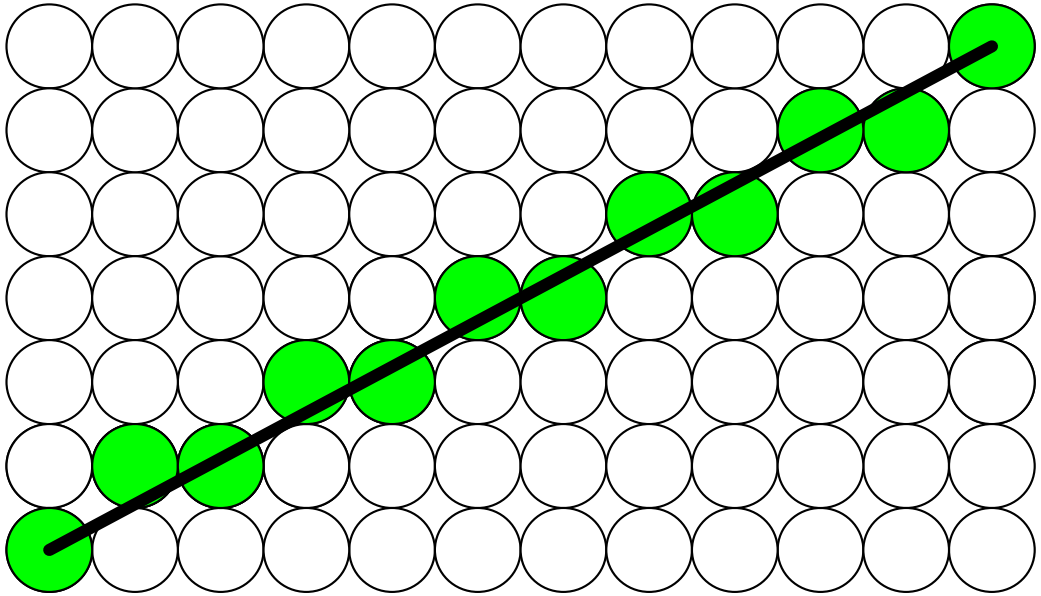
基本图形生成算法

华中科技大学
何云峰

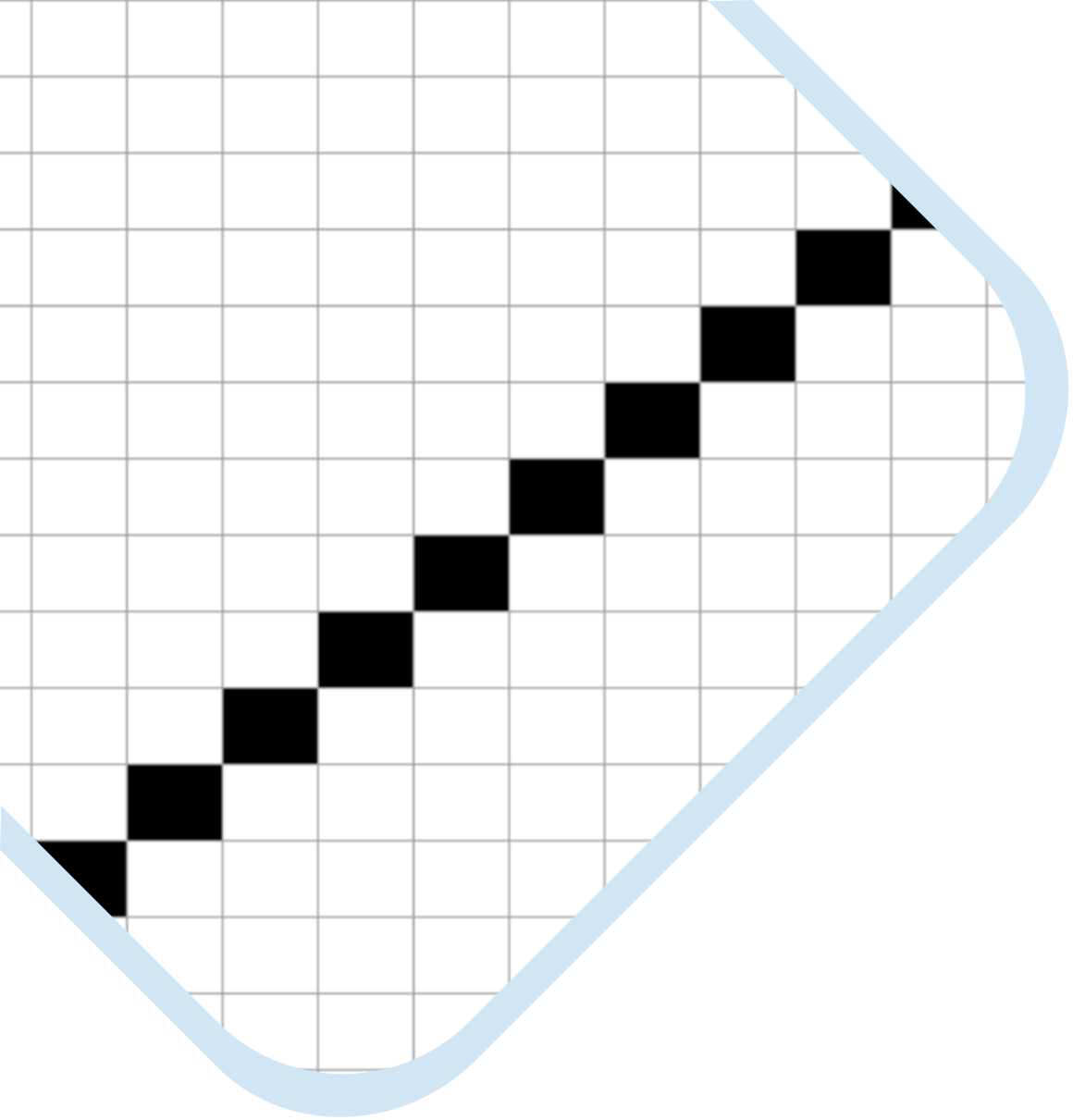
CONTENTS

扫描转换

- 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象素集的过程。



用象素点集逼近直线



PART 01

直线段的绘制

01 直线段的绘制

- 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

01 直线段的绘制

- 解决的问题：给定直线两端点P0(x0,y0)和P1(x1,y1)，画出该直线
  - 数值微分法
  - 中点Bresenhan算法

01 直线段的绘制

数值微分法

直线的微分方程

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

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

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

ε=1/max(|△x|,|△y|)

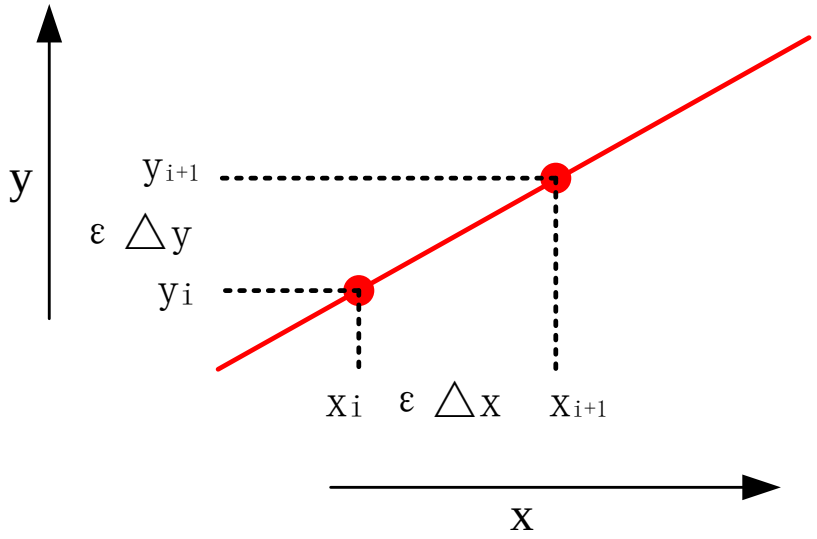


图4.2 DDA算法原理

01 直线段的绘制

- 数值微分法
  - max(|△x|,|△y|)=|△x|，即|k|≤1的情况

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

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

01 直线段的绘制

数值微分法

- max(|△x|,|△y|)=|△y|，此时|k|≥1

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

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

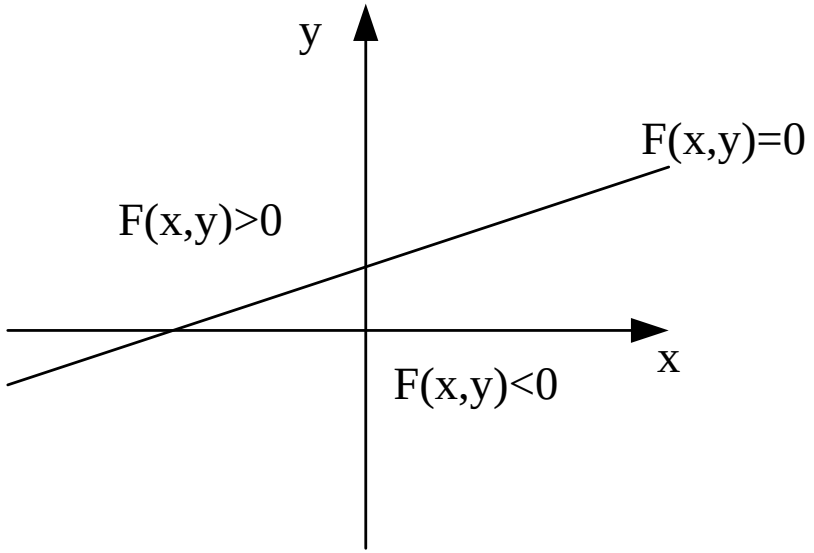
01 直线段的绘制

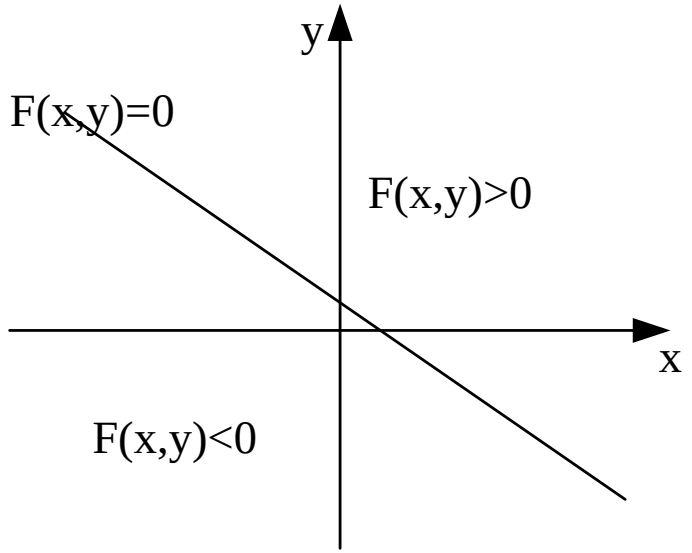
- 数值微分法
  - 特点
    - 增量算法
    - 直观、易实现
    - 不利于用硬件实现

