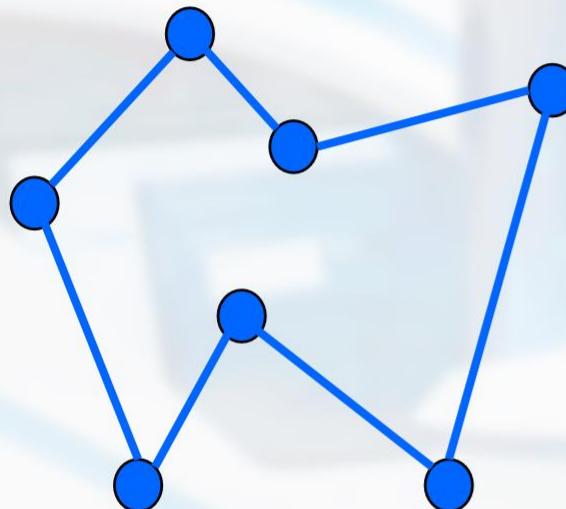


多边形扫描转换算法

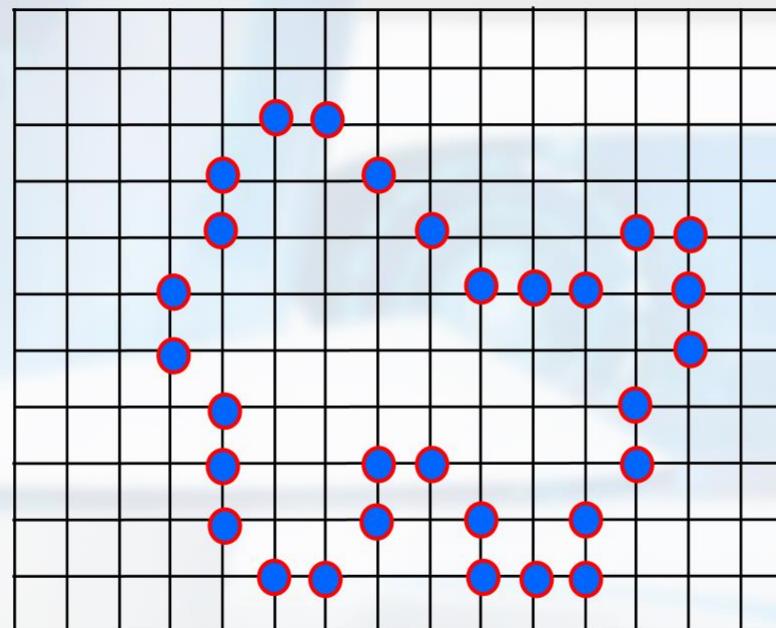
- 多边形表示
- 扫描转换算法
 - X-扫描线算法
 - Y-向连贯性算法
 - 边标志算法
- 区域填充算法

1. 多边形表示

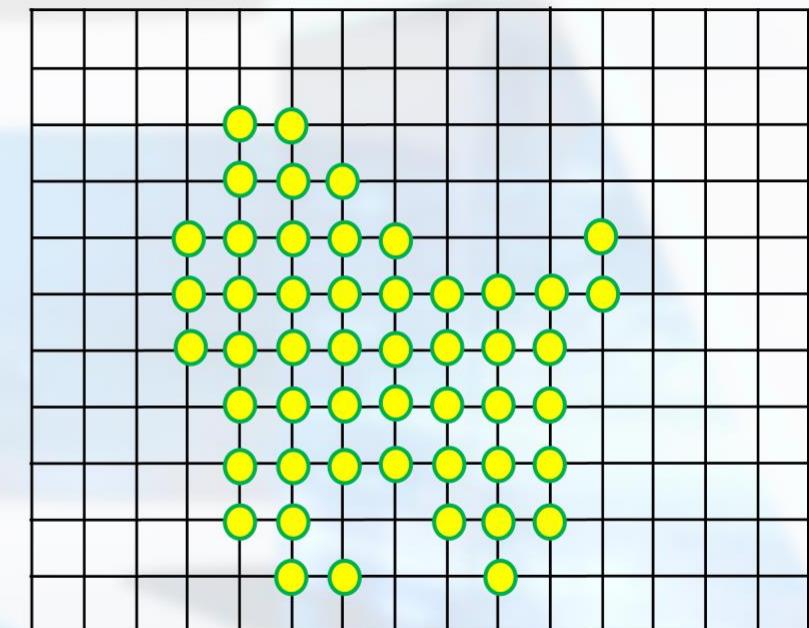
“区域”的几种表示方法：



顶点表示



边界表示

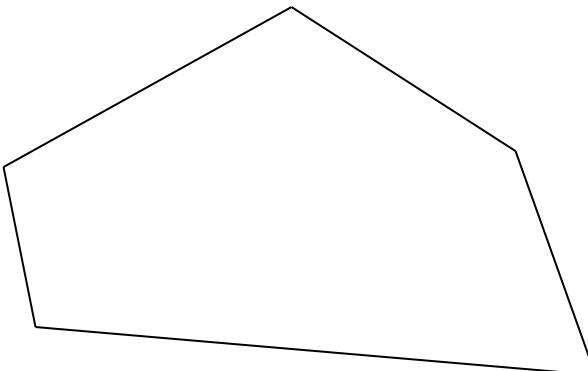


内点表示

Drawing simple convex polygons

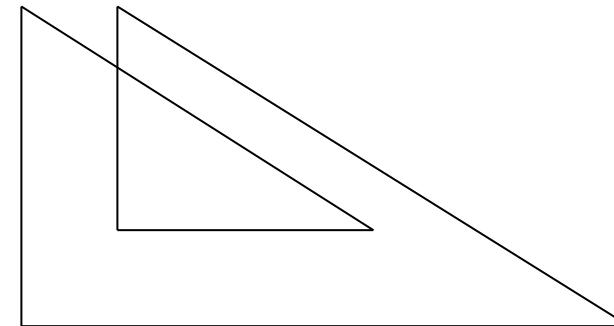
openGL only fill (simple and convex polygon简单凸多边形)

- 简单多边形: A polygon is **simple** if no edges cross each other
- 凸多边形: A polygon is **convex** if any line segment connecting two points in the polygon lies inside the polygon.

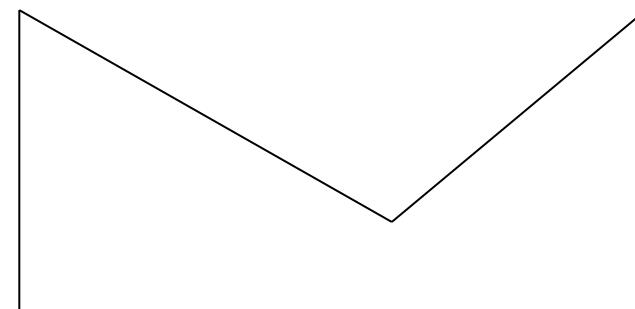


A simple convex polygon

Show polygon express



A *non-simple* polygon



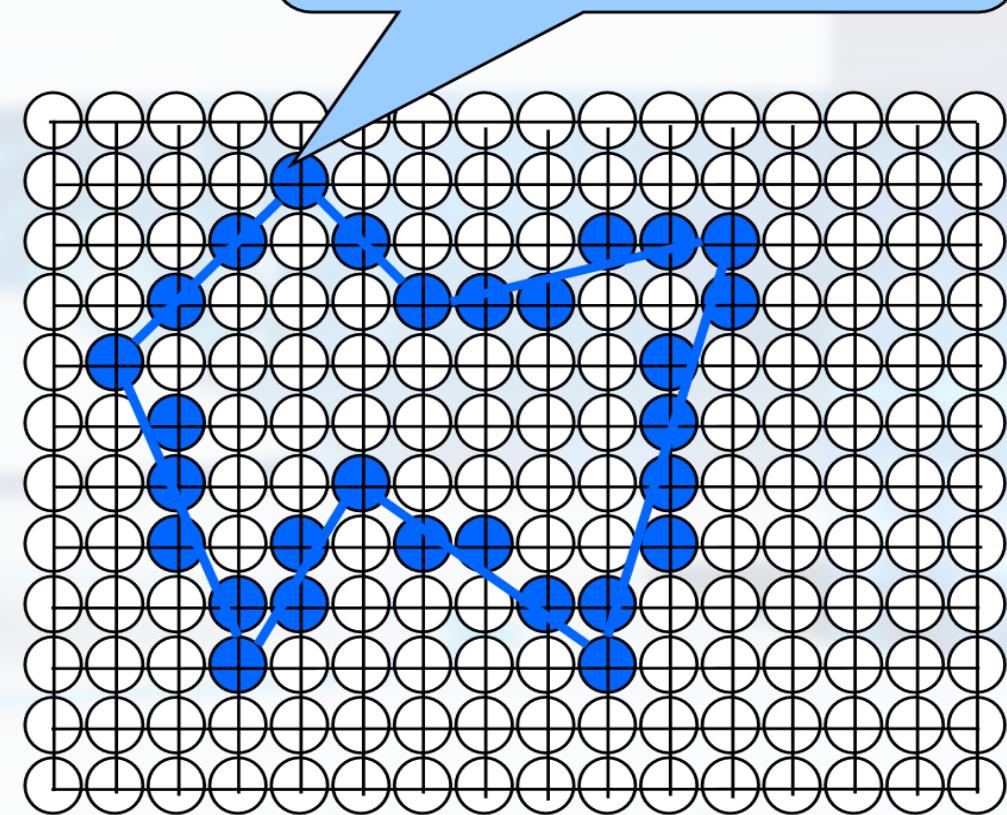
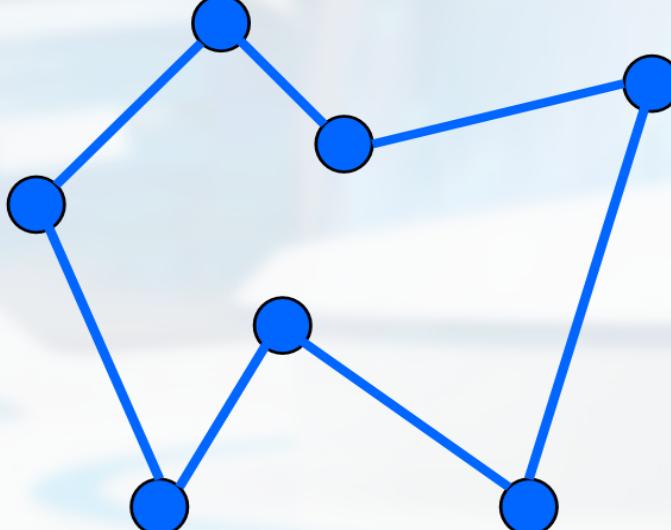
A concave polygon/non-convex polygon

