差项的递推 ($d > 0$)

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

$$\begin{aligned} d_2 &= (x_i + 1 + 1)^2 + (y_i - 0.5 - 1)^2 - R^2 \\ &= (x_i + 1)^2 + 2x_i + 3 + (y_i - 0.5)^2 - 2(y_i - 0.5) + 1 - R^2 \\ &= (x_i + 1)^2 + (y_i - 0.5)^2 + 2(x_i - y_i) + 5 \\ &= d_1 + 2(x_i - y_i) + 5 \end{aligned}$$

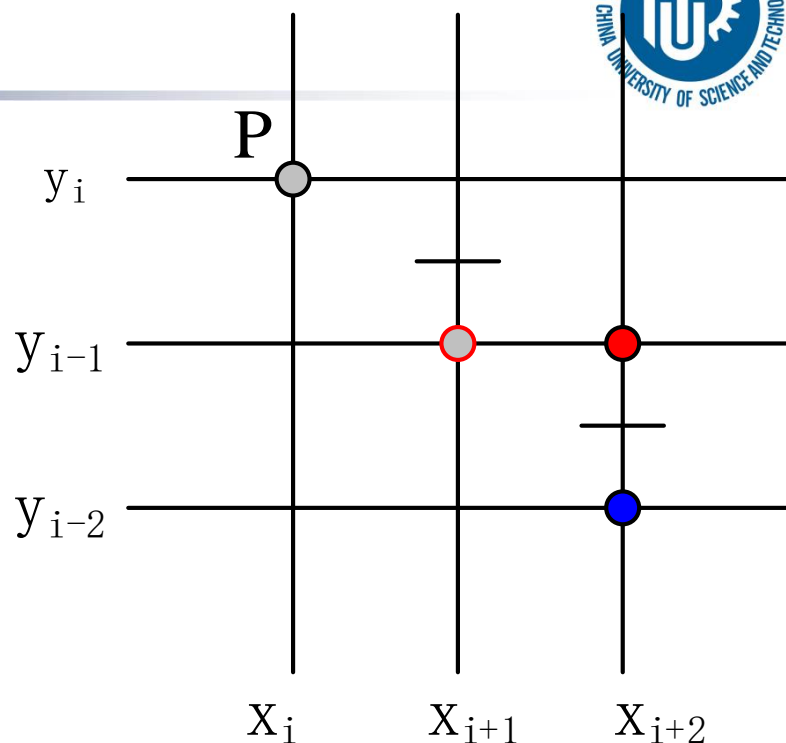


图5.15 $d > 0$ 的情况

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：

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

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

$$d_0 = 1 - R$$

$$\begin{array}{ccc} d \leq -0.25 & \longrightarrow & d < 0 \\ d > -0.25 & & d \geq 0 \end{array}$$

中点Bresenham画圆——算法步骤

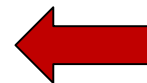


1. 输入圆的半径 R 。
2. 计算初始值 $d=1-R$ 、 $x=0$ 、 $y=R$ 。
3. 绘制点 (x,y) 及其在八分圆中的另外七个对称点。
4. 判断 d 的符号。若 $d<0$ ，则先将 d 更新为 $d+2x+3$ ，再将 (x,y) 更新为 $(x+1,y)$ ；否则先将 d 更新为 $d+2(x-y)+5$ ，再将 (x,y) 更新为 $(x+1,y-1)$ 。
5. 当 $x<y$ 时，重复步骤3和4。否则结束。 [程序演示](#)

多边形的扫描转换与区域填充

1

多边形的扫描转换



2

区域填充

3

多边形扫描转换与区域填充的比较

4

相关概念

多边形的扫描转换

- 顶点表示用多边形的顶点序列来刻划多边形。
- **点阵表示**是用位于多边形内的像素的集合来刻划多边形。
- **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。