中点Bresenham算法

直线的方程

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



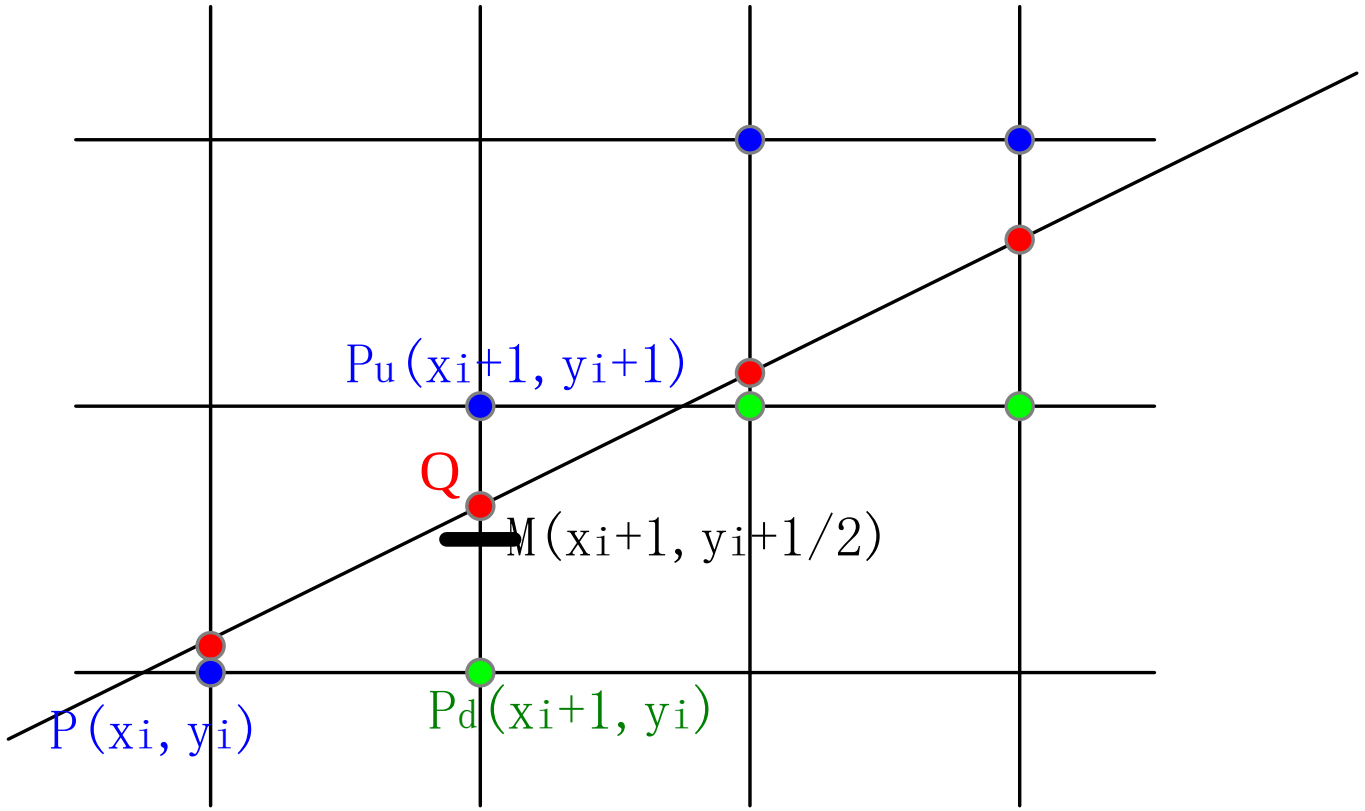


01 直线段的绘制

- 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
    - 增量值取决于实际直线与相邻象素点的距离，这一距离称为误差项