多边形扫描转换算法

输入：多边形顶点序列 $P_1(x_1, y_1)$ 到 $P_7(x_7, y_7)$

输出：最佳逼近这个多边形的像素点集

**并不是仅仅需要边界
而是需要整个多边形区域**



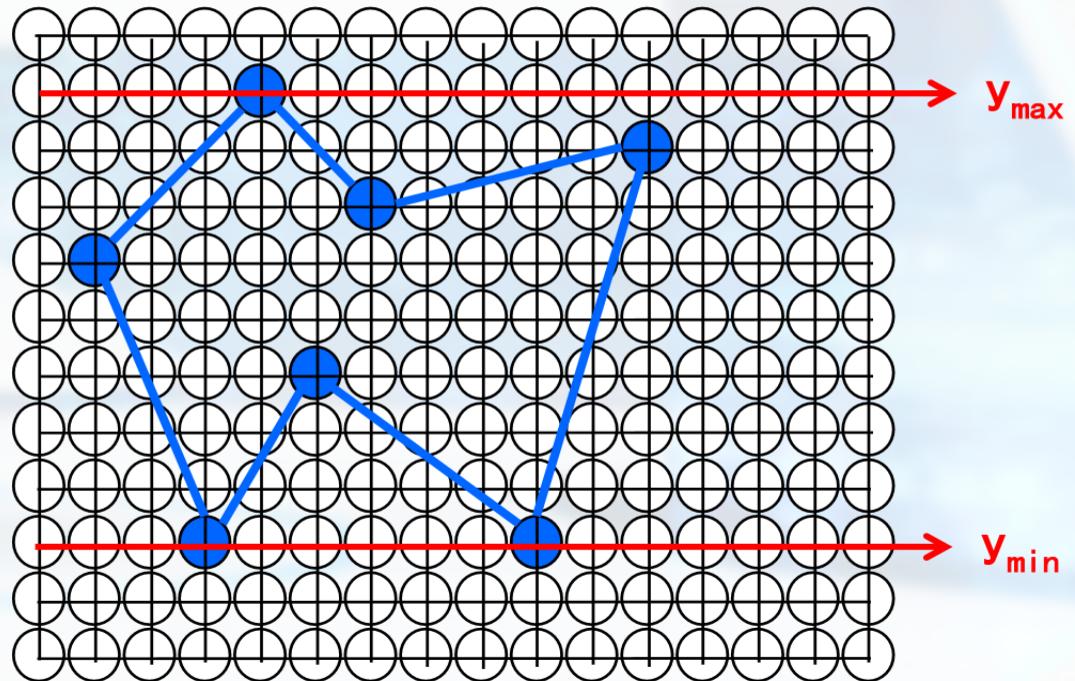
多边形扫描转换算法

- 多边形表示
- 扫描转换算法
 - X-扫描线算法
 - Y-向连贯性算法
 - 边标志算法
- 区域填充算法

1) X-扫描线算法

算法步骤：

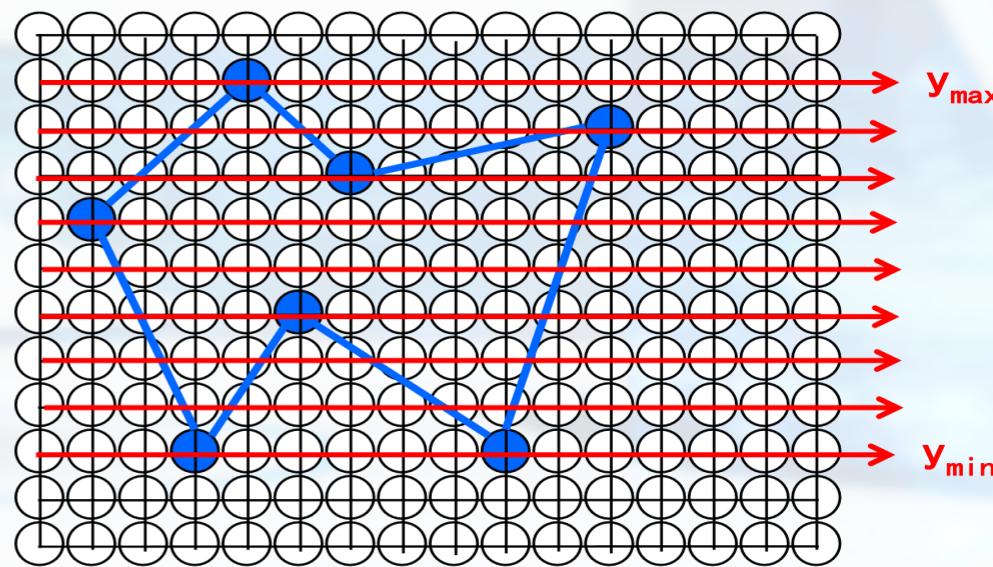
- (1)确定多边形所占有的最大扫描线数，
得到多边形顶点的最小和最大y值
(y_{min} 和 y_{max})。
- (2)从 $y=y_{min}$ 到 $y=y_{max}$ ，每次用一条扫描
线进行填充。
- (3)对一条扫描线填充的过程可分为四个
步骤：
 - a.求交
 - b.排序
 - c.交点配对
 - d.区间填色



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：
 - a. 求交
 - b. 排序
 - c. 交点配对
 - d. 区间填色

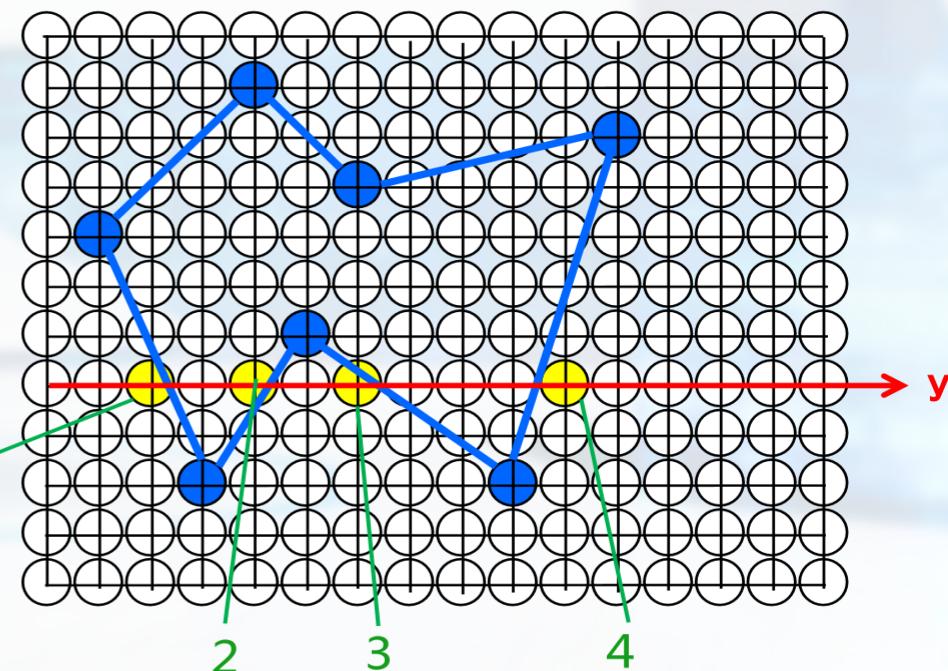


X-扫描线算法

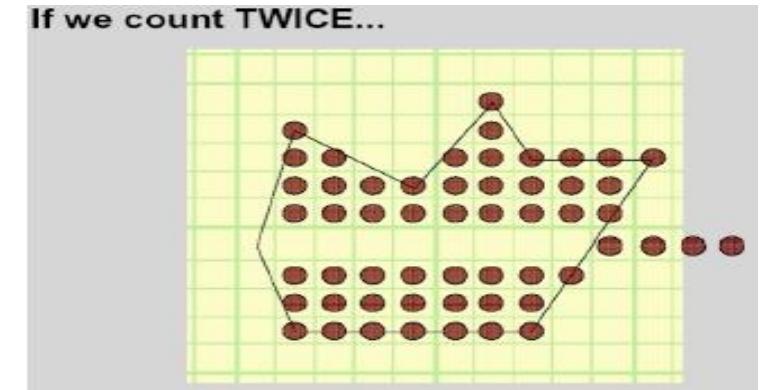
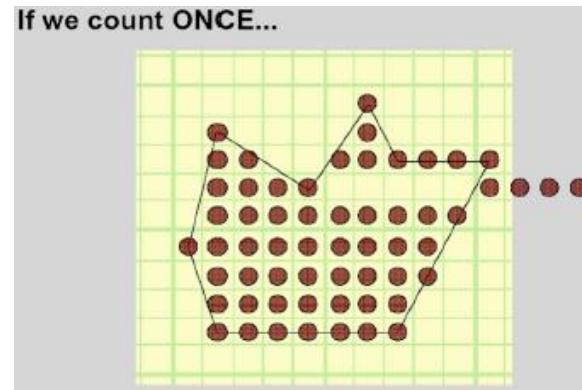
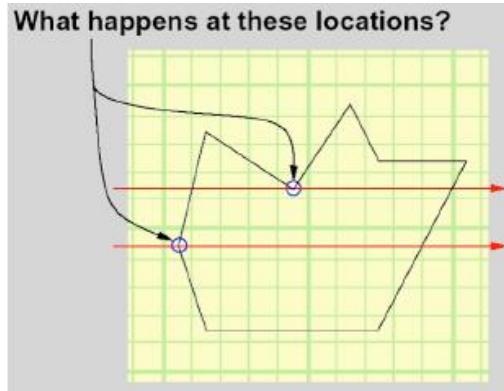
算法步骤：

- (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：
 - a.求交
 - b.排序
 - c.交点配对
 - d.区间填色

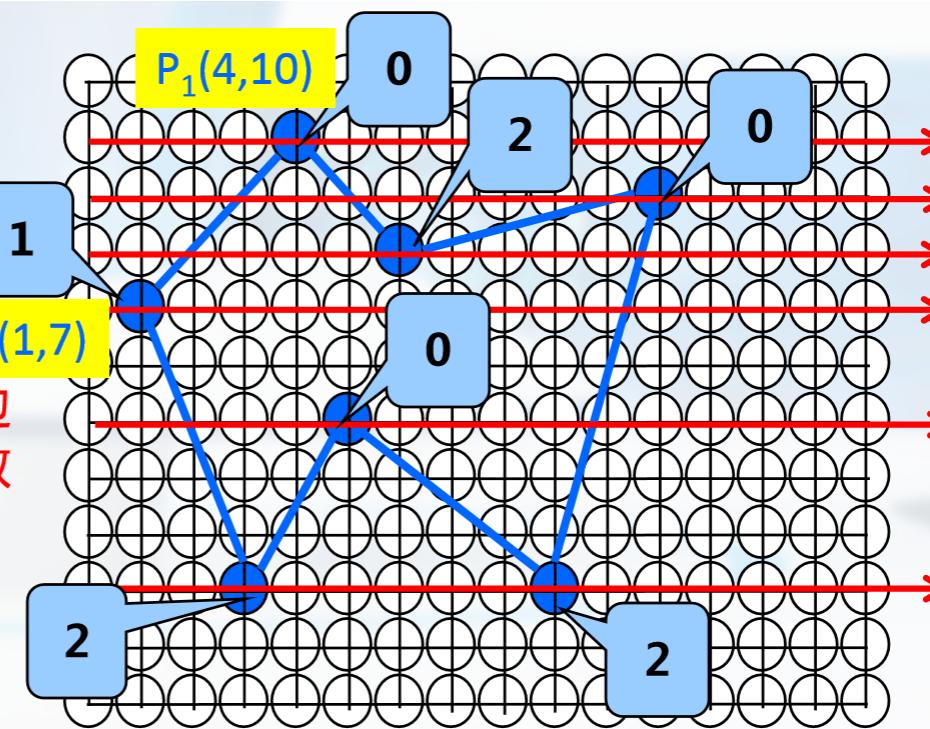


问题讨论1：Number of intersections ?