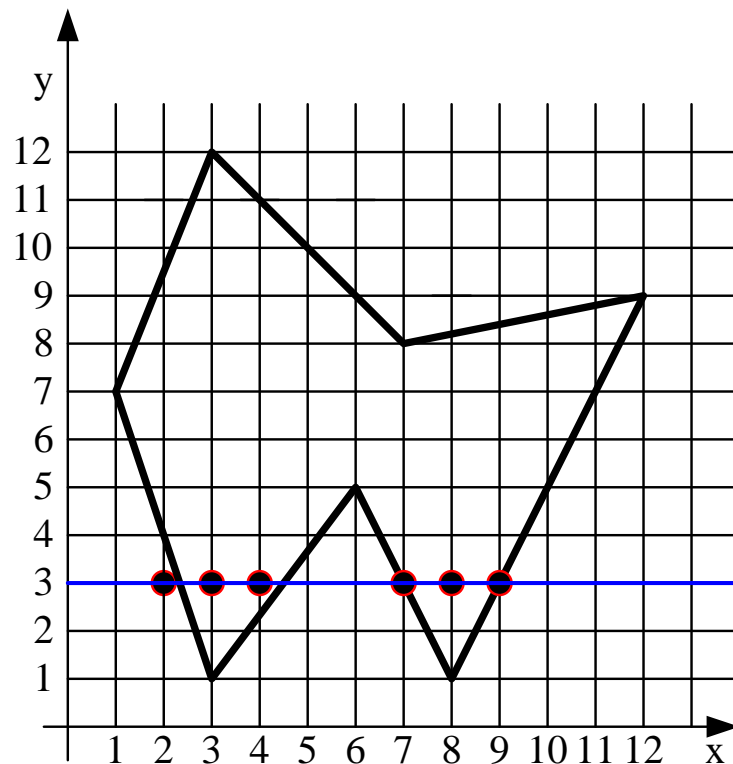


图5.16 x-扫描线算法填充多边形

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；

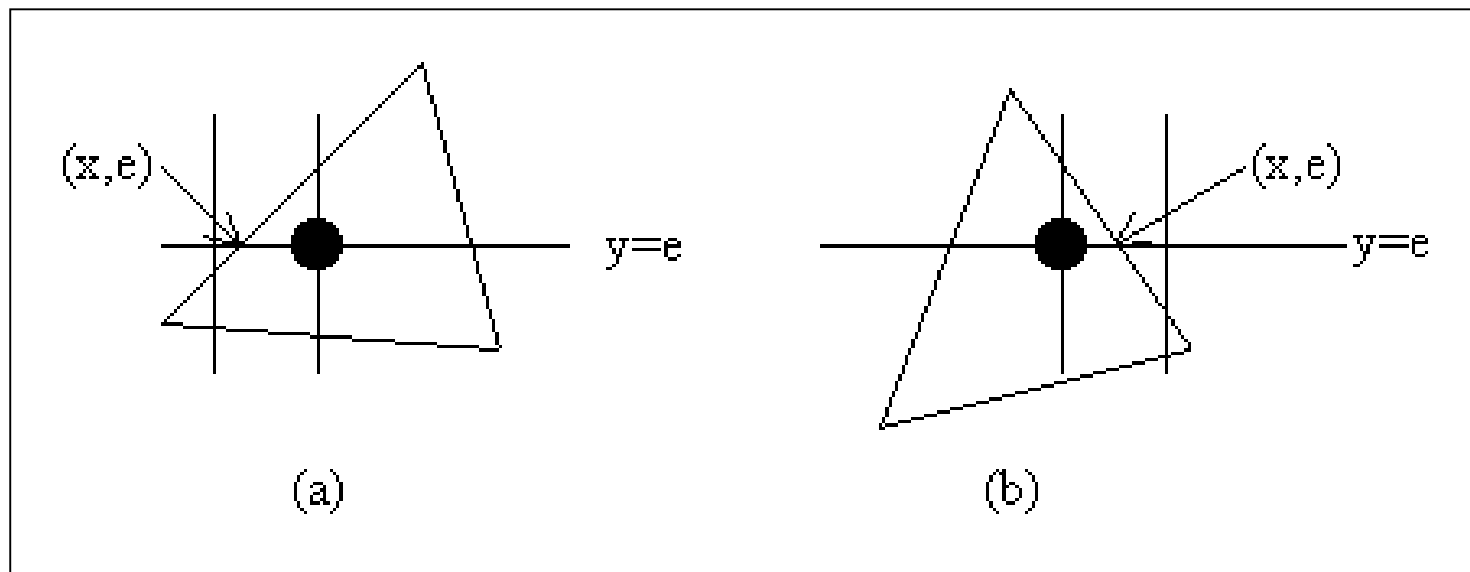


图5.17 取整规则1

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**）

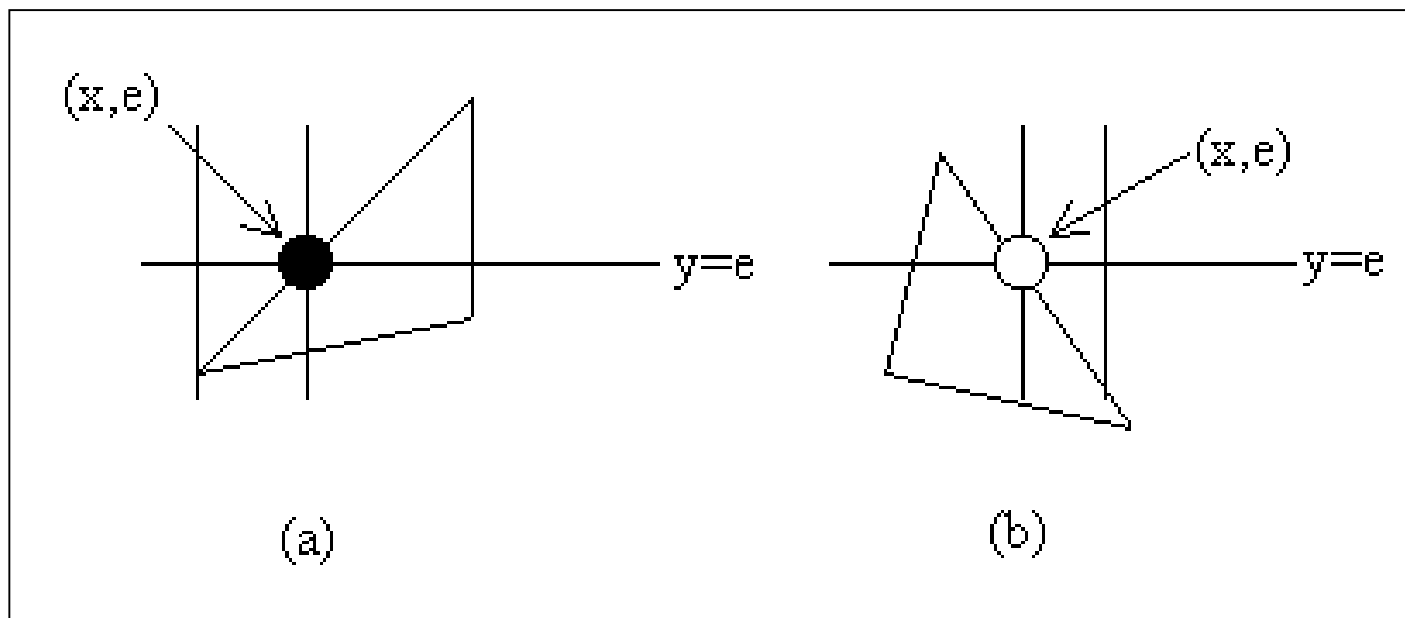


图5.18 取整规则2

- **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。

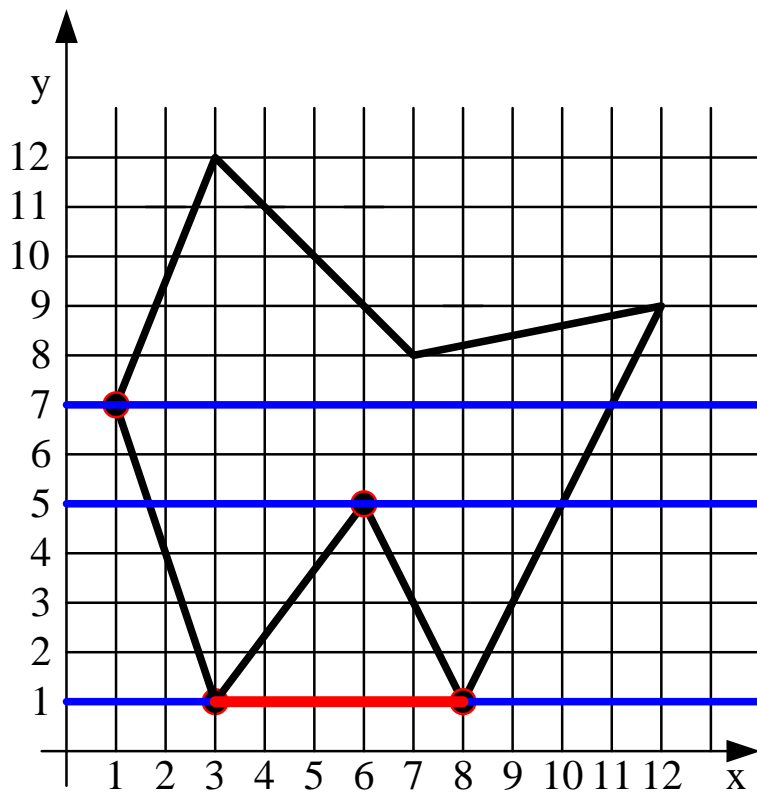


图5.19 取整规则3

X-扫描线算法——取整规则

交点的取舍解决方法:

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。

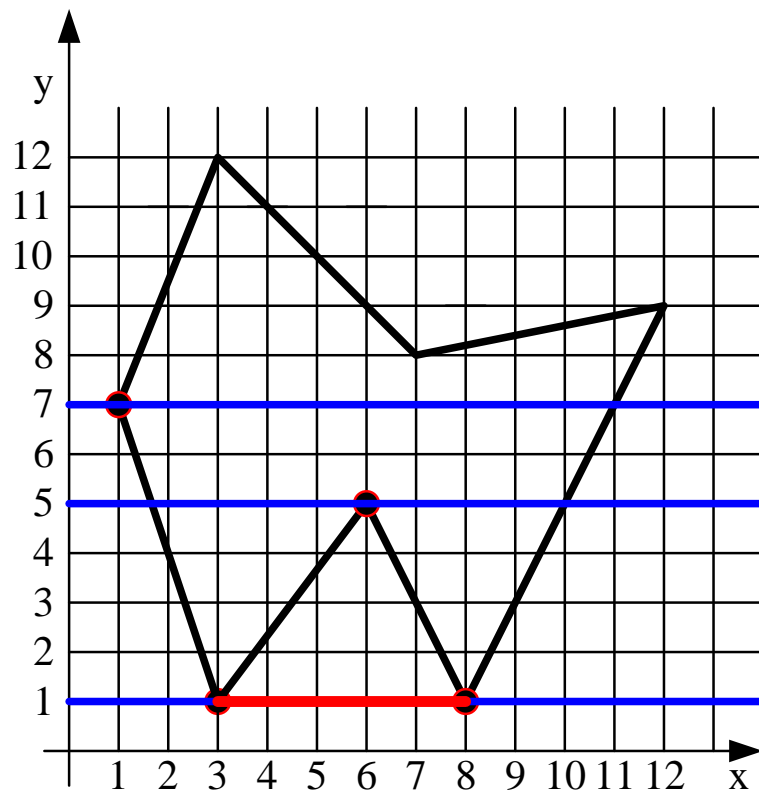


图5.20 取整规则3

X-扫描线算法——取整规则

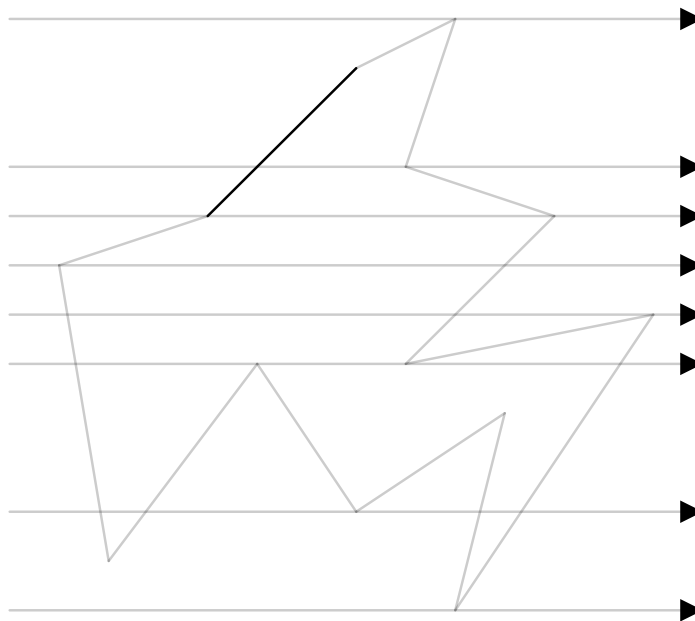


图5.21 与扫描线相交的多边形顶点的交点数

填充过程实例

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。

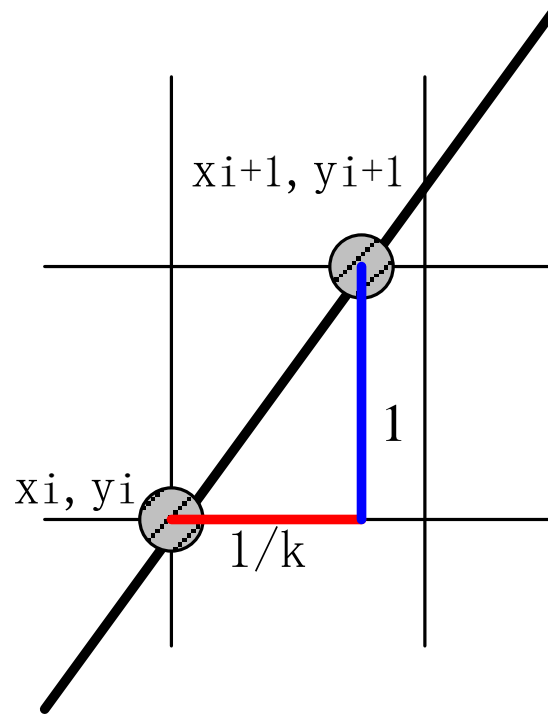
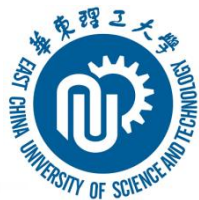


图5.22 与多边形边界相交的两条连续扫描线交点的相关性

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:

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

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：

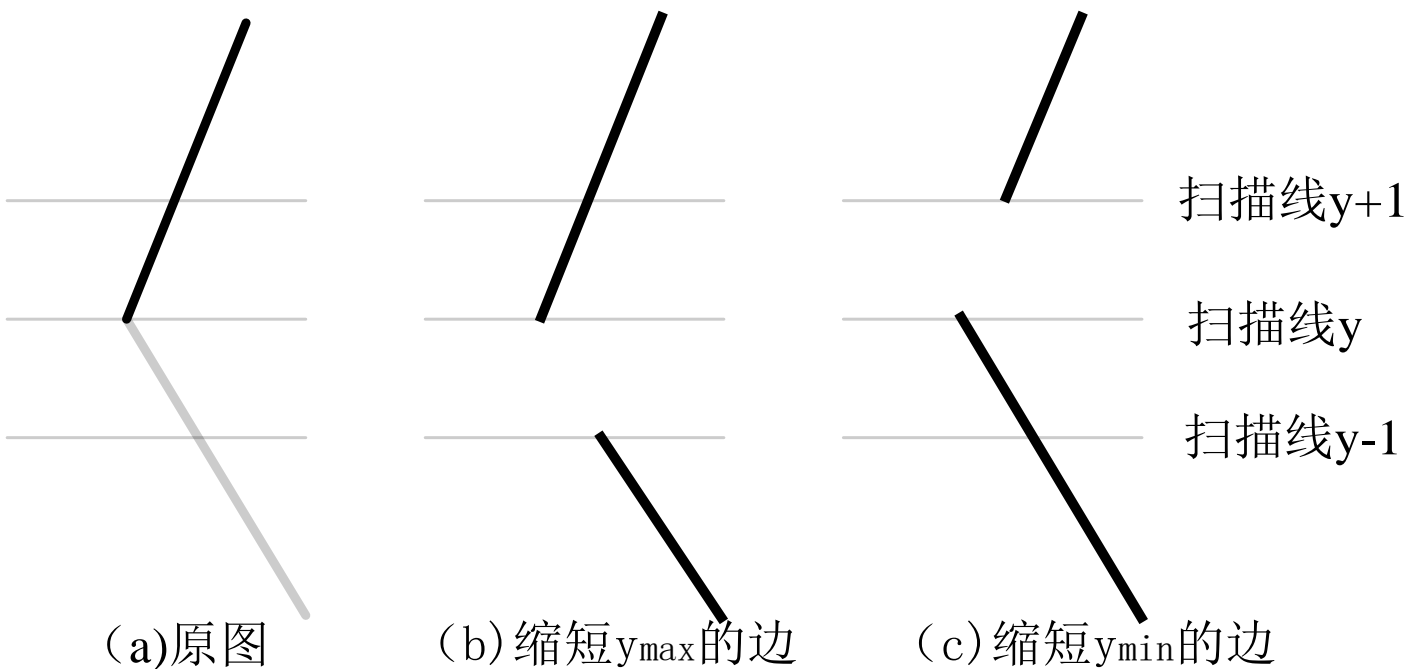
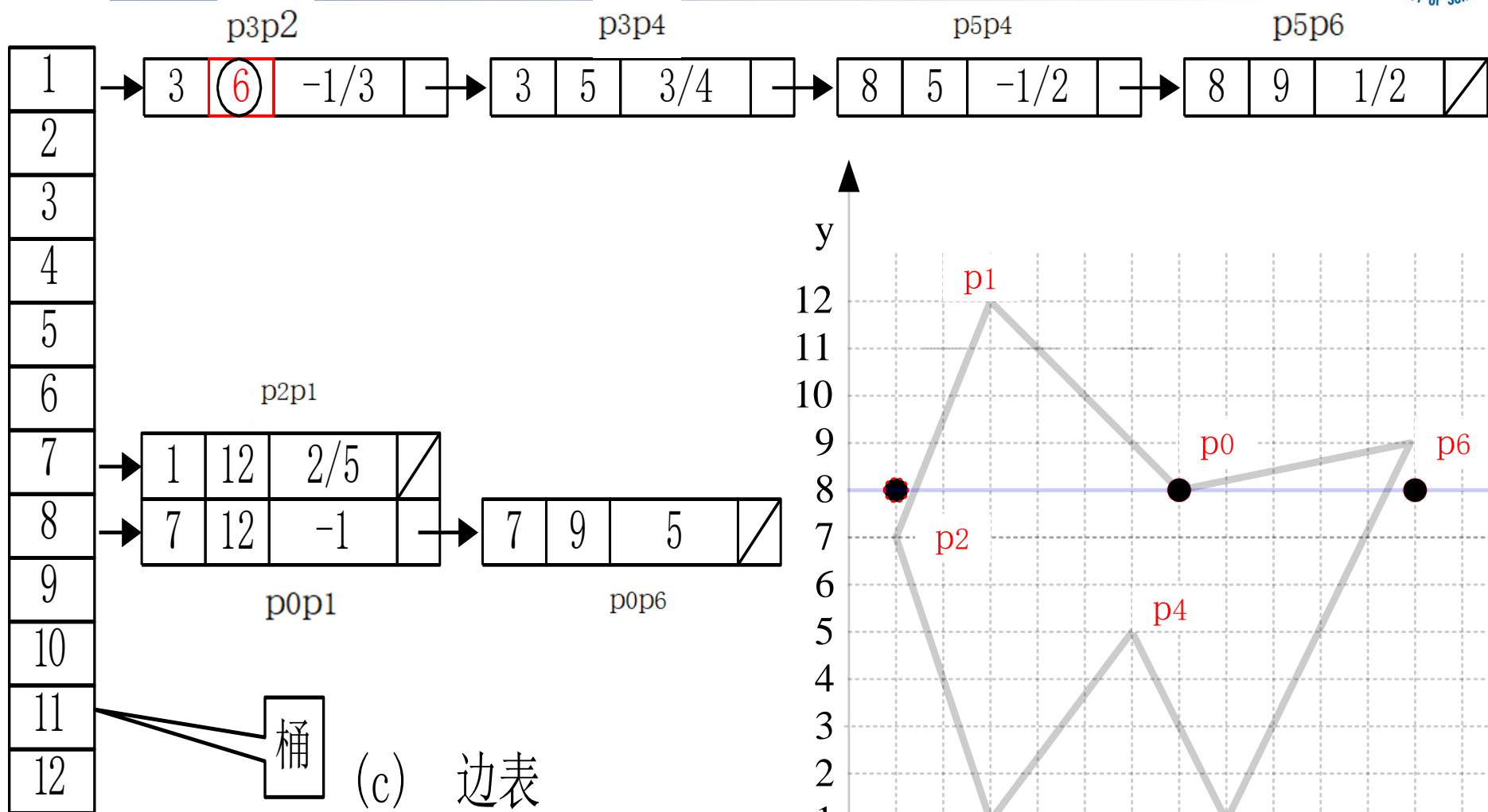


图5-23 将多边形的某些边缩短以分离那些应计为1个交点的顶点





(c) 边表

图5.24 多边形 $P_0P_1P_2P_3P_4P_5P_6$

- (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。

填充过程实例

改进的有效边表算法利用的连贯性有（）

- A. 多边形边的连贯性
- B. 多边形边的连贯性、扫描线之间的连贯性
- C. 扫描线之间的连贯性、扫描线内部配对的连贯性
- D. 多边形边的连贯性、扫描线之间的连贯性、扫描线内部配对的连贯性