直线段的绘制

- 中点Bresenham算法
  - 假定0≤k≤1，即0≤ Δy/Δ x ≤1，x是最大位移方向



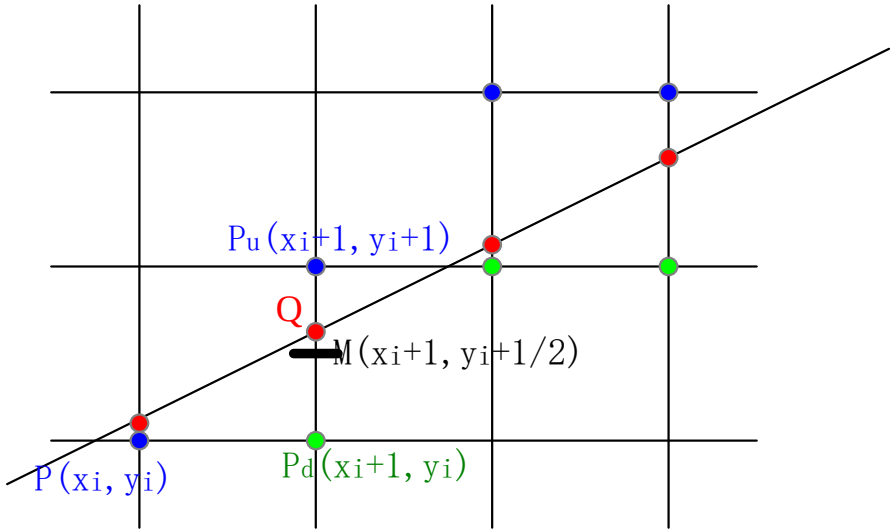
Brensemham算法生成直线的原理

中点Bresenham算法

判别式

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

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



Brensemham算法生成直线的原理

直线段的绘制

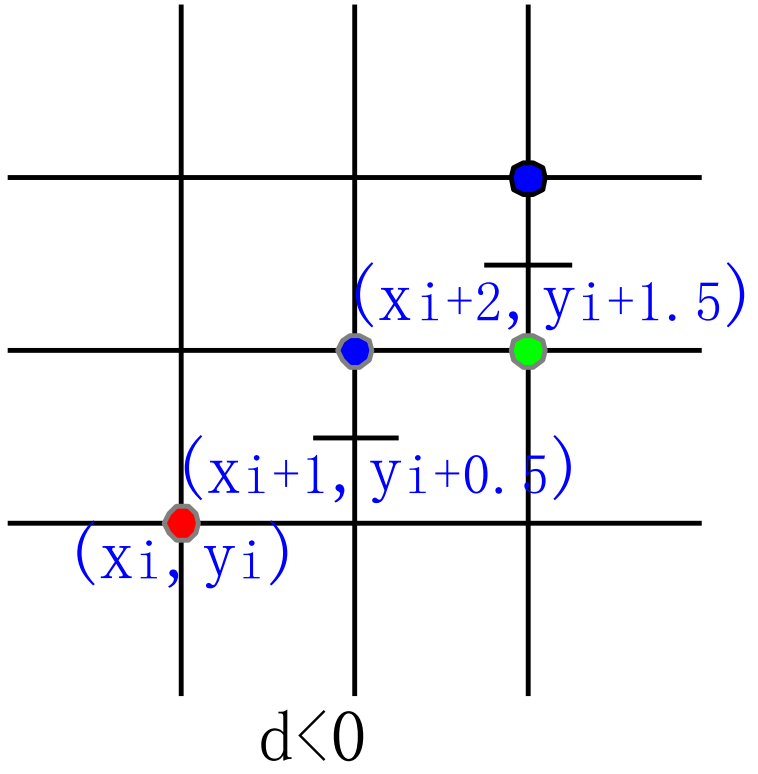
中点Bresenham算法

误差项的递推（d<0）

$$d_1 = F(x_i + 1, y_i + 0.5)$$
$$= y_i + 0.5 - k(x_i + 1) - b$$

$$d_2 = F(x_i + 2, y_i + 1.5)$$
$$= y_i + 1.5 - k(x_i + 2) - b$$

$$d_2 = y_i + 0.5 - k(x_i + 1) - b + 1 - k$$
$$= d_1 + 1 - k$$



误差项递推

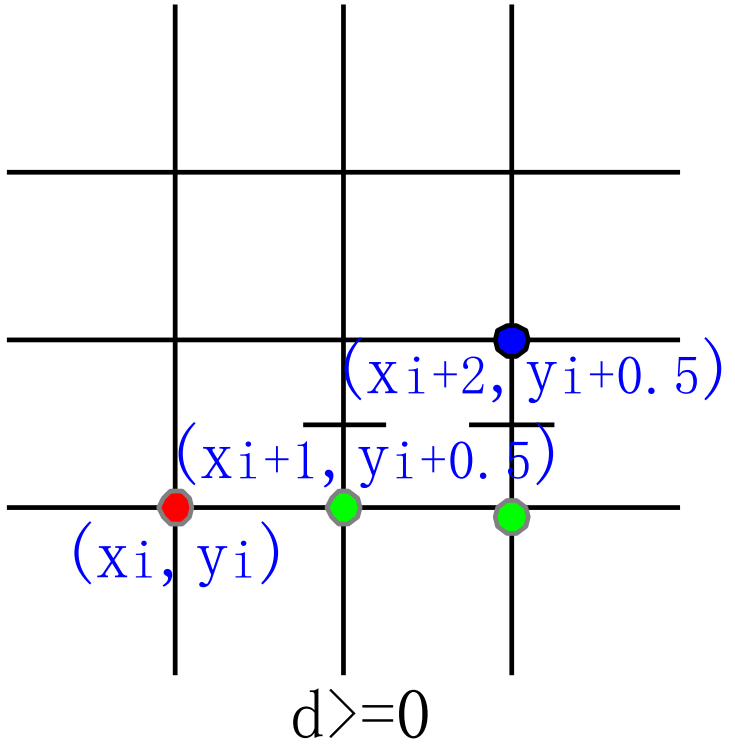
直线段的绘制

- 中点Bresenham算法
  - 误差项的递推（ d≥0 ）

$$d_1 = F(x_i + 1, y_i + 0.5)$$
$$= y_i + 0.5 - k(x_i + 1) - b$$

$$d_2 = F(x_i + 2, y_i + 0.5)$$
$$= y_i + 0.5 - k(x_i + 2) - b$$

$$d_2 = y_i + 0.5 - k(x_i + 1) - b - k$$
$$= d_1 - k$$



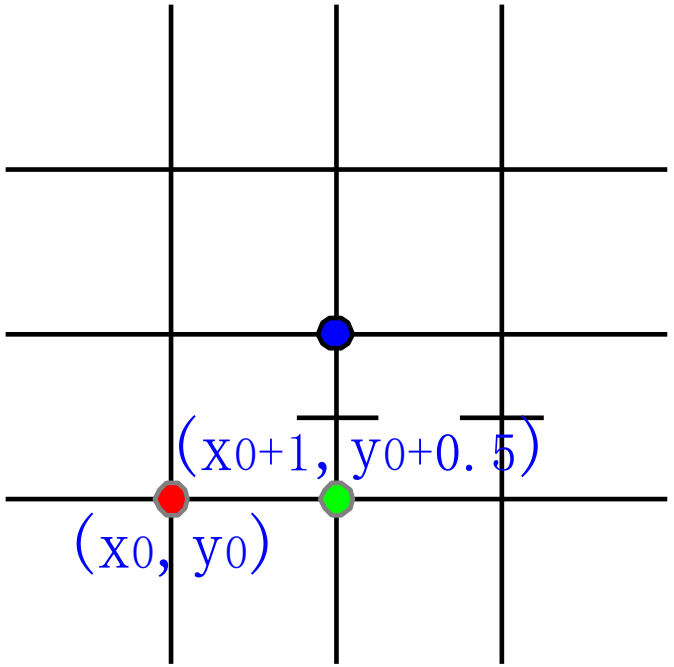
误差项递推

直线段的绘制

中点Bresenham算法

初值

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



计算初值

直线段的绘制

- 中点Bresenham算法
  - 问题：计算中存在浮点数
  - 改进：用2d△x代替d ,令D = 2d△x

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

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