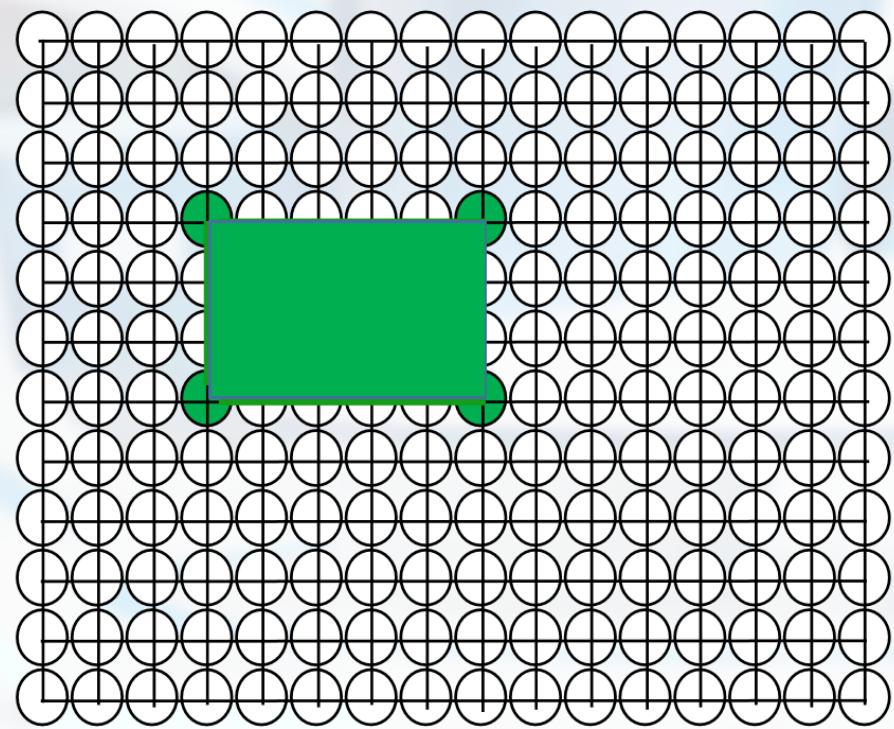
策略：

交点个数 =
 $P_7(1,7)$
构成这个顶点的两条边
位于扫描线上方的条数

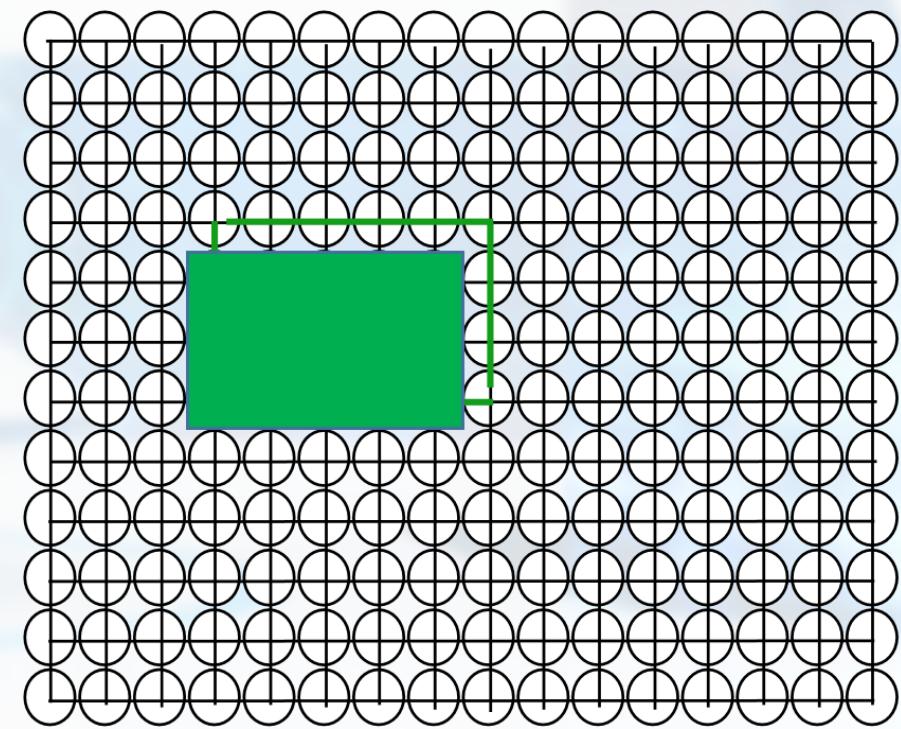


问题讨论2：区间填充原则

(2) 左闭右开，下闭上开。



形状保持
中心偏移
半个像素
→

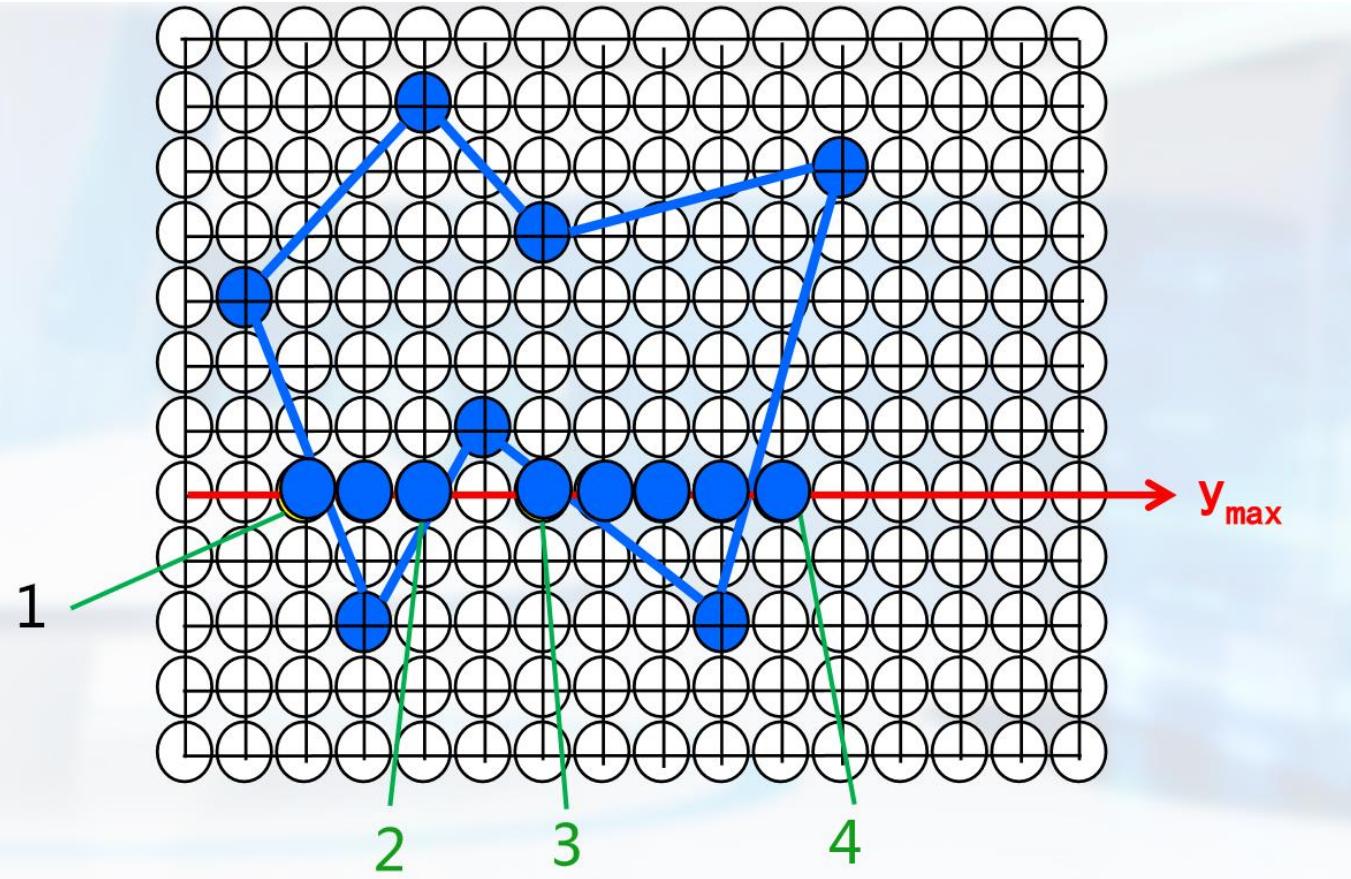


问题讨论2：算法效率

(3) 算法的效率问题。

对一条扫描线填充的过程
可分为四个步骤：

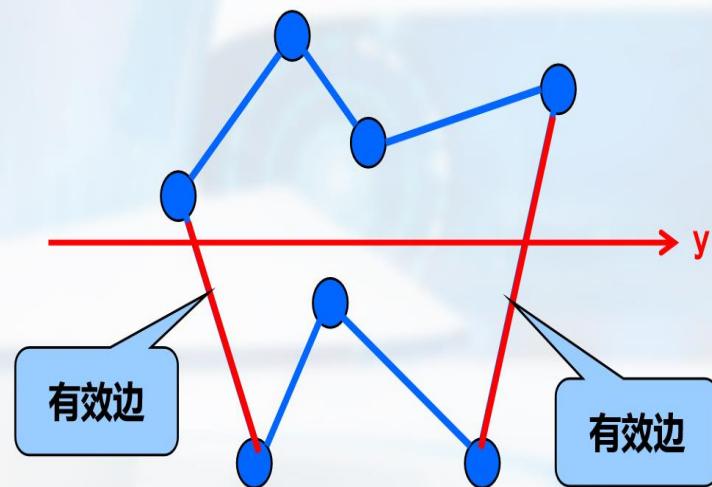
- a.求交
- b.排序
- c.交点配对
- d.区间填色



“求交”优化

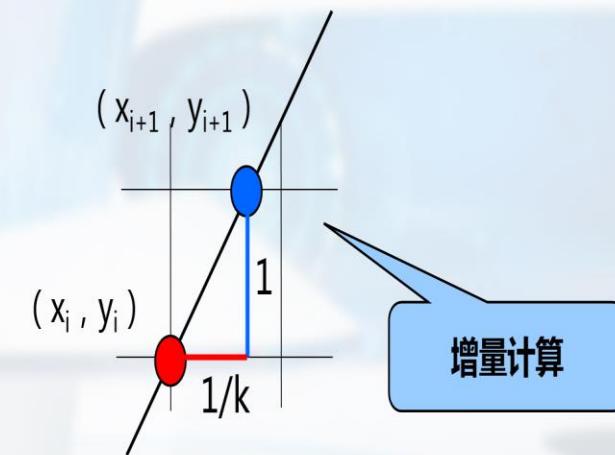
(1)对于某一条扫描线，需要与所有的边求交吗？

- a.求交
- b.排序
- c.交点配对
- d.区间填色



(2)扫描线和直线在Y方向上都有连贯性，那么交点可以怎么求？

- a.求交
- b.排序
- c.交点配对
- d.区间填色



(1)只对有交的边求交

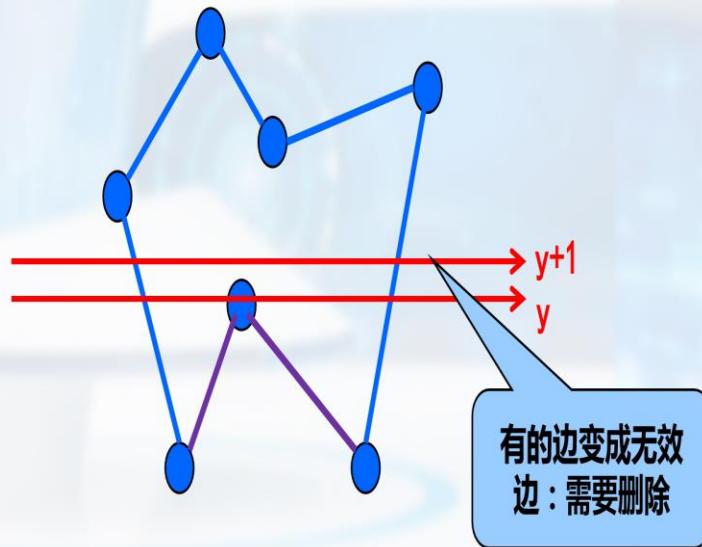
(2)利用扫描线和线段的连贯性，增量计算交点位置

$$y = y + 1; \\ x = x + 1/k;$$

排序优化

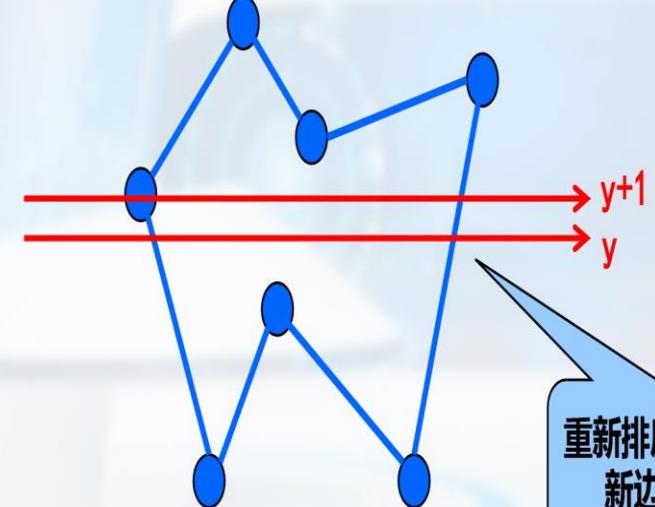
(3)每次都需要排序吗？

- a.求交
- b.排序
- c.交点配对
- d.区间填色



(3)每次都需要排序吗？

- a.求交
- b.排序
- c.交点配对
- d.区间填色



重新排序的时机：
新边加入时

2) 有效边表算法 (Y向连贯性算法)

- 先创建ET表 (新边表)
- 然后创建按动态变化的AET表 (活性边表)