改进的有效边表算法, 利用(), 记录所有的边信息。

- A. 扫描线的连贯性
- B. 边的连贯性
- C. 边表ET
- D. 有效边表AET

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。

- 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。

多边形的扫描转换与区域填充

1

多边形的扫描转换

2

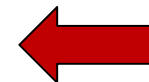
区域填充

3

多边形扫描转换与区域填充的比较

4

相关概念



区域填充

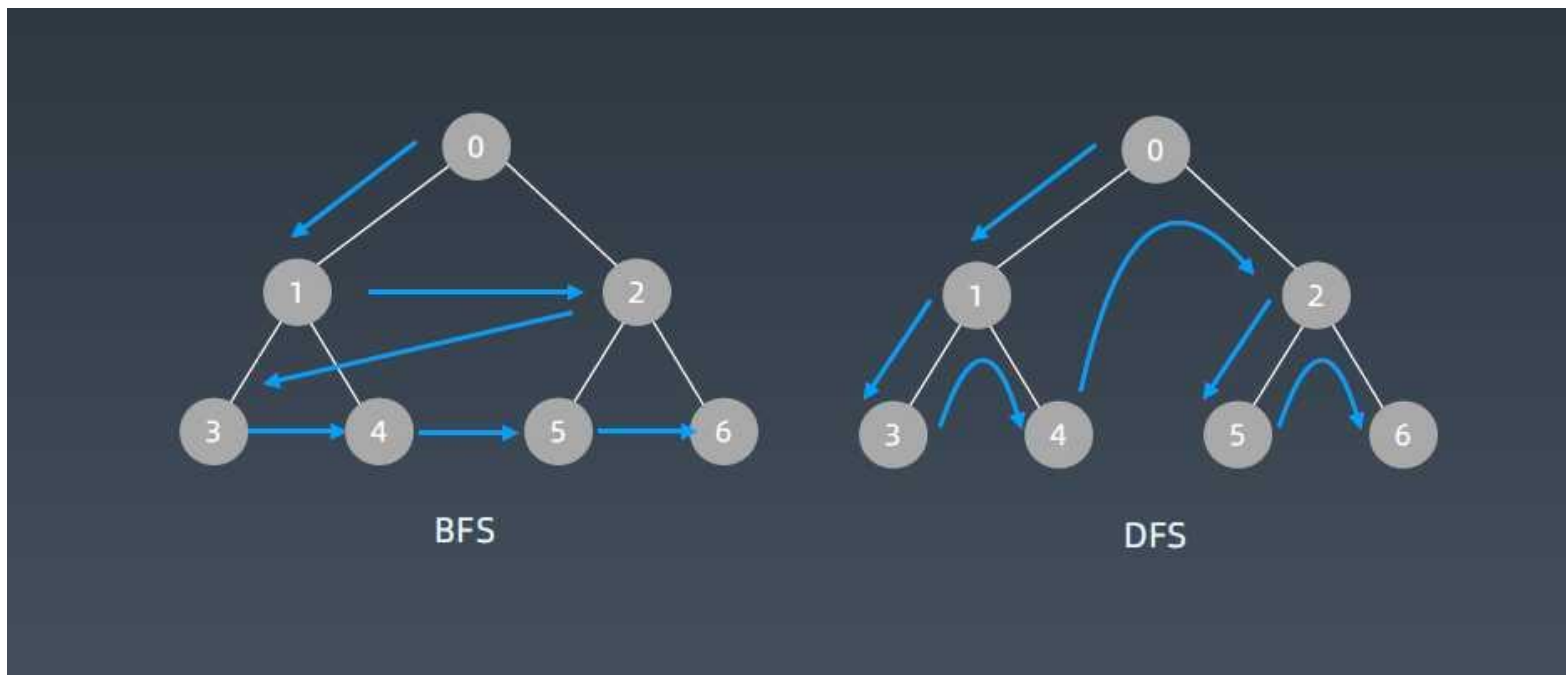
- 基本概念
- 区域的表示方法
- 区域的分类
- 区域填充算法



(图片来源网络)

DFS

- - 深度优先: depth first search
- - 广度优先: breadth first search



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。

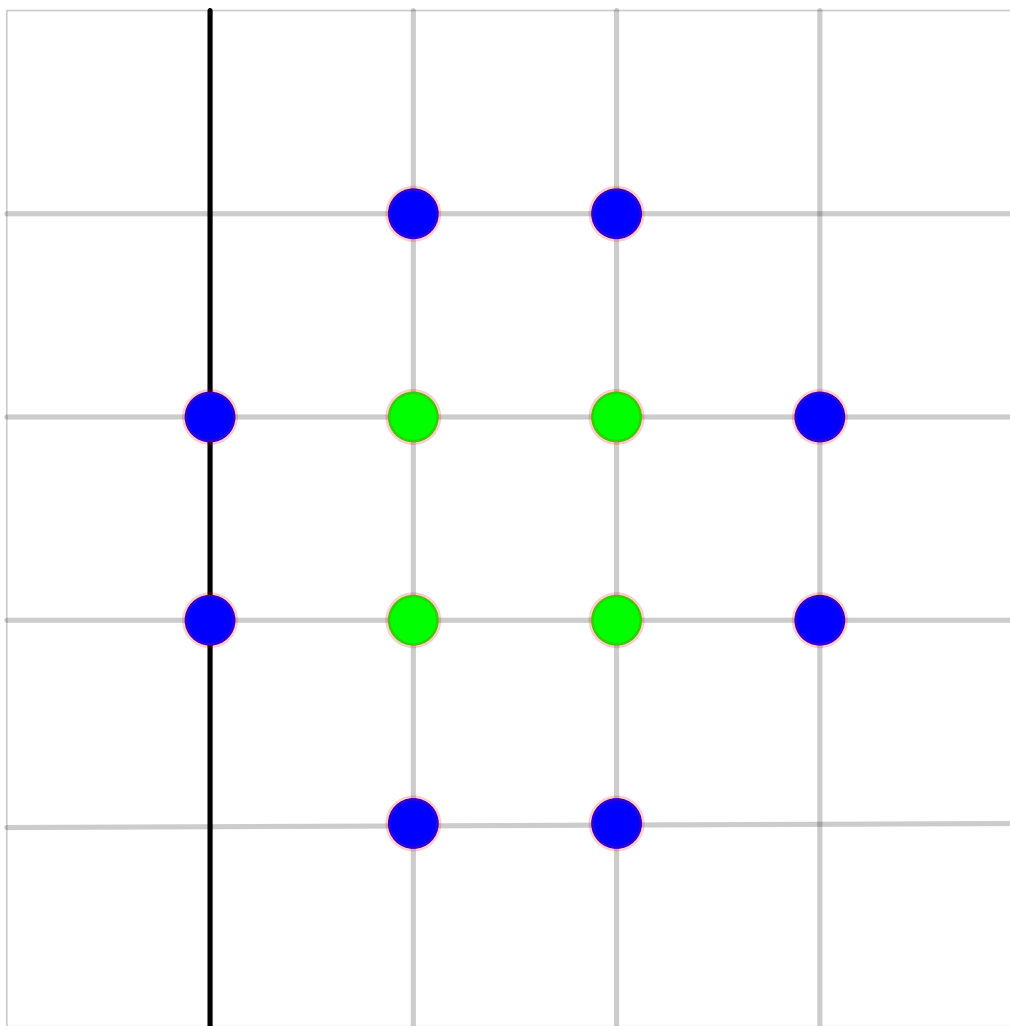


图5-25 区域的概念

区域的表示方法

- **边界表示法**：把位于给定区域的边界上的像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，填充算法可以逐个像素地向外处理，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）。