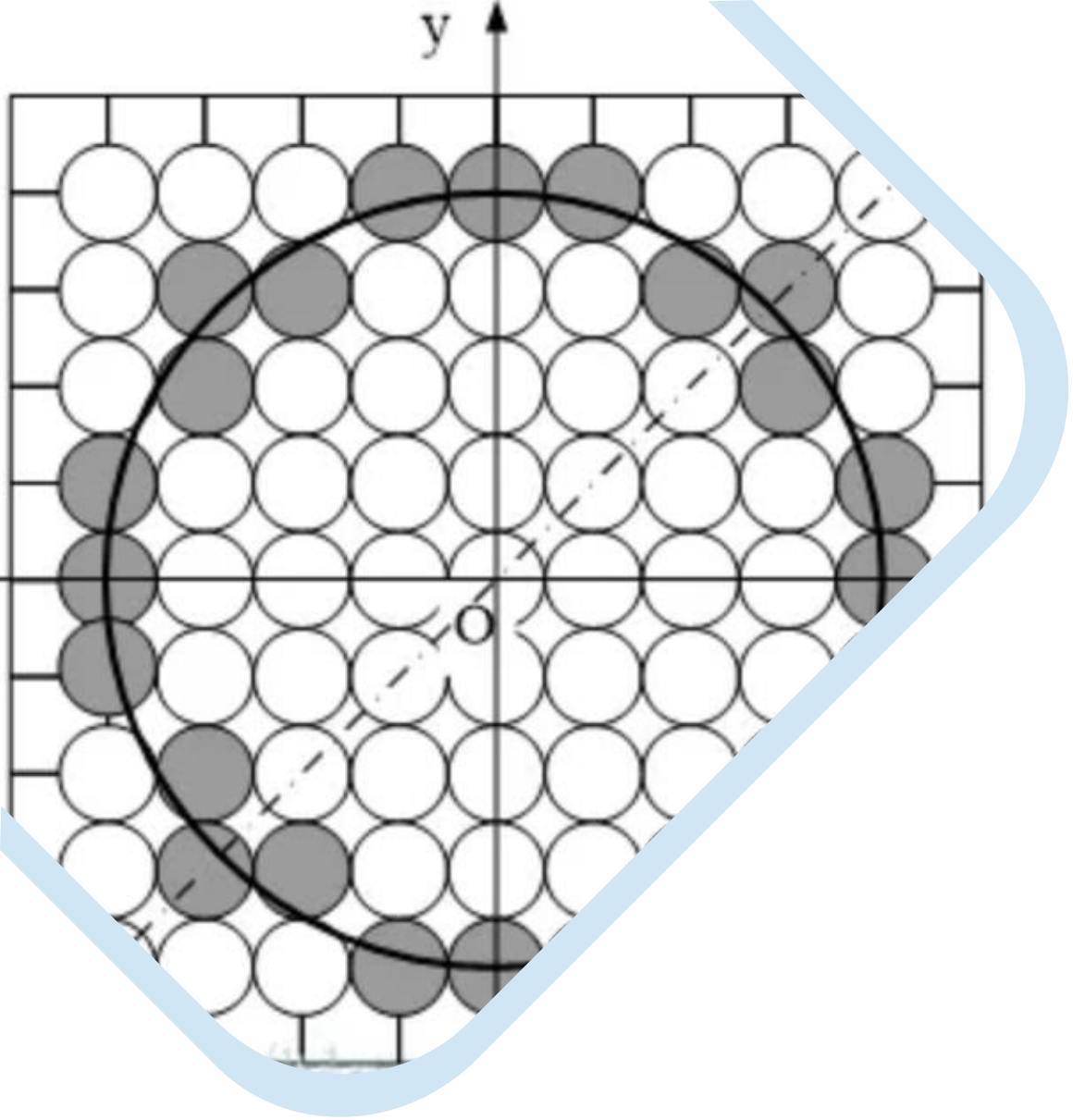
$$D = 2\Delta x d = 2\Delta x(d+1-k)$$

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

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

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

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

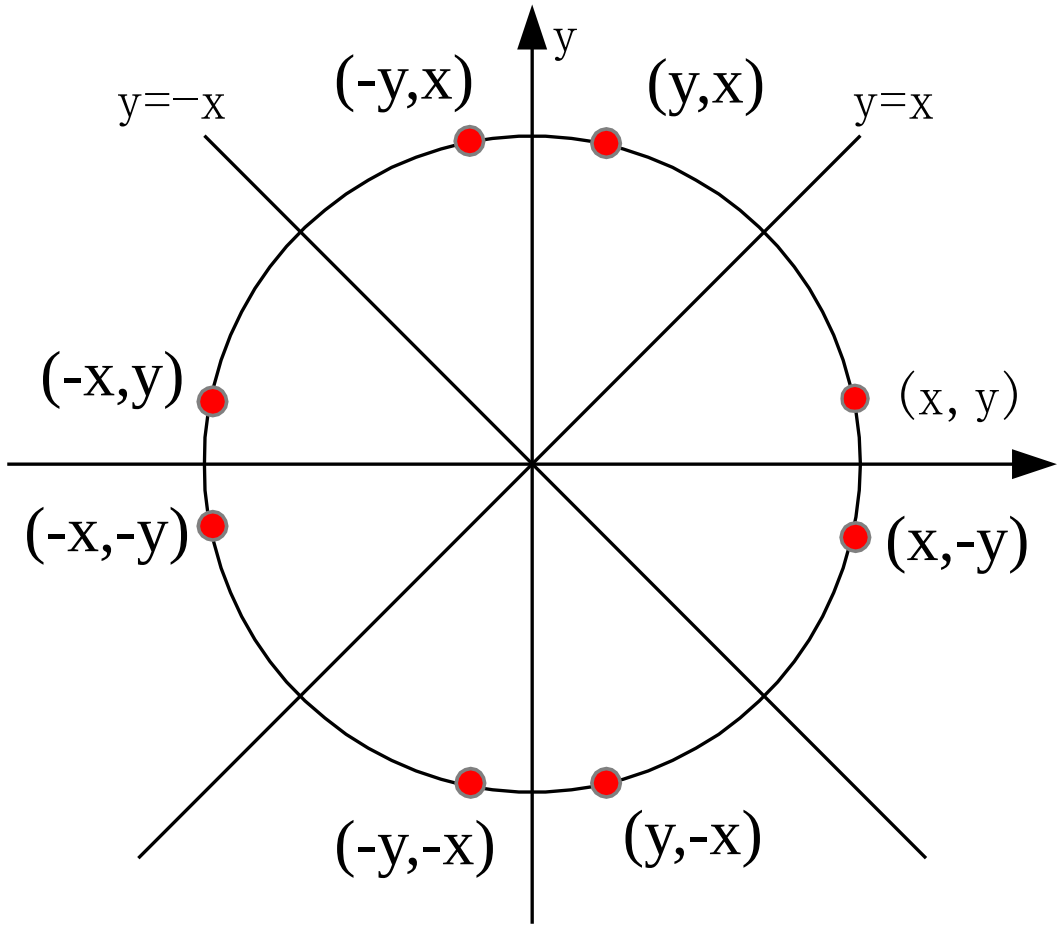


PART 02

圆的绘制

圆的绘制

- 解决的问题：绘出圆心在原点，半径为整数R的圆$x^2+y^2=R^2$

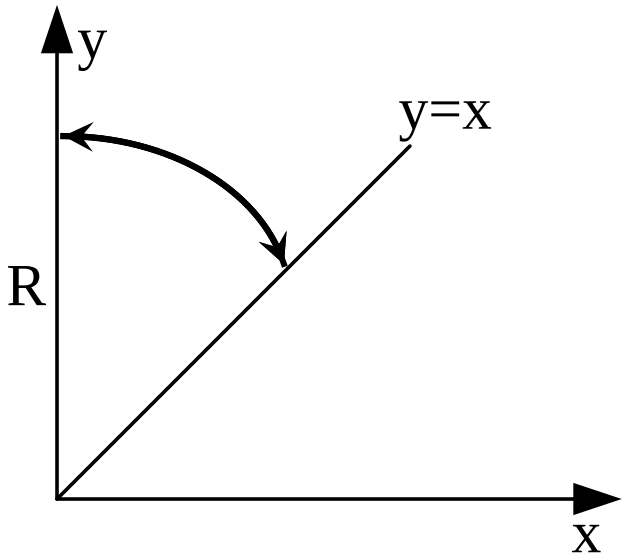


八分法画圆

02

圆的绘制

解决的问题



1/8圆弧

02

圆的绘制

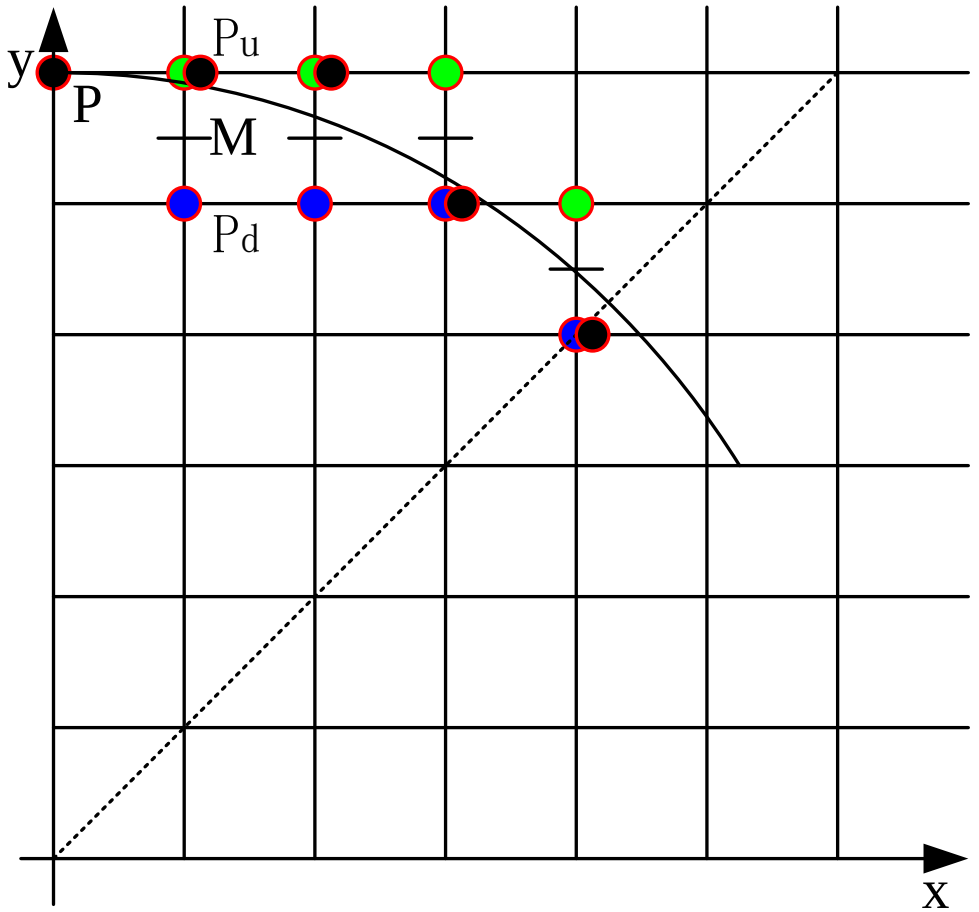
中点Bresenham算法画圆

- 给定圆心在原点，半径为整数R的圆，其方程

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

- 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

中点Bresenham算法画圆



中点Bresenham画圆的原理

圆的绘制

中点Bresenham算法画圆

构造判别式

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

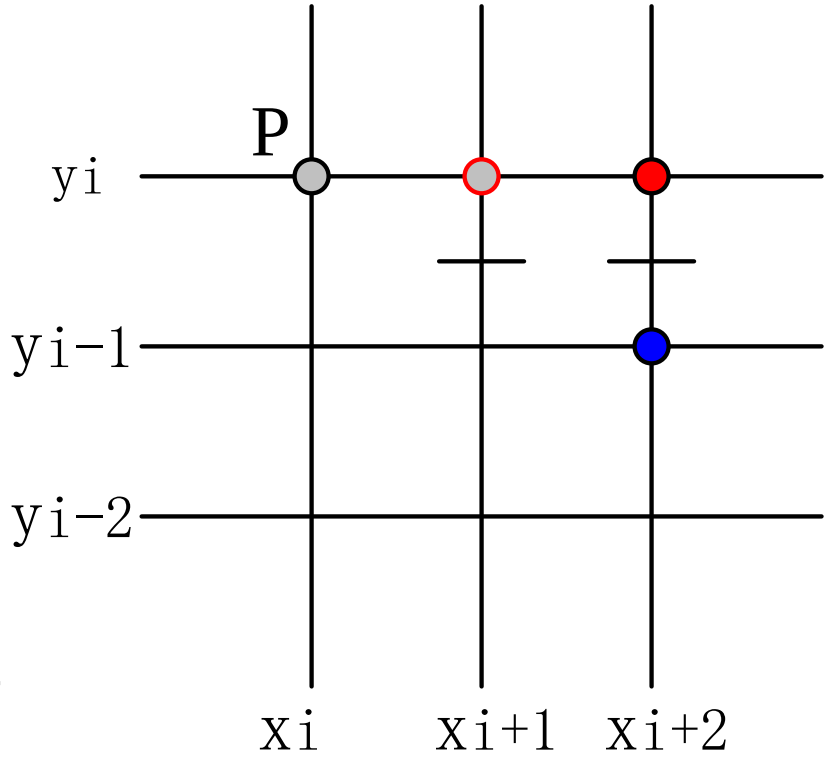
- 当d≤0时，下一点取$P_u(x_i+1, y_i)$；
- 当d>0时，下一点取$P_d(x_i+1, y_i-1)$。

圆的绘制

- 中点Bresenham算法画圆
  - 误差项的递推（d≤0）

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

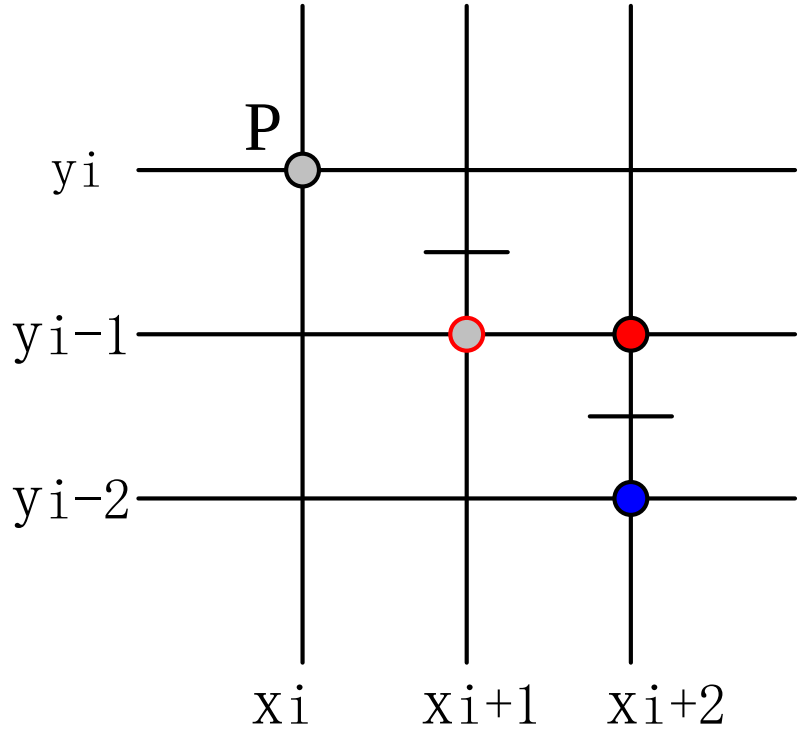
$$d_2 = (x_i + 1 + 1)^2 + (y_i - 0.5)^2 - R^2$$
$$= (x_i + 1)^2 + 2x_i + 3 + (y_i - 0.5)^2 - R^2$$
$$= d_1 + 2x_i + 3$$