- AET表

算法描述

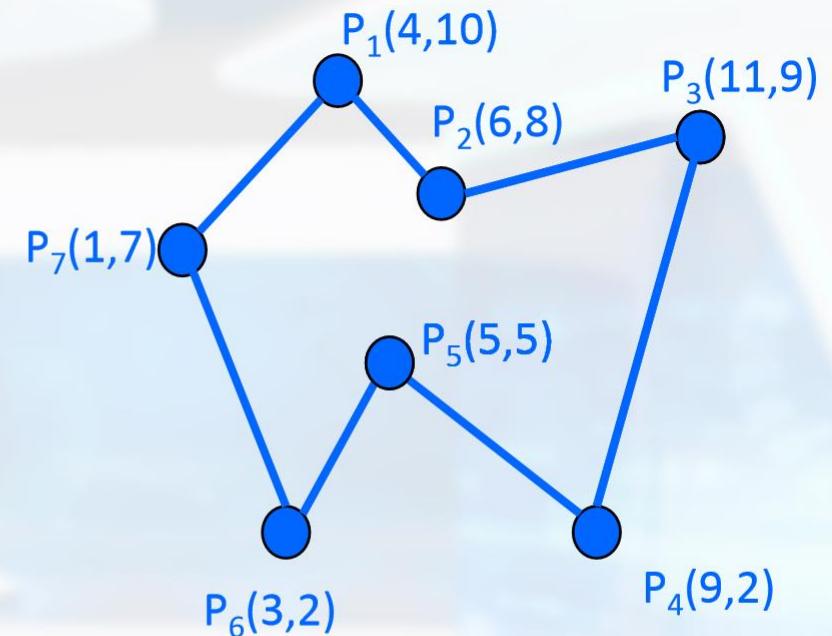
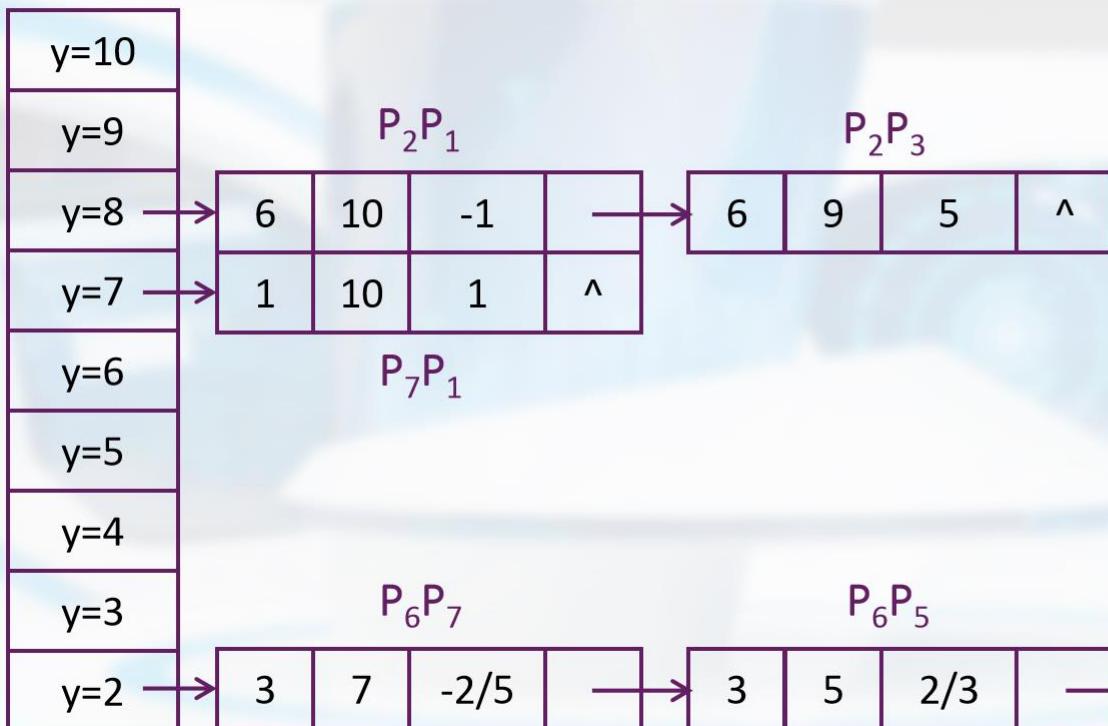
```
void polyfill (polygon, color)
{ //建立边表ET
    for (各条扫描线) 初始化边表头指针ET [i];
    for (各条边) 把 $y_{min} = i$  的边表示成边结点链接到边表ET [i]中;
    //建立动态表AET
    初始化活性边表AET为空;
    for (i=ymin;i++;i<=ymax)      //i为当前扫描线
    {
        1) 遍历AET表，把 $y_{max} = i$  的边结点从AET表中删除;
           把 $y_{max} > i$ 的边结点的x值变为 $X+1/k$ 后插入到AET链表中;
           若边表ET[i]中有结点，则把边结点添加到AET表. (求交)
        2) 用冒泡等排序法等对AET表排序 (边结点按x坐标递增顺序) 若X相同则按1/k由小到大排 (排序)
        3) 遍历AET表，配对交点区间内的象素并填充 (配对, 填充)
    }
} /* polyfill */
```

数据结构-边表ET

每条新边对应一个结点：

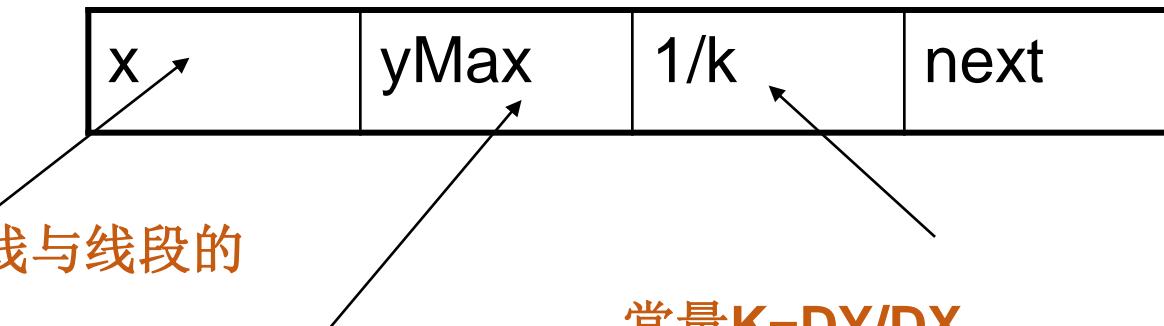
x	y_{max}	$1/k$	next
---	-----------	-------	------

边表 (Edge Table) :



数据结构-边结构

Edge Node:



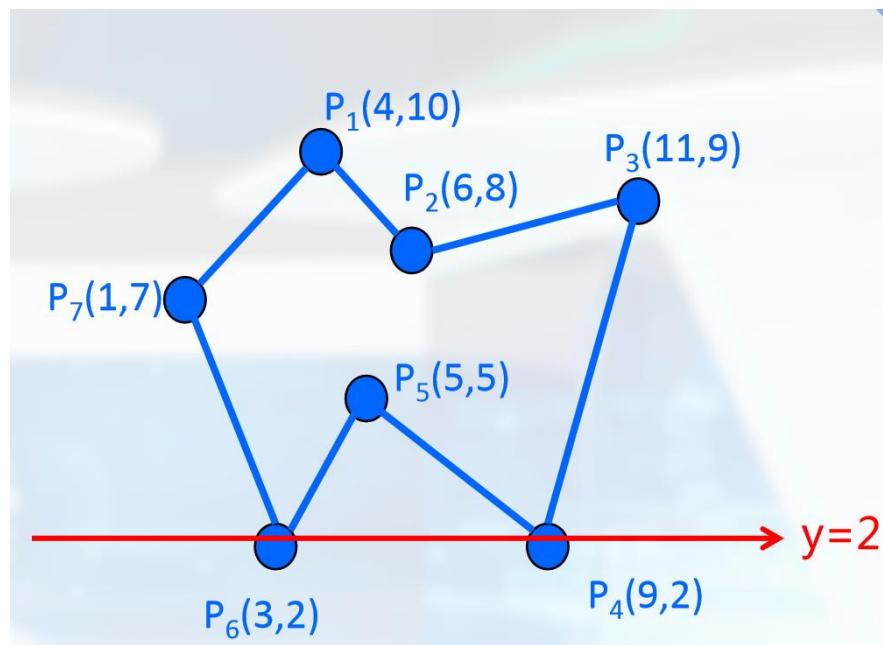
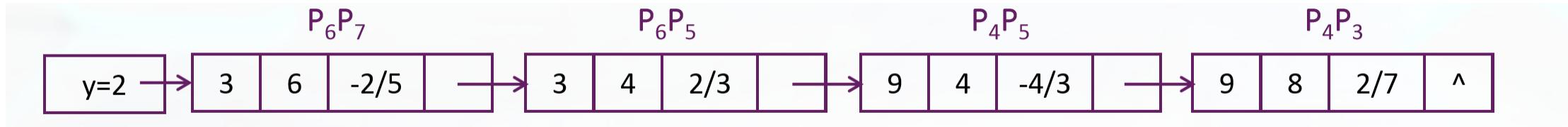
存放扫描线与线段的
交点X