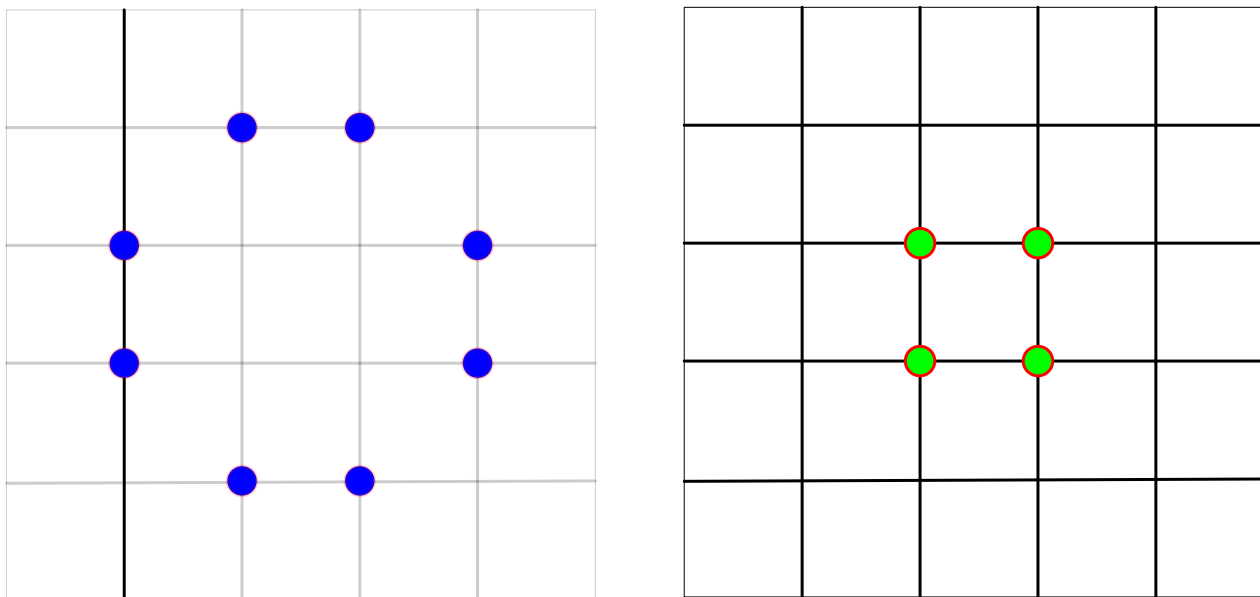
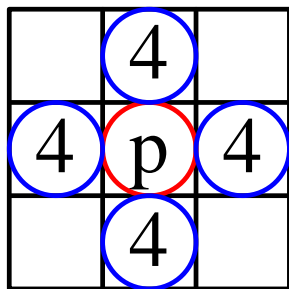


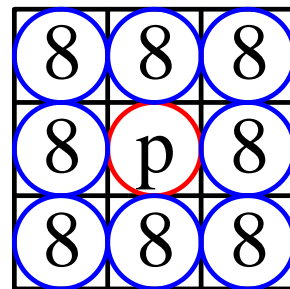
图5-26 区域的表示方法

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5-27 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。