圆的绘制

中点Bresenham算法画圆

误差项的递推（d>0）



$$d_2 = F(x_i + 2, y_i - 1.5)$$

$$= (x_i + 2)^2 + (y_i - 1.5)^2 - R^2$$

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

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

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

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

圆的绘制

- 中点Bresenham算法画圆
  - 误差项的初值

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

圆的绘制

- 中点Bresenham算法画圆
  - 用d-0.25代替d

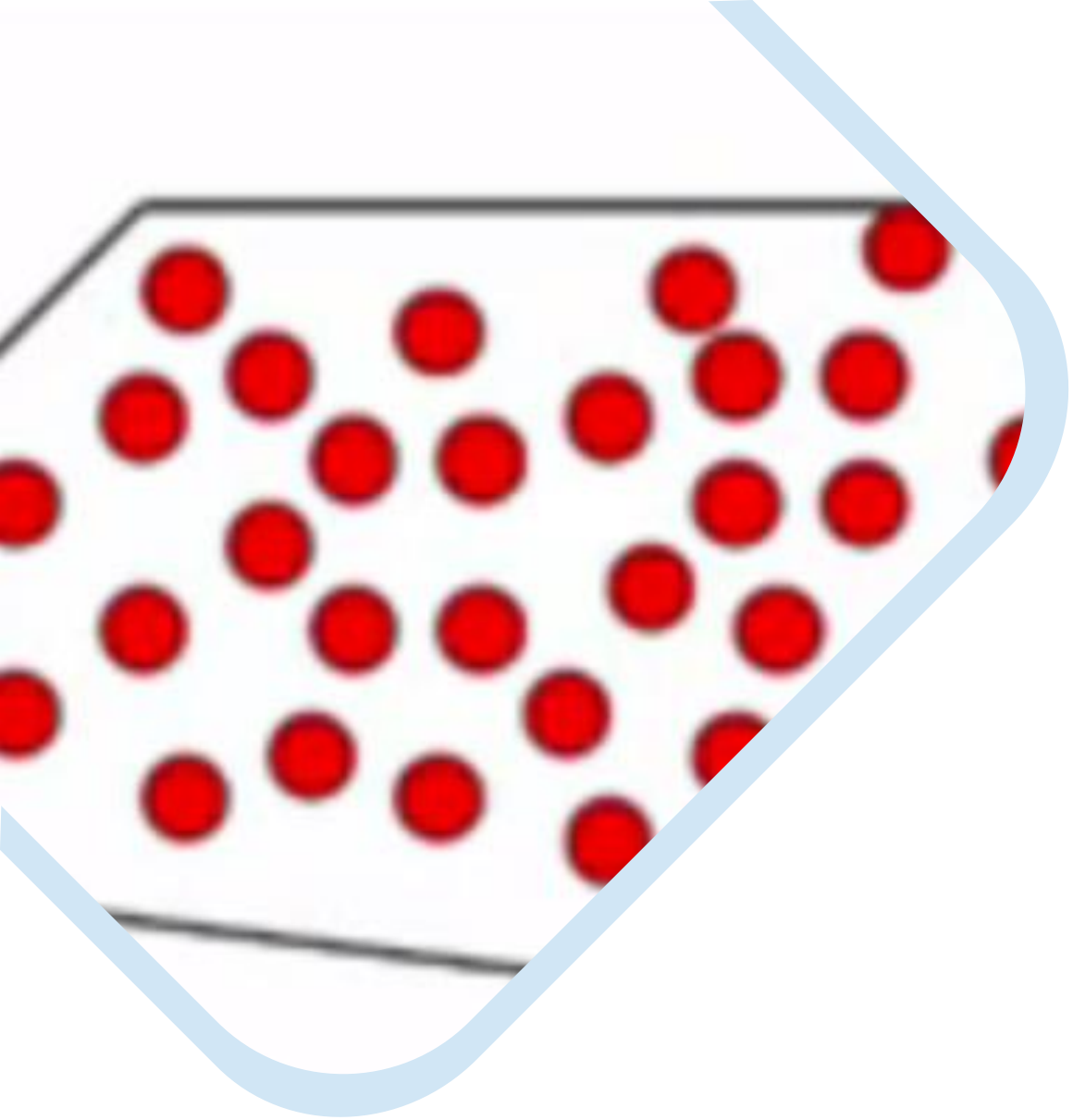
$$d = d + 2x_i + 3 \qquad d \le -0.25$$

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

$$d_0 = 1 - R$$

$$d \le -0.25 \rightarrow d < 0$$

$$d > -0.25 \rightarrow d \ge 0$$



PART 03

多边形的绘制

03 多边形的绘制

- 多边形的扫描转换
- 区域填充

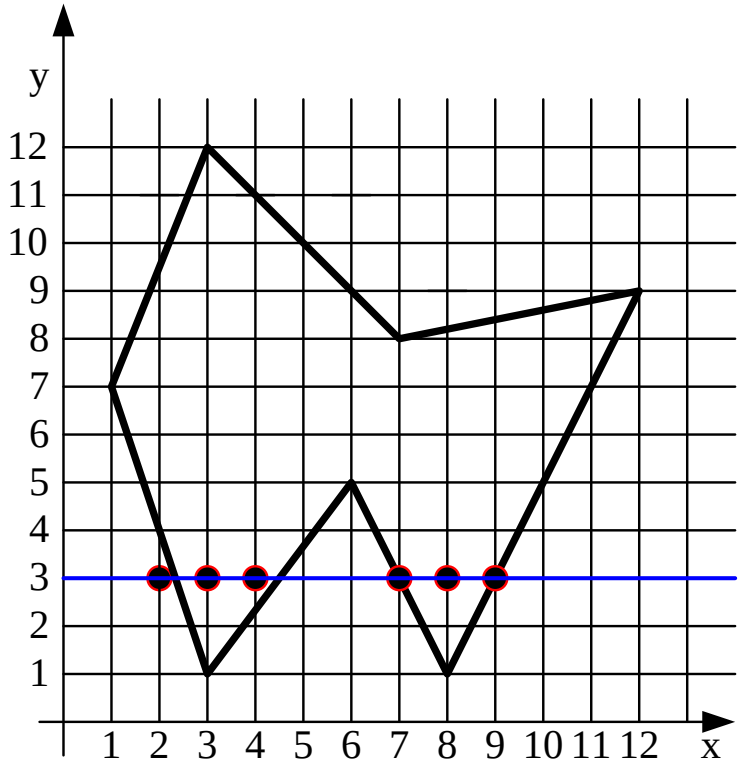
03 多边形的扫描转换

- 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象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象素，并将其颜色值写入帧缓存中相应单元的过程

03 多边形的扫描转换

X-扫描线算法

- 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象素



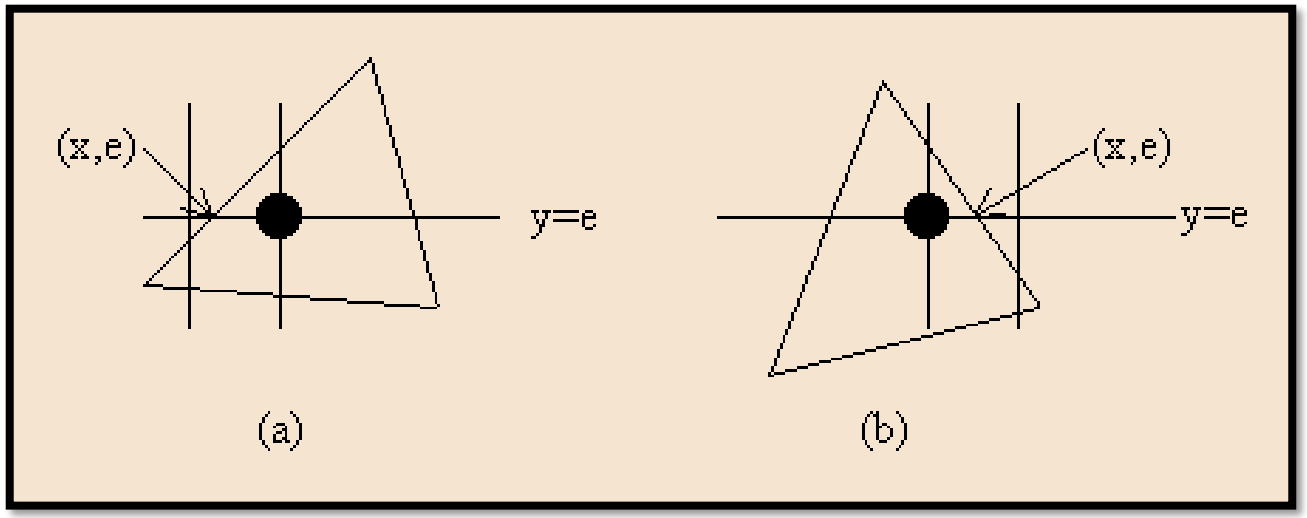
03 多边形的扫描转换

- X-扫描线算法
  - 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（ymin和ymax）
  - 从y=ymin到y=ymax，每次用一条扫描线进行填充
  - 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