Y_{max} ---该边最大Y值，
扫描线 $< Y_{max}$ 时有交点

常量 $K = DY/DX$
用以增量计算下一交点 $X = X + 1/k$

数据结构-AET

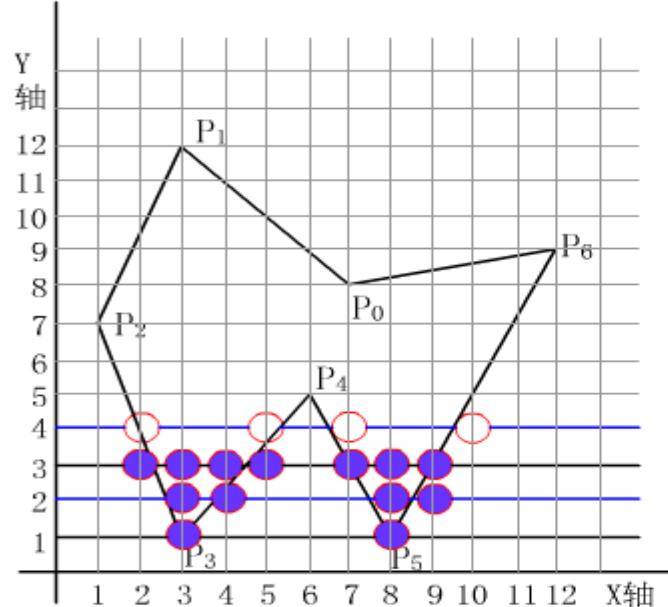
- AET Active Edge Table (活性边表)
 - 与扫描线相交的边节点构成的链表



Example (演示)

- 多边形顶点序列为
 $P_0(7, 8), P_1(3, 12), P_2(1, 7), P_3(3, 1), P_4(6, 5), P_5(8, 1), P_6(12, 9)$

*** 有效边表算法 ***



AET表

$y=4$	—	P_2P_3	—	P_3P_4	—	P_4P_5	—	P_5P_6	—
	—	$2 6 -1/3 $	—	$5 3 5 3/4 $	—	$6 5 5 -1/2 $	—	$9 5 9 1/2 $	—

1	3	6	-1/3	—	3	5	3/4	—
:	8	5	-1/2	—	8	9	1/2	—
7	1	12	2/5	—				
8	7	12	-1	—	7	9	5	—
:	x	y _{min}	y _{max}	1/k	next			
12								

边表

算法讨论

1) 交点是顶点时，算作几个交点？

答：

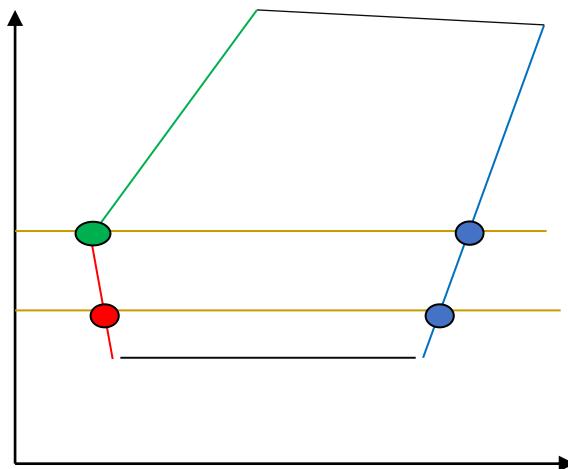
按算法，扫描线与某线段顶点相交时其实就是某活性边到顶，或者某新边将加入AET称为活性边。

所以，若是活性边的结束顶点，则不计为交点；若是活性边的开始顶点，需要计为交点；

算法讨论

2) 求交后为什么总要排序?

- 有新的活性边加入活性边表时，可能出现顺序不正确，需要排序。

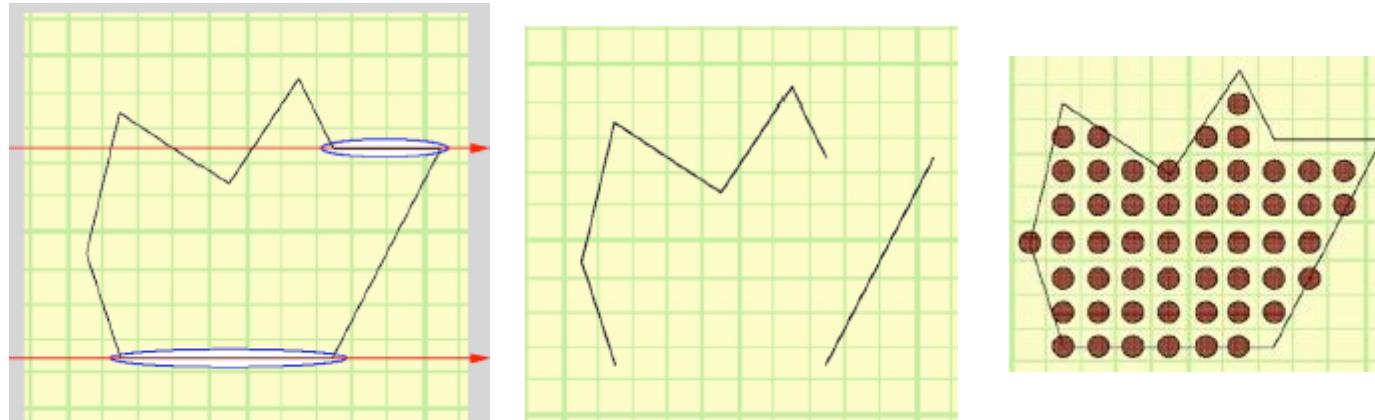


算法讨论

3) 某条边与X坐标轴平行，如何处理？

该边记入ET表吗？如果记入需要加入AET表吗，能否保证交点的奇偶配对正确填充吗？

结论：斜率的倒数不存在， $DY=0$ ，应该不考虑其与扫描线有交点，即不记入ET表。



Deal with horizontal edges

活性边表算法分析

算法分析：

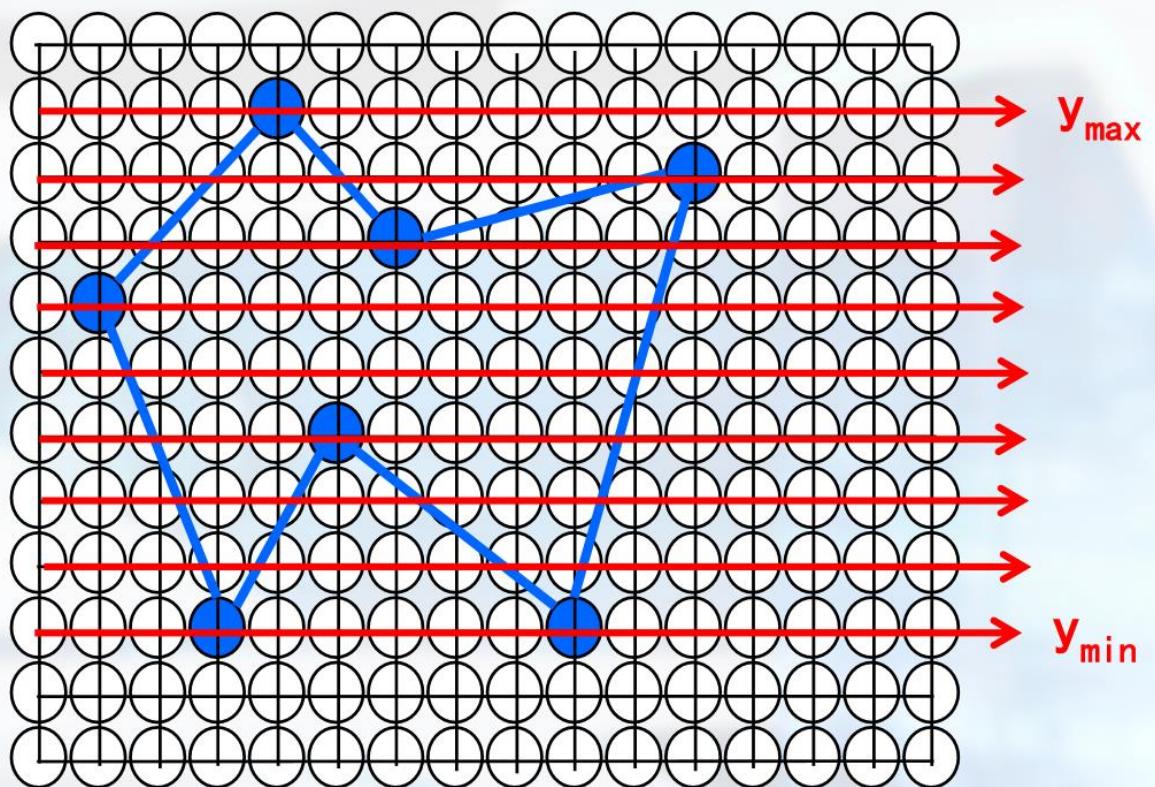
优点：

- ◆ 采用增量计算的方法进行交点计算
- ◆ 仅仅在新边加入时排序

(边数 << 扫描线数)

缺点：

桶表、链表的维护开销



多边形扫描转换算法

- 多边形表示
- 扫描转换算法
 - X-扫描线算法
 - Y-向连贯性算法
 - 边缘填充算法, 栅栏填充算法, 边标志算法
- 区域填充算法

边缘填充算法（演示）

- 优点：算法简单巧妙
- 缺点：访问像素点次数很多，虽然限定了外接矩形。

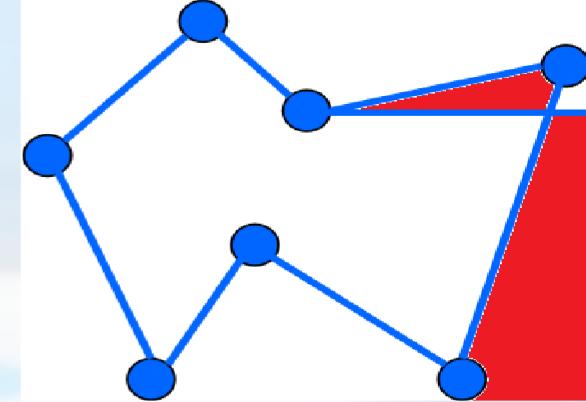
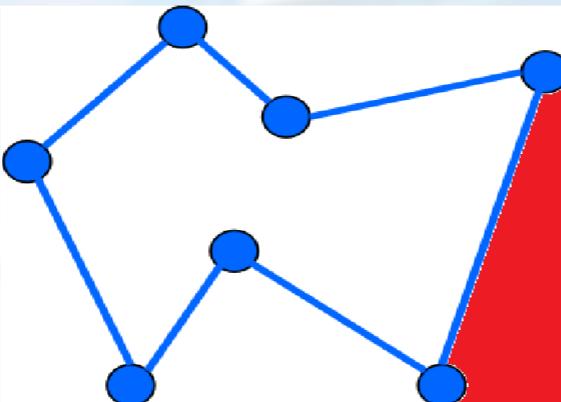
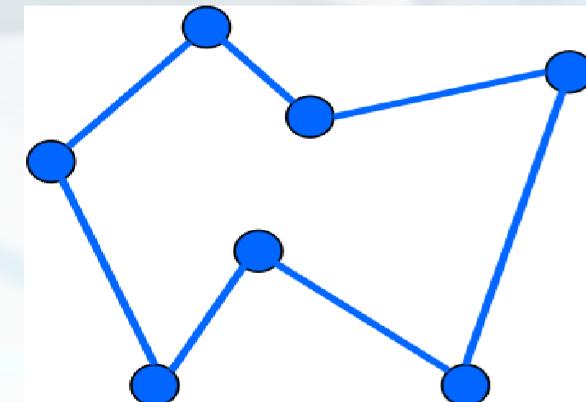
1