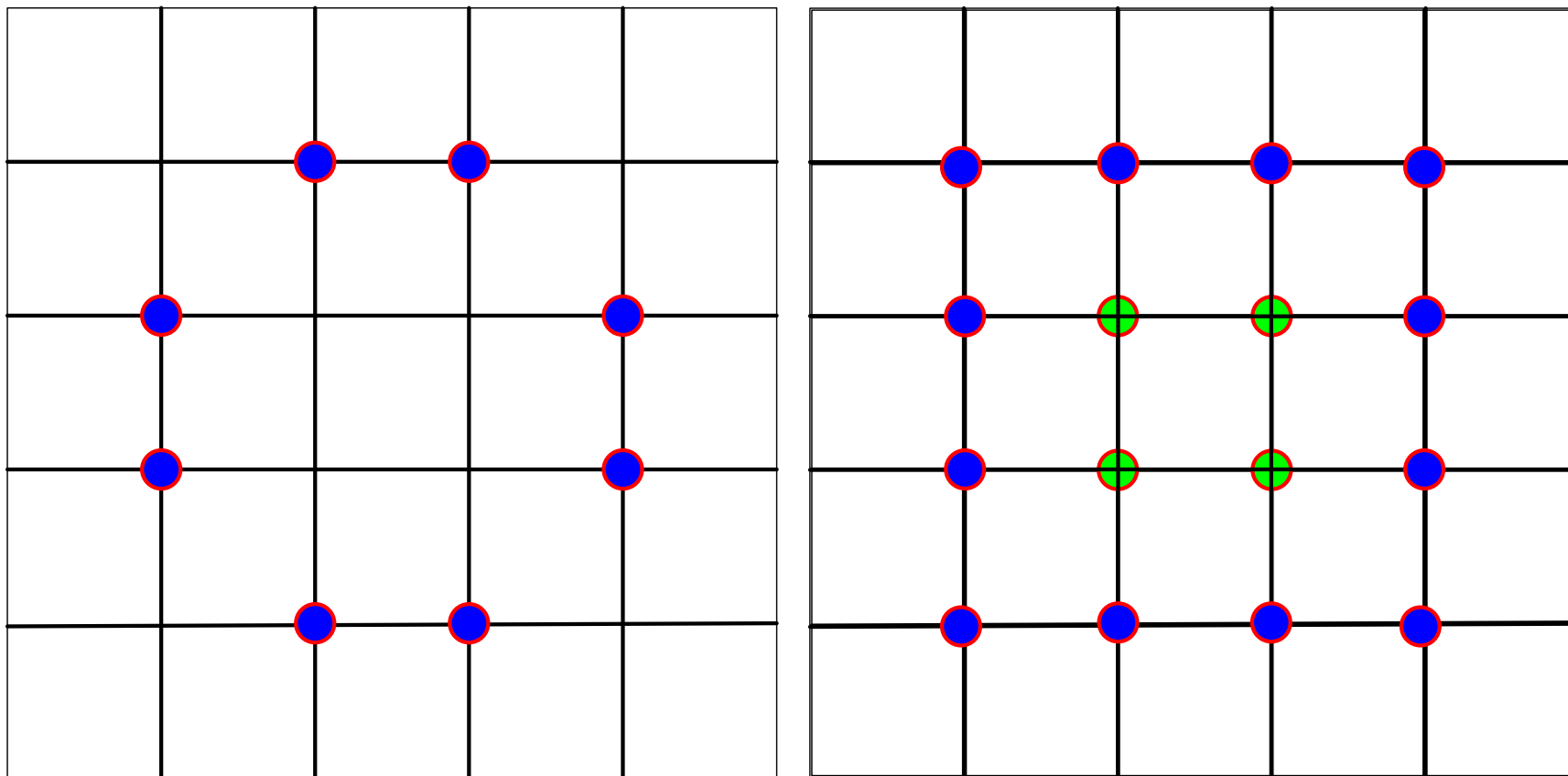


图5-28 4 - 连通与8 - 连通区域

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。

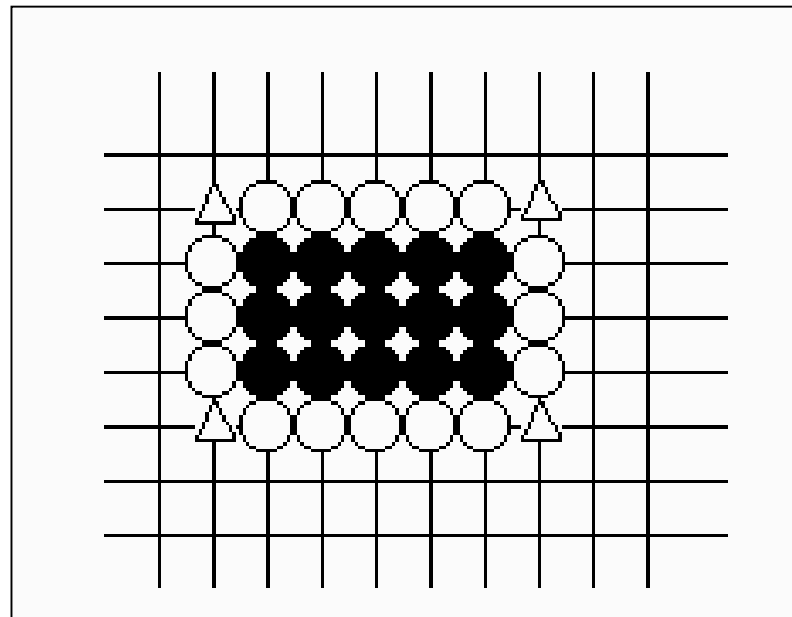


图5-29 4 - 连通与8 - 连通区域

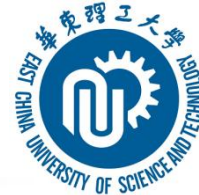
区域填充算法

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

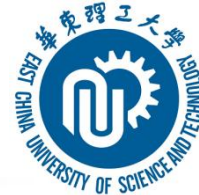


栈结构实现4-连通边界填充算法的算法步骤为：

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。

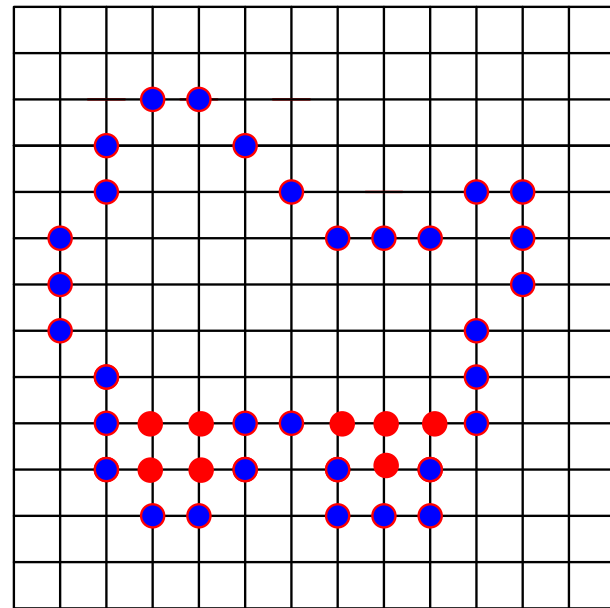
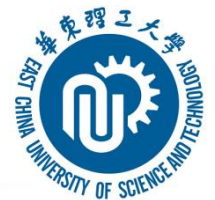