03 多边形的扫描转换

X-扫描线算法

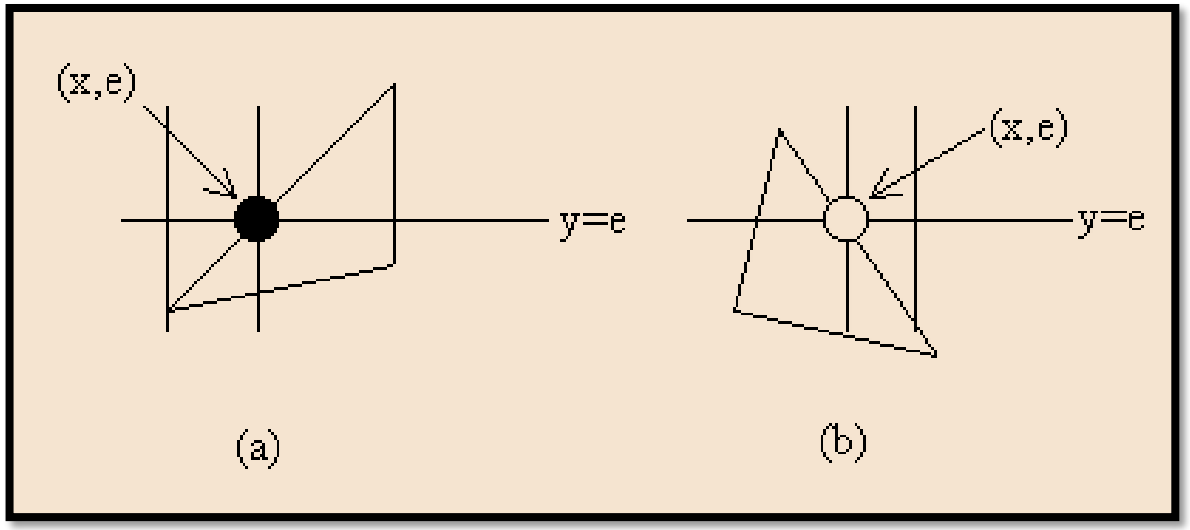
- 取整规则：假定非水平边与扫描线y=e相交，交点的横坐标为x
  - 规则1：x为小数，即交点落于扫描线上两个相邻像素之间时：交点位于左边界之上，向右取整；交点位于右边界之上，向左取整



多边形的扫描转换

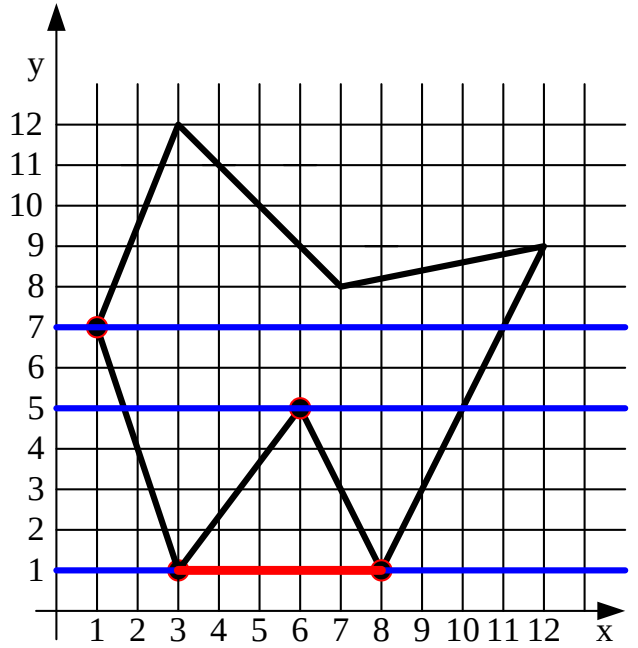
X-扫描线算法

- 取整规则：假定非水平边与扫描线y=e相交，交点的横坐标为x
  - 规则2：边界上象素的取舍问题，避免填充扩大化。规定落在右边边界上的象素不予填充。（具体实现时，只要对扫描线与多边形的相交区间左闭右开）



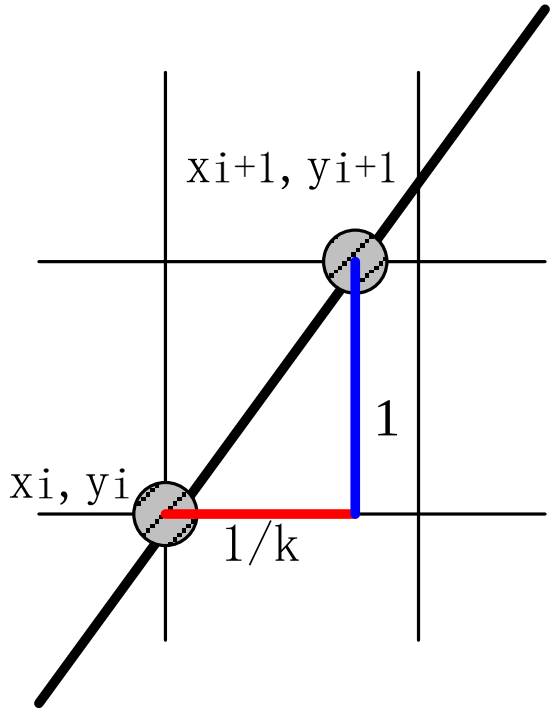
03 多边形的扫描转换

- X-扫描线算法
  - 取整规则：假定非水平边与扫描线y=e相交，交点的横坐标为x
    - 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



03 多边形的扫描转换

- 改进的有效边表算法
  - 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



03 多边形的扫描转换

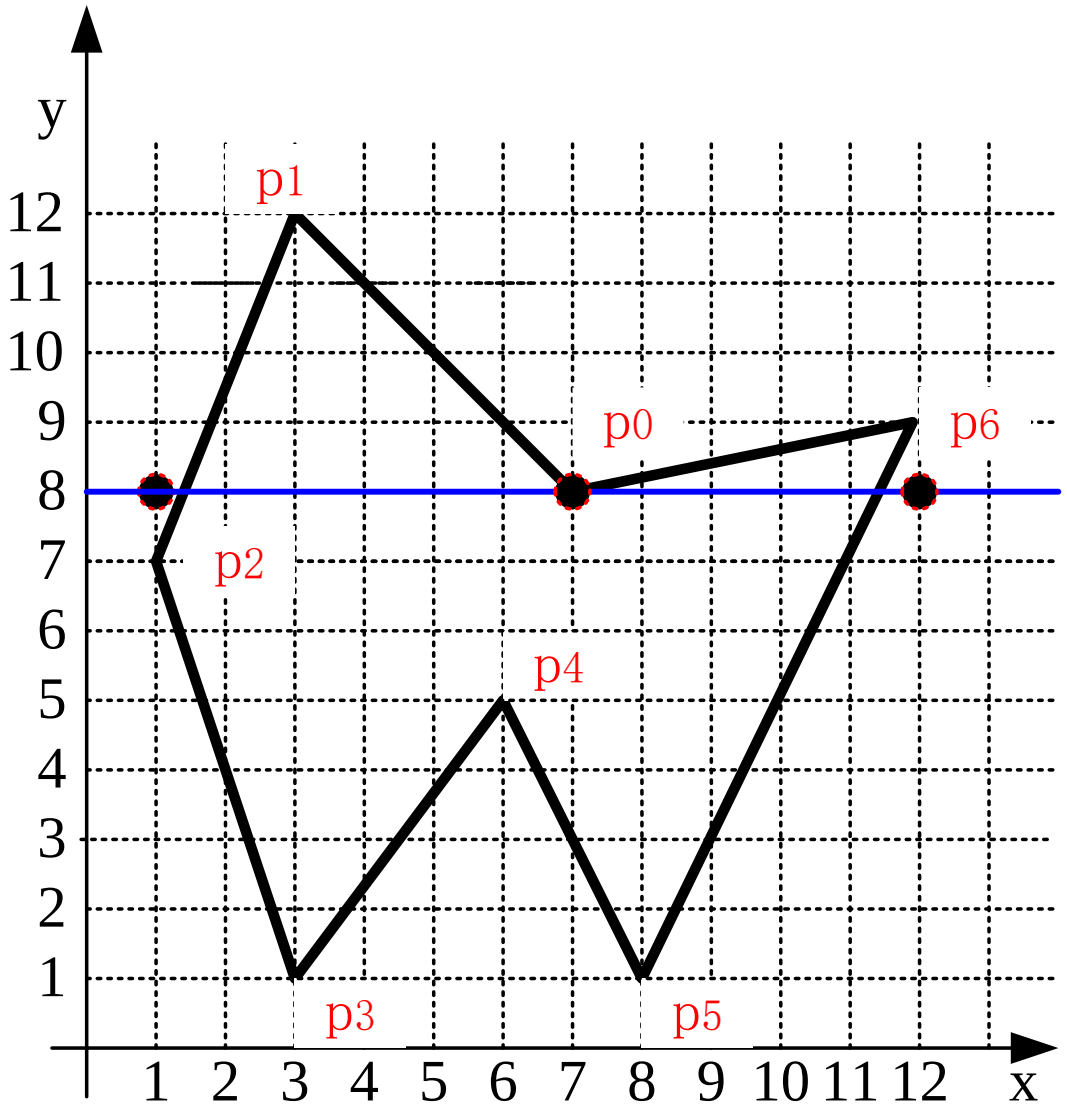
- 改进的有效边表算法
  - 有效边（Active Edge）：指与当前扫描线相交的多边形的边，也称为活性边
  - 有效边表（Active Edge Table, AET）：把有效边按与扫描线交点x坐标递增的顺序存放在一个链表中，此链表称为有效边表
  - 有效边表的每个结点：