奇妙的想法

输入：多边形顶点序列 $P_1(x_1, y_1)$ 到 $P_7(x_7, y_7)$

输出：最佳逼近这个多边形的像素点集

逐边向右取反



栅栏填充算法 (演示)

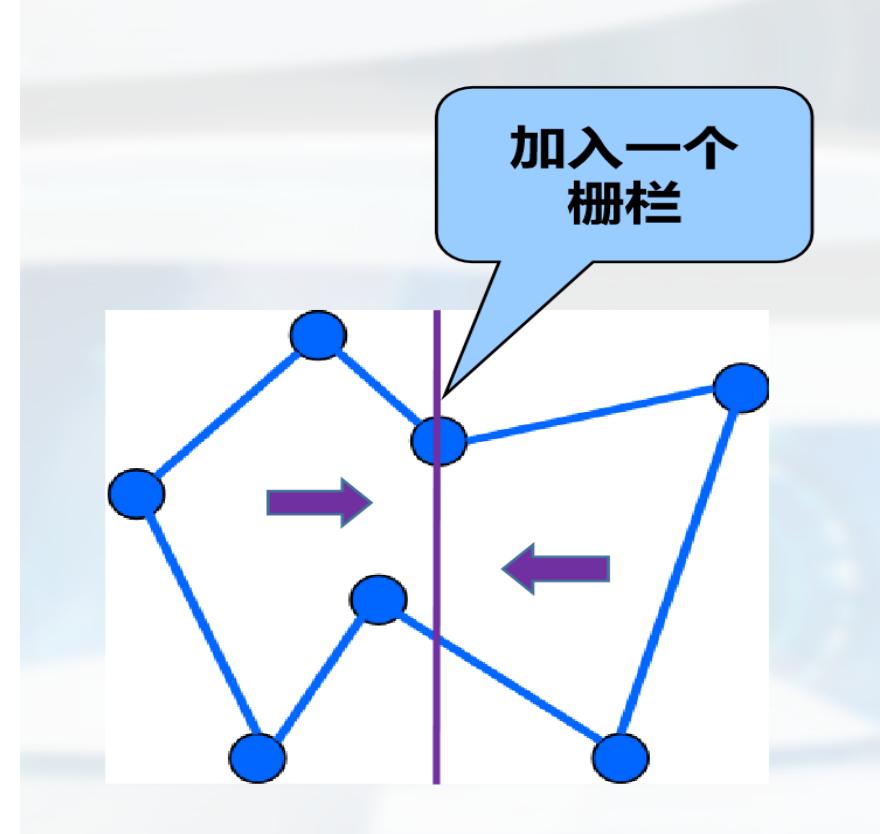
- 思想：

外接矩形区域中加入一个栅栏（过多边形顶点与扫描线垂直，把多边形分成两半的一条直线）

对于每条扫描线和每条多边形边的交点，只将交点和栅栏之间的像素取补。

- 优缺点：

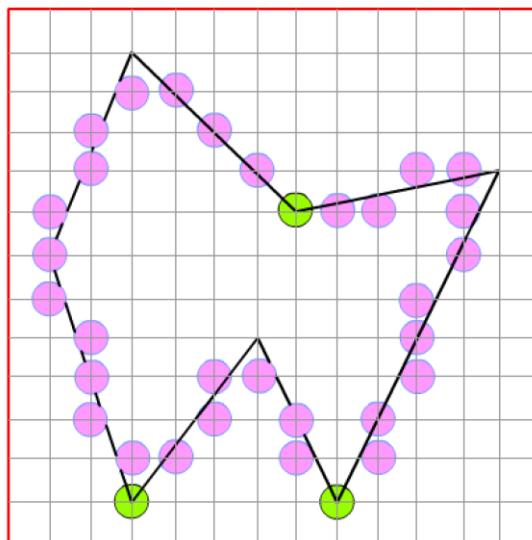
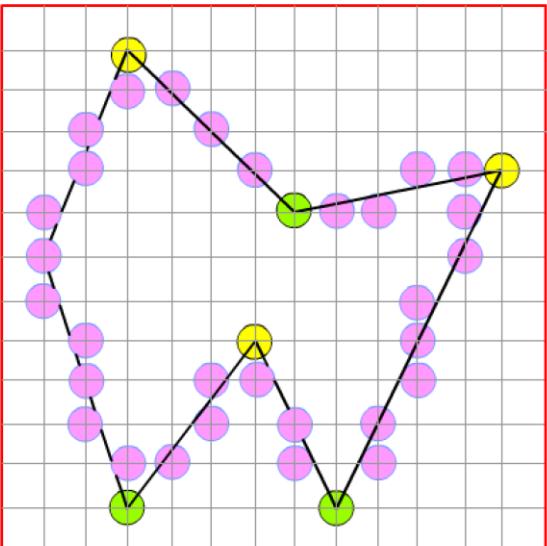
减少像素访问次数，
但仍有大量像素被重复访问。治标不治本



3) 边标志算法 (演示)

Step1:打标记:

- 对多边形每条边进行扫描转换 (即: 边界像素点被打上标记)
- 对一条扫描线来说, 标记点个数应该是偶数(局部最高点计为零个点, 最低点处理为两个点)
- `



- 1) 打标记
- 2) 局部最高点处理
为零个重叠交点
- 3) 局部最低点处理
为两个重叠交点
- 4) 填充, 按左闭右开的原则

3) 边标志算法-续

Step2:填充（左闭右开原则）

```
inside=FALSE;//设定一个布尔量
```

```
//沿着每条扫描线对每个像素点进行判定处理
```

```
For(扫描线上的每个像素从左到右)
```

```
{
```

```
    If(像素点是“标记点”)    inside=取反(inside);
```

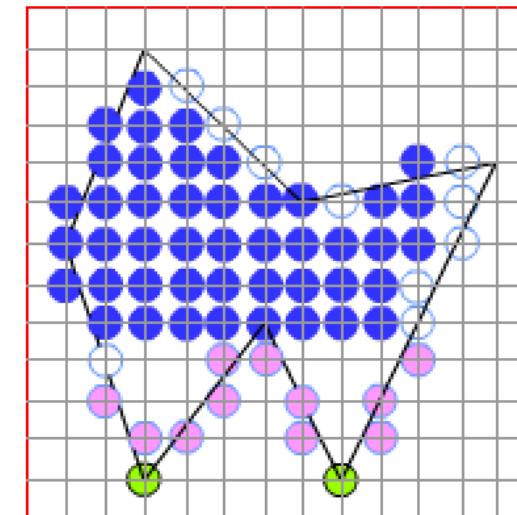
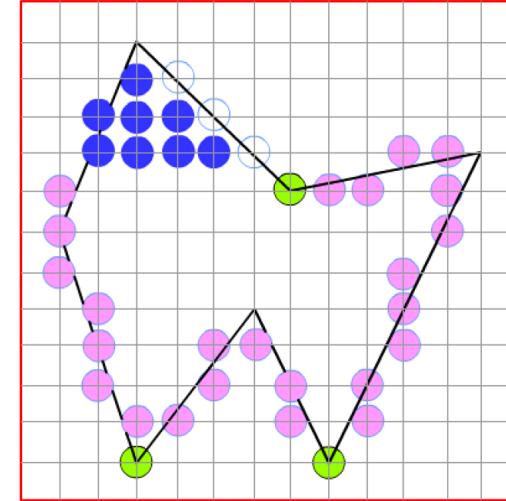
```
    If(inside)
```

```
        该像素填充;
```

```
    else
```

```
        该像素不填充;
```

```
};
```



边标记算法-算法分析

- 对每个像素仅访问一次

避免了对大量元素的多次赋值，但是仍需要逐条扫描线地对帧缓存中的元素进行搜索和比较。

- 适合硬件实现

软件实现与Y向连贯性算法（活性边表）相当，但是用硬件实现后速度会有很大提高。

多边形扫描转换算法

- 多边形表示
- 扫描转换算法
 - X-扫描线算法
 - Y-向连贯性算法
 - 边标志算法
- **区域填充算法**
 - 种子填充算法

区域表示：边界表示和内点表示

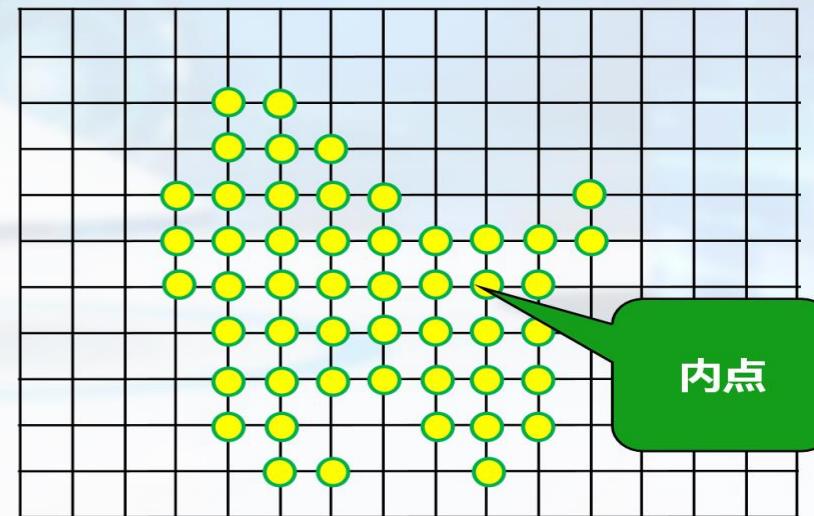
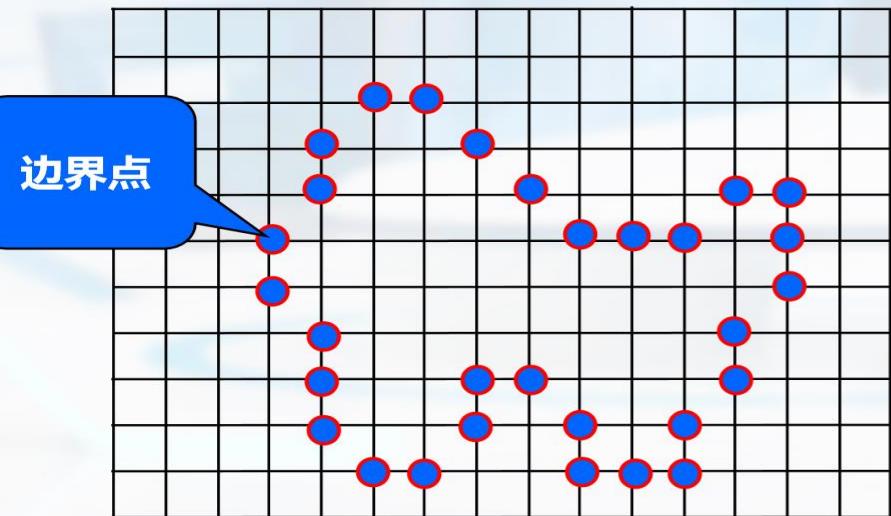
1