图5-30 扫描线种子填充算法

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。

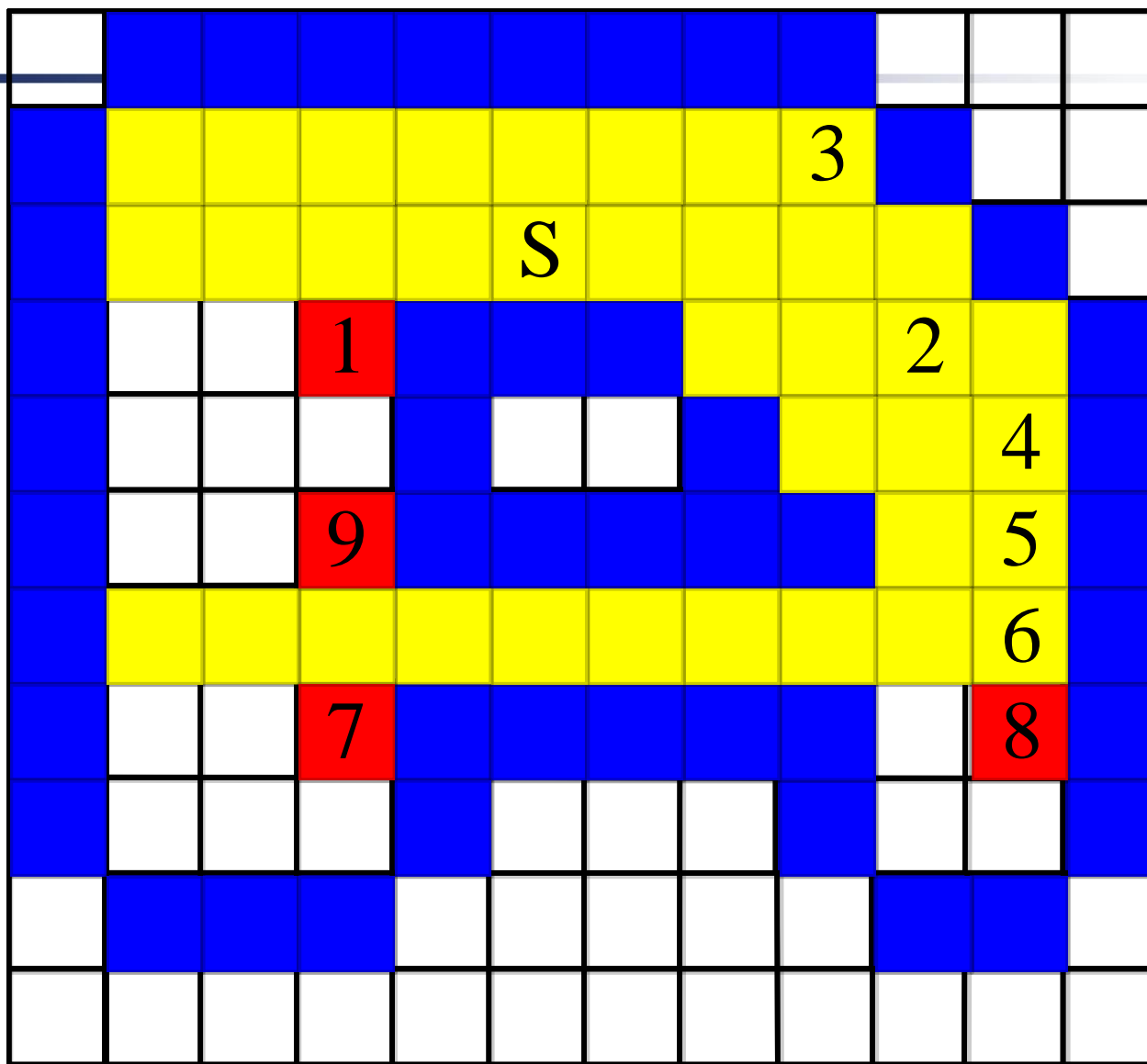


图5-31 扫描线种子填充算法过程

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。

多边形的扫描转换与区域填充

1

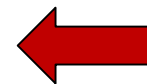
多边形的扫描转换

2

区域填充

3

多边形扫描转换与区域填充的比较



4

相关概念

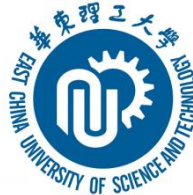
多边形的扫描转换与区域填充

- **多边形的扫描转换**主要是通过确定穿越区域的扫描线的覆盖区间来填充。
- **区域填充**是从给定的位置开始涂描直到指定的边界条件为止。

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不同点



- **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。

多边形的扫描转换与区域填充

1

多边形的扫描转换

2

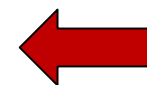
区域填充

3

多边形扫描转换与区域填充的比较

4

相关概念



相关概念——内外测试

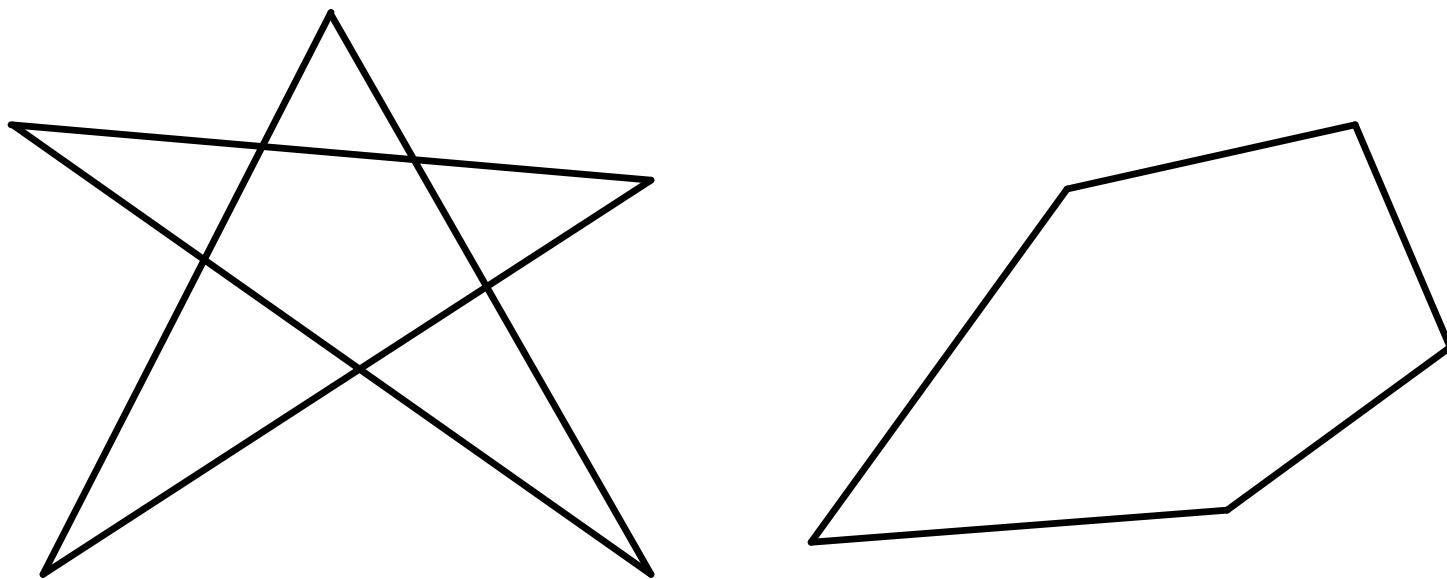
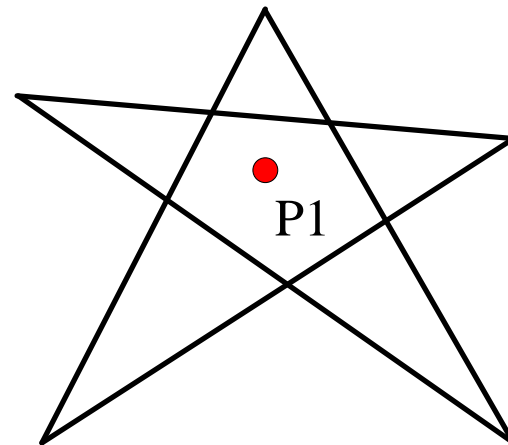


图5-32 不自交的多边形与自相交的多边形

相关概念——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。



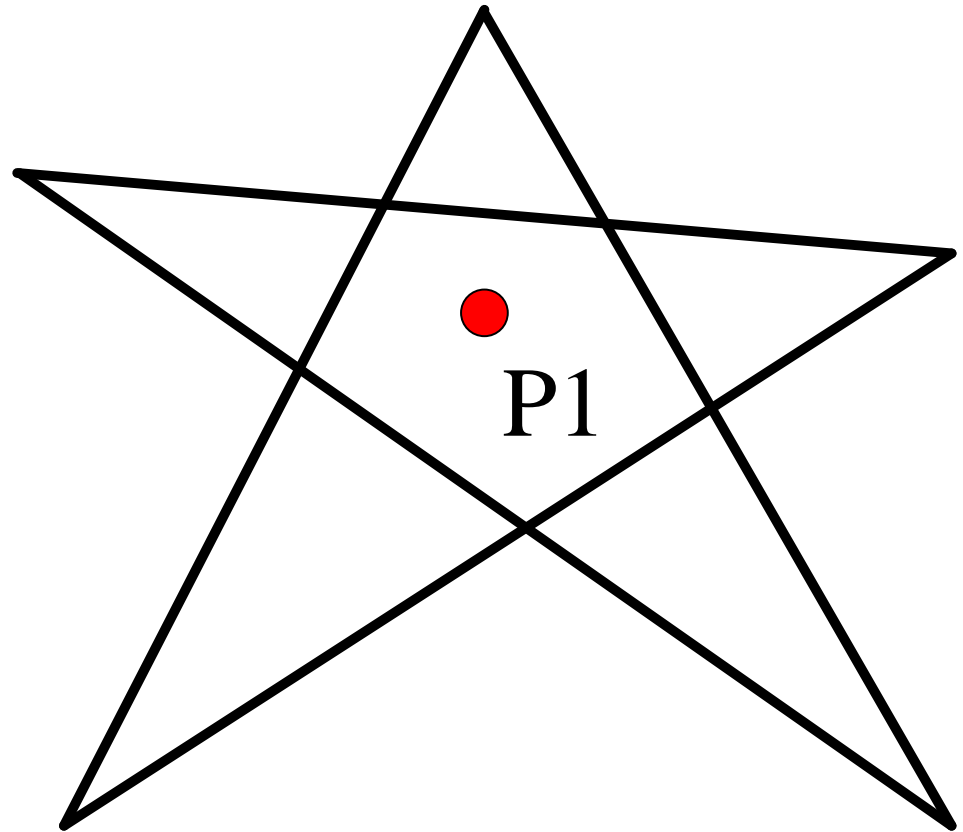
相关概念——内外测试

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。

相关概念——内外测试

- 两种规则的比较



谢谢！

*Thank
You*