x ymax 1/k next

多边形的扫描转换

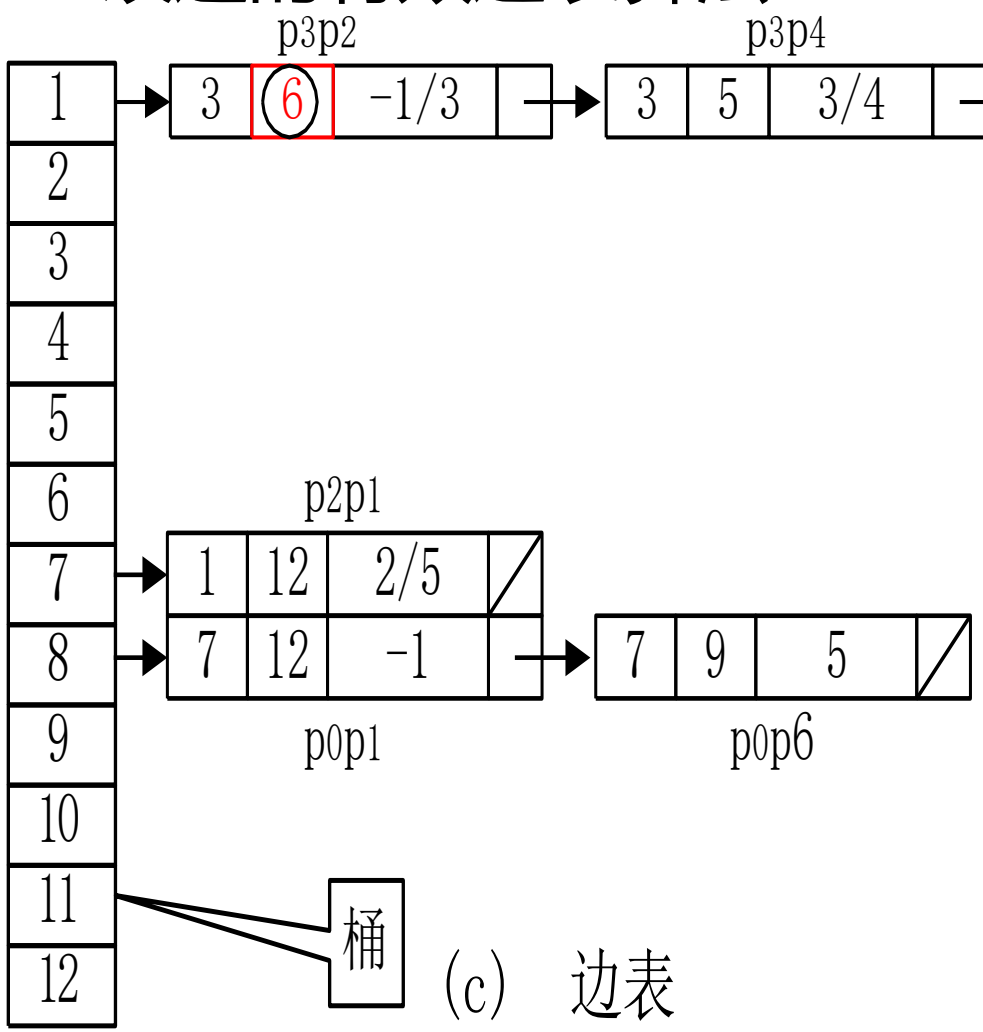
改进的有效边表算法

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	

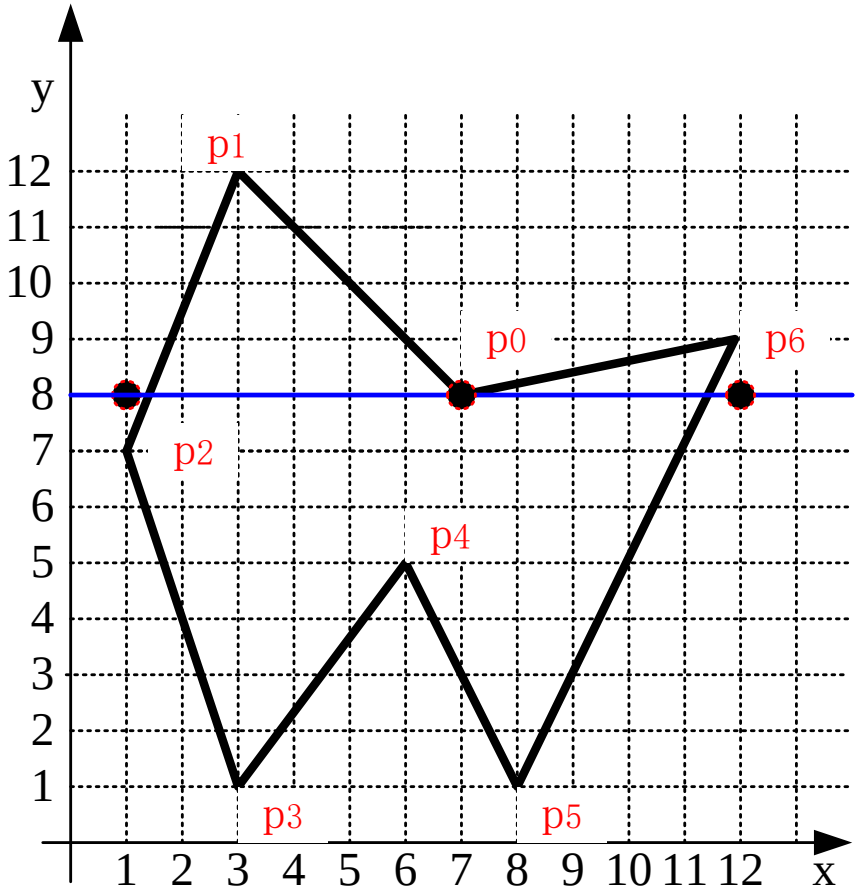


多边形的扫描转换

改进的有效边表算法



(c) 边表



03 多边形的扫描转换

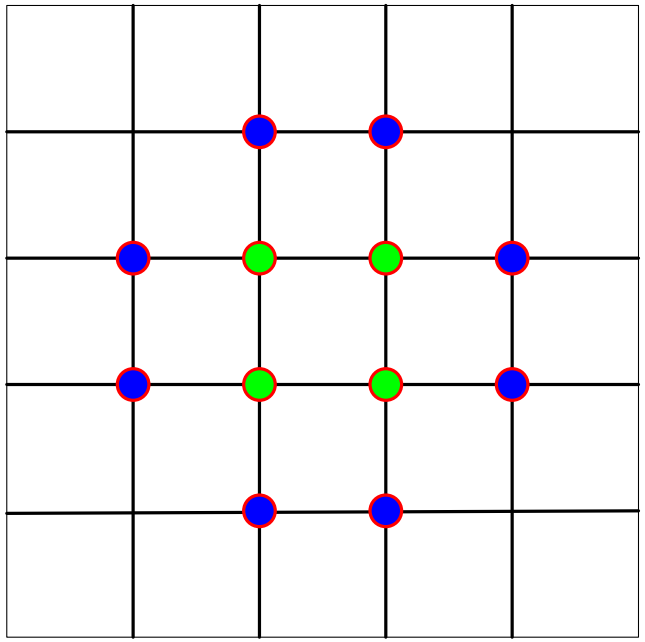
- 边缘填充算法
  - 按任意顺序处理多边形的每条边。处理时，先求出该边与扫描线的交点，再对扫描线上交点右方的所有象素取反
- 栅栏填充算法
  - 按任意顺序处理多边形的每一条边，但处理每条边与扫描线的交点时，将交点与栅栏之间的象素取反

03 多边形的扫描转换

- 边标志算法

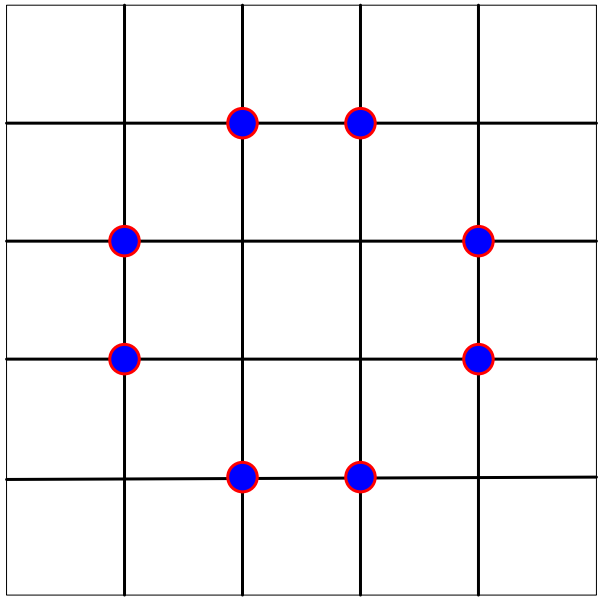
03 区域填充

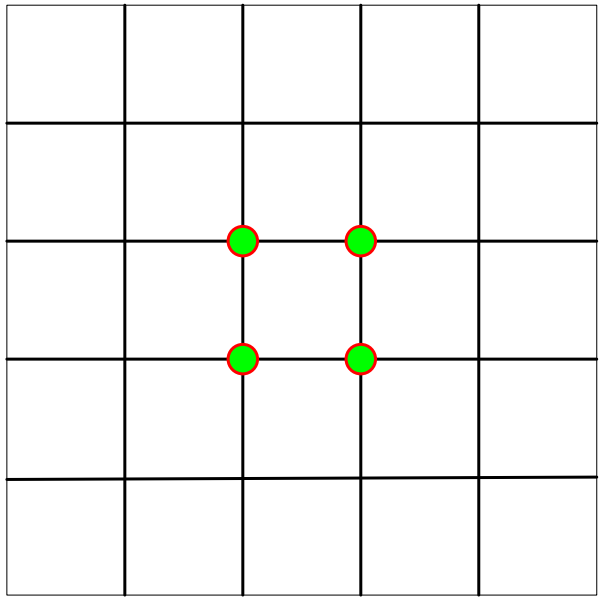
- 区域填充是指从区域内的某一个象素点（种子点）开始，由内向外将填充色扩展到整个区域内的过程
- 区域是指已经表示成点阵形式的填充图形，它是相互连通的一组像素的集合