区域的定义

区域的定义：指已经表示成点阵形式的填充图形，它是像素集合。

两种表示形式：

- ◆ 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称为边界表示法。
- ◆ 内点表示法：枚举出给定区域内所有象素的表示方法称为内点表示。



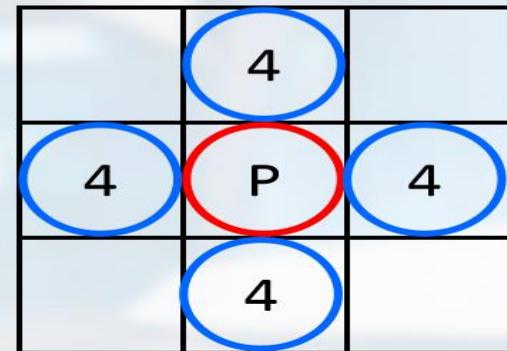
区域分类：4连通和8连通

1

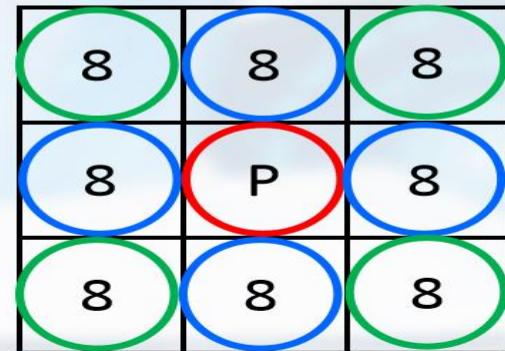
区域的定义

区域的分类：4连通和8连通？

定义：4-邻接点 8-邻接点



P的4-邻接点



P的8-邻接点

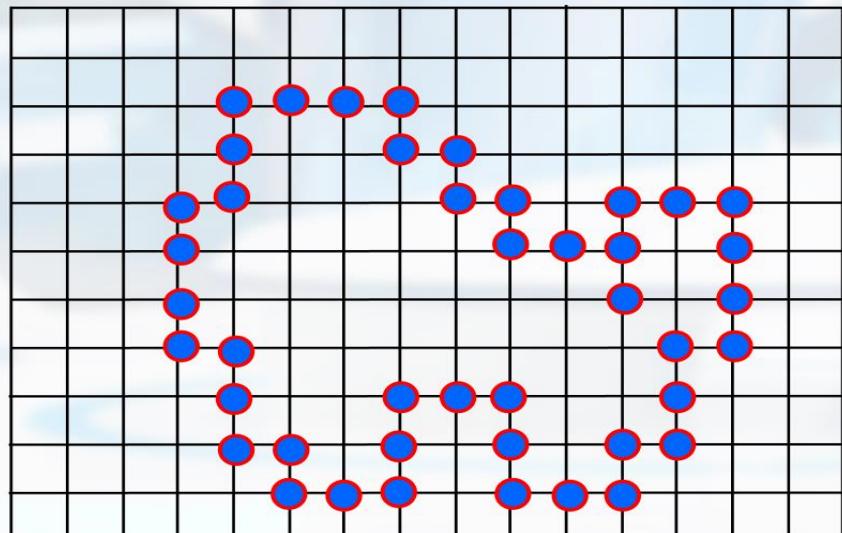
区域填充算法的分类：边界填充和泛填充

- 边界表示区域填充~边界填充算法 (4连通和8连通)
- 内点表示区域填充~泛滥填充算法 (4连通和8连通)

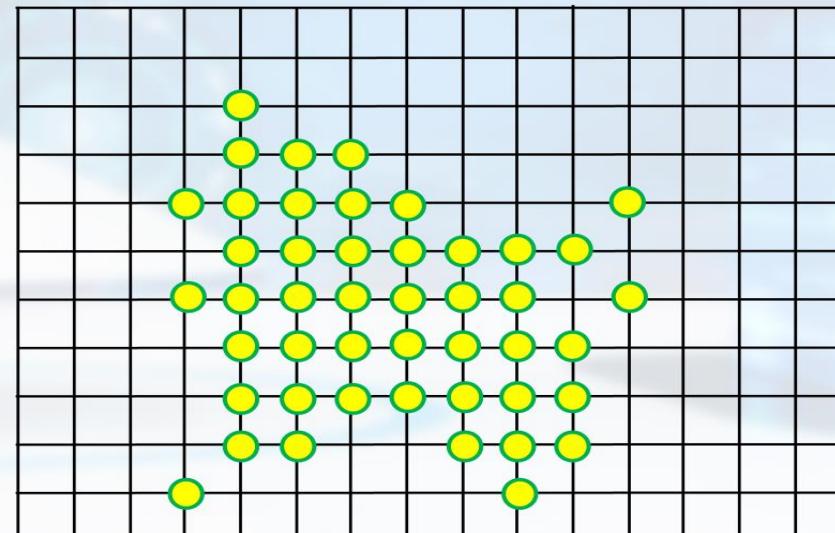
区域填充算法的分类：

4连通和8连通 → [4连通算法
8连通算法]

针对内点表示还是边界表示 → [边界填充 (边界表示)
泛填充 (内点表示)]



8连通边界填充算法

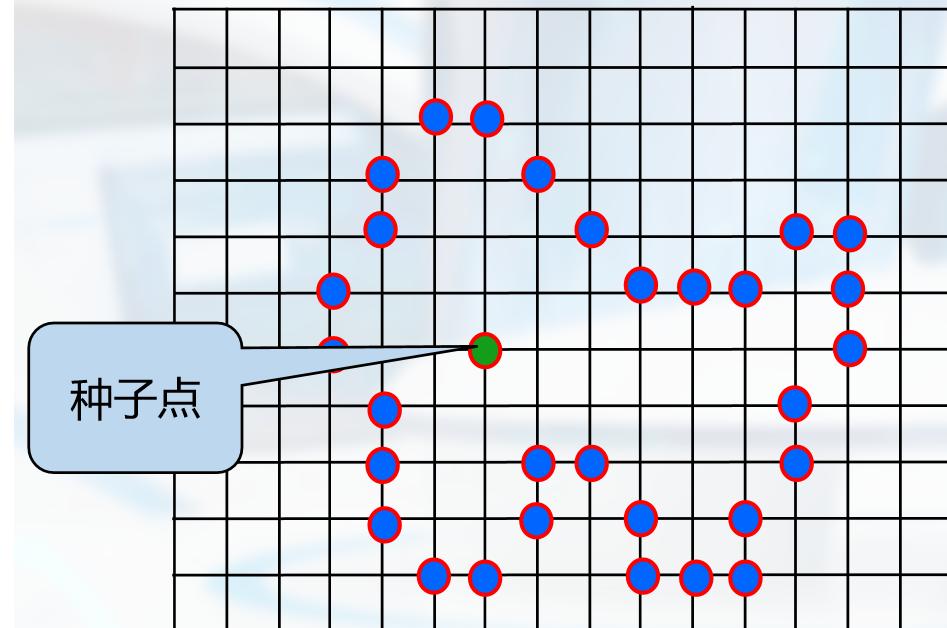


8连通泛填充算法

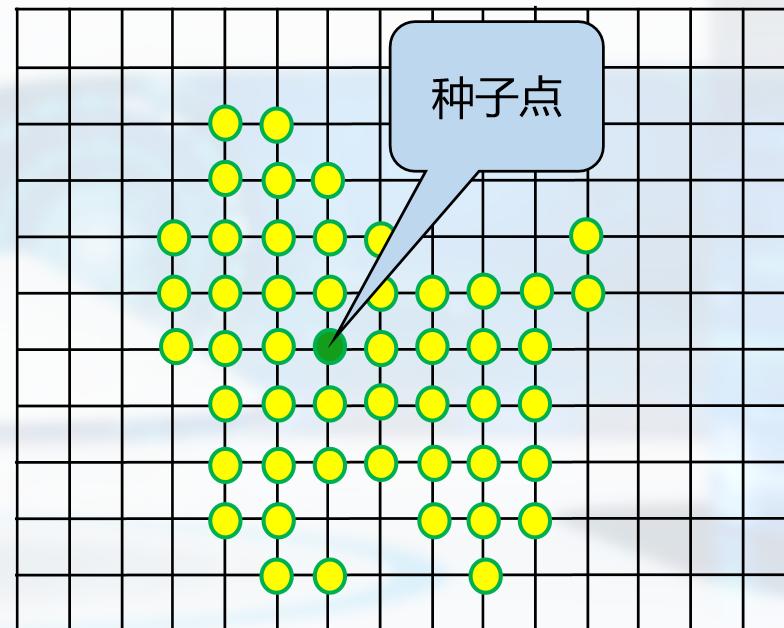
种子

- 无论边界表示还是内点表示，都采用的是种子填充算法思想

种子的定义：边界表示区域内的任意一点或者内点表示区域的任意一点



4连通边界表示



4连通内点表示

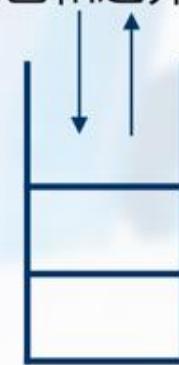
种子填充算法（边界填充算法）

边界填充算法

算法输入：种子点坐标(x,y)，填充色和边界颜色。

数据结构：栈结构

算法输出：最佳逼近的像素点集



4-连通边界填充算法步骤：

种子象素入栈，栈非空时重复执行三步操作：

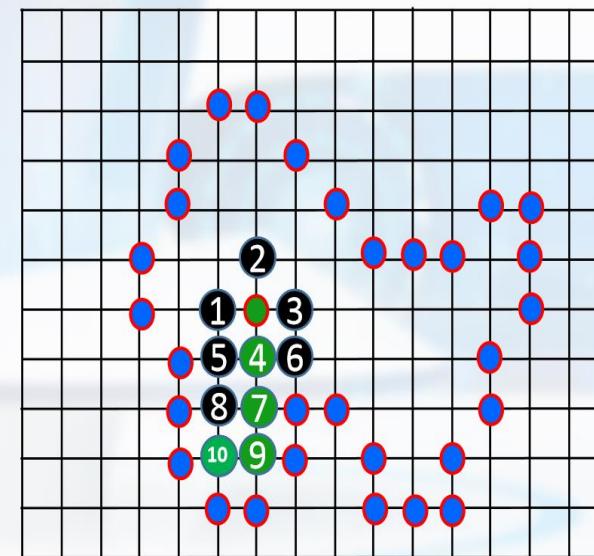
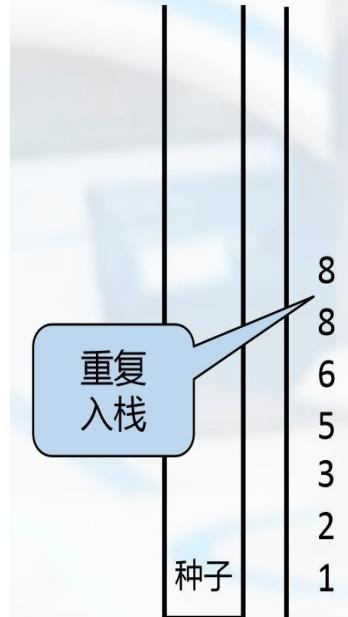
(1) 栈顶象素出栈；