03 区域填充

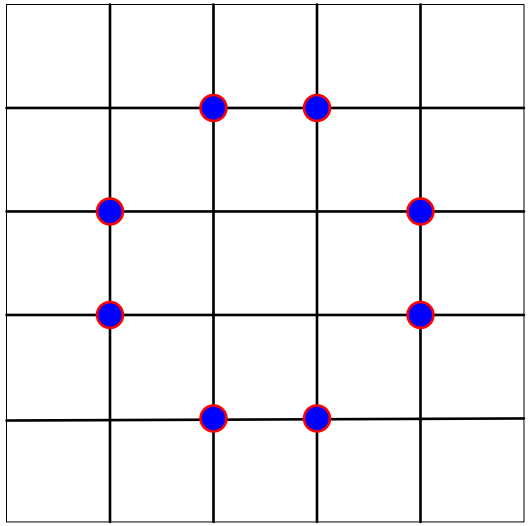
- 边界表示法：把位于给定区域的边界上的象素一一列举出来的方法
- 内点表示：枚举出给定区域内所有象素的表示方法

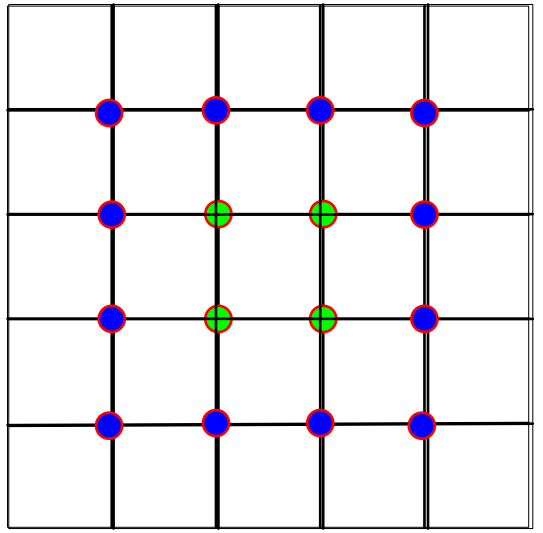




03 区域填充

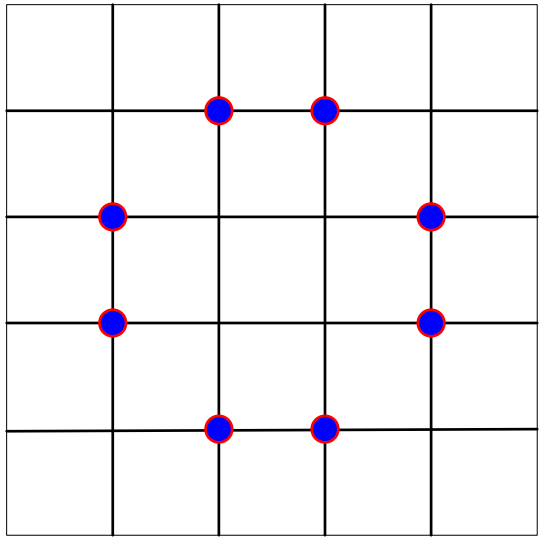
- 4-连通区域：从区域上的一点出发，通过访问已知点的4-邻接点，在不越出区域的前提下，遍历区域内的所有象素点
- 8-连通区域：从区域上的一点出发，通过访问已知点的8-邻接点，在不越出区域的前提下，遍历区域内的所有象素点





03 区域填充

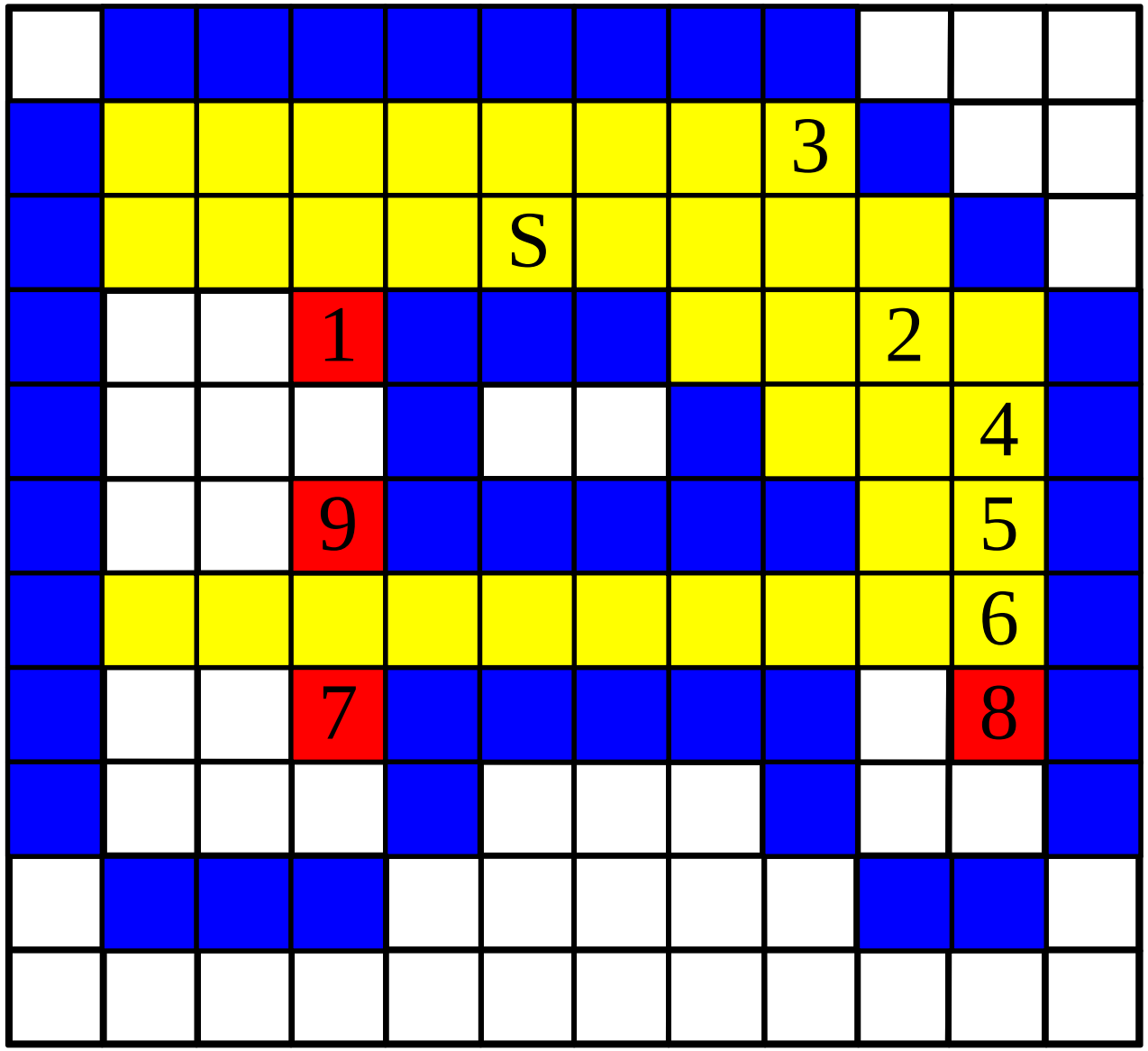
- 边界填充算法（Boundary-fill Algorithm）
  - 算法的输入：种子点坐标(x,y)，填充色以及边界颜色
  - 利用堆栈实现简单的种子填充算法
    - 算法从种子点开始检测相邻位置是否是边界颜色，若不是就用填充色着色，并检测该象素点的相邻位置，直到检测完区域边界颜色范围内的所有象素为止



03 区域填充

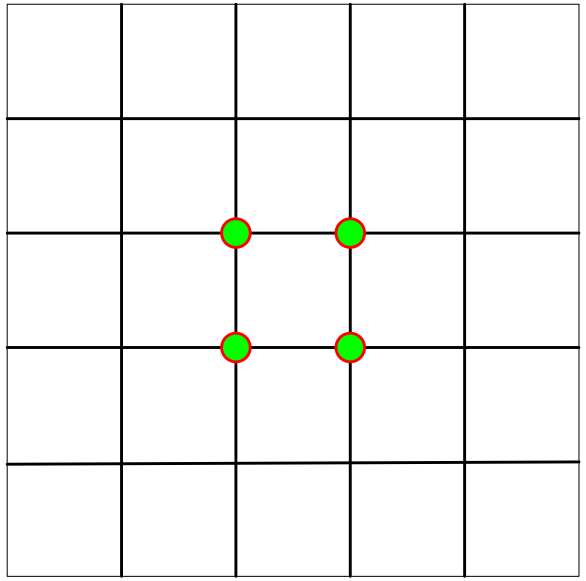
- 边界填充算法（Boundary-fill Algorithm）
  - 算法的输入：种子点坐标(x,y)，填充色以及边界颜色
  - 扫描线种子填充算法
    - 当给定种子点时，首先填充种子点所在的扫描线上的位于给定区域的一个区段，然后确定与这一区段相通的上下两条扫描线上位于给定区域内的区段，并依次保存下来。反复这个过程，直到填充结束

03 区域填充



03 区域填充

- 泛填充算法（Flood-fill Algorithm）
  - 算法的输入：种子点坐标(x,y)，填充色和内部点的颜色
  - 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象素点



感谢您的观看