(2) 将出栈象素置成填充色；

(3) 检查出栈象素的4-邻接点，若其中某个象素点

不是边界色且未置成多边形色，则把该象素入栈。

以边界表示为例看种子填充思想：



4连通边界表示

边界填充算法（演示）

边界表示的区域填充算法

缺陷：需要较大的存储相邻点的栈空间

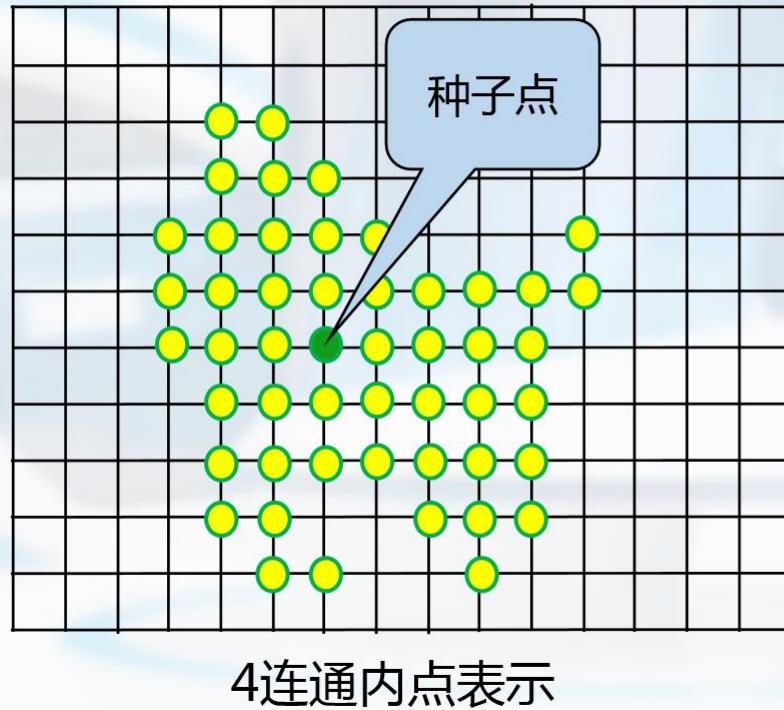
区域颜色在参数**fill**中指定；边界颜色用参数**boundary**指定

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{
    int current; current=getPixel (x, y);
    if((current !=boundary) && (current !=fill))
    {setColor (fill);
        setPixel (x, y);
        boundaryFill4 (x+1, y, fill, boundary);
        boundaryFill4 (x-1, y, fill, boundary);
        boundaryFill4 (x, y+1, fill, boundary);
        boundaryFill4 (x, y-1, fill, boundary);
    }
}
```

种子填充算法（泛填充算法）

内点表示区域

换成4连通内点表示的区域：



4-连通泛填充算法步骤如下：

种子象素入栈；当栈非空时重复执行如下三步操作：

- (1) 栈顶象素出栈；
- (2) 将出栈象素置成填充色；
- (3) 检查出栈象素的4-邻接点，若其中某个象素点是给定内部点的颜色且未置成新的填充色，则把该象素入栈。

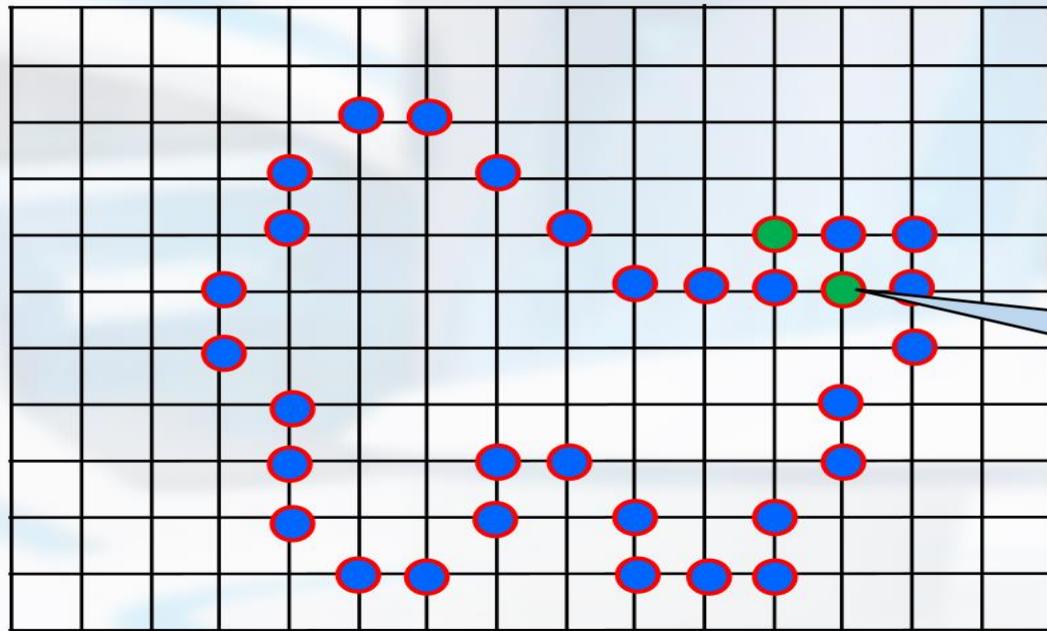
泛滥填充算法（演示）

- 泛滥填充算法：内点定义的区域
- 通过替换指定的内部颜色来对这个区域涂色(填充)。从指定的内部点(x,y)开始，用所希望的填充颜色赋给所有当前设置为给定内部颜色的像素。

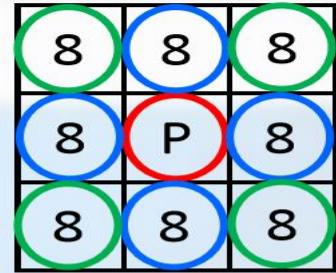
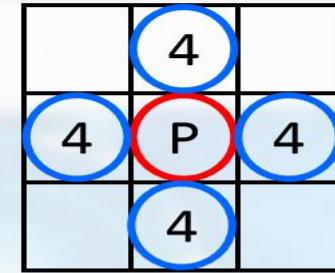
```
void floodFill4(int x,int y,int fillColor,int oldColor) {  
    int cent=getpixel(x,y);  
    if (cent==oldColor && cent!=fillcolor)  
    {  
        setColor(fillColor);  
        setpixel(x,y);  
        floodFill4(x+1,y,fillColor,oldColor);  
        floodFill4(x-1,y,fillColor,oldColor);  
        floodFill4(x,y+1,fillColor,oldColor);  
        floodFill4(x,y-1,fillColor,oldColor);  
    }  
}
```

分析和分析1

(1) 8连通边界算法可以填充4连通的边界表示区域吗？



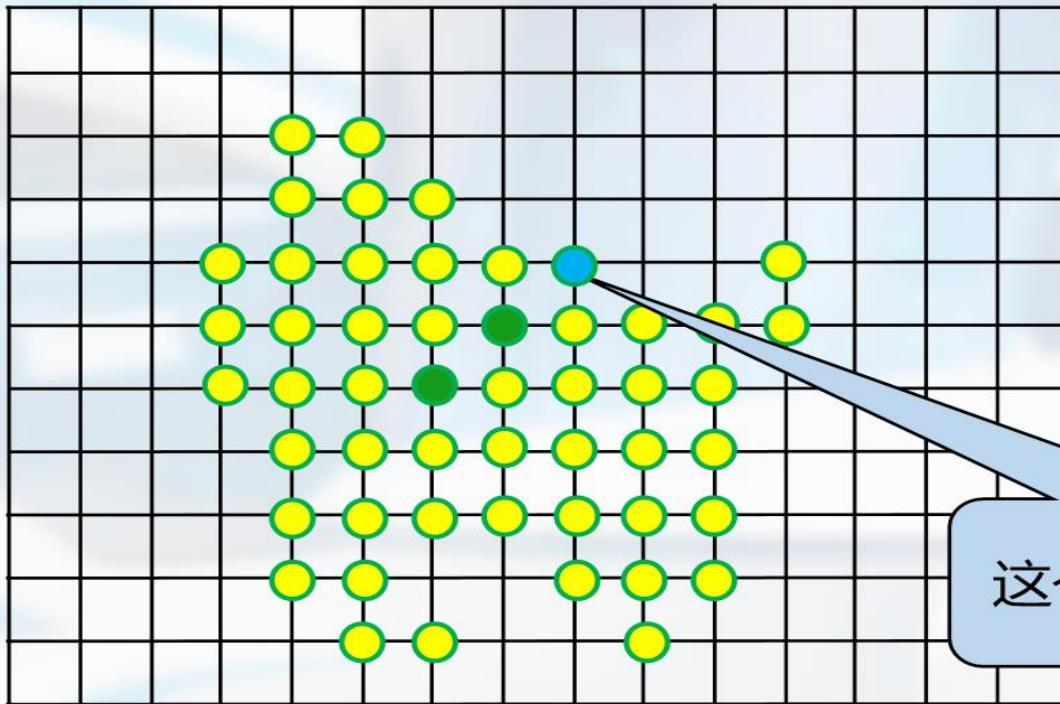
4连通边界表示



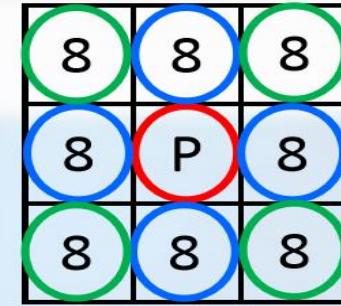
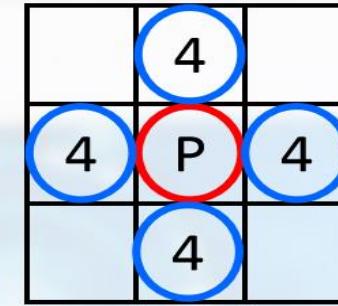
这个点

分析和改进

(2) 8连通泛填充算法可以填充4连通的内点表示区域吗 ?



4连通内点表示

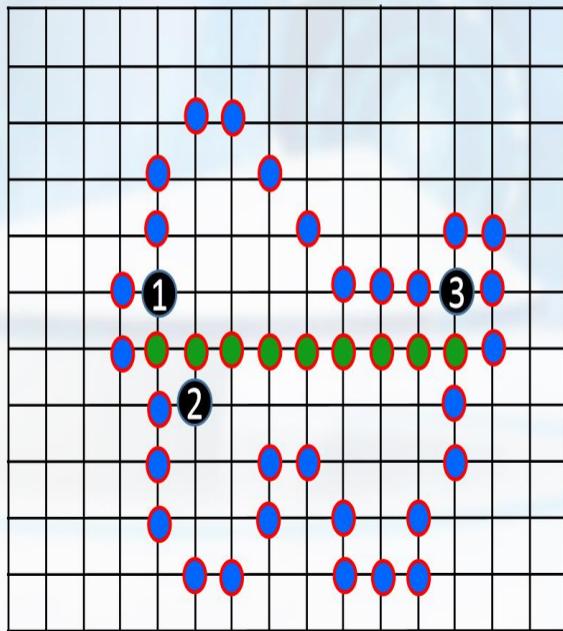


这个点会被填色吗 ?

分析和改进-水平像素段填充

(3) 有重复入栈的现象，如何提高效率？

种子出栈时填充水平像素段：



4连通边界表示

3
2
1

- 为减少存储相邻点的栈空间，可通过沿扫描线填充水平像素段来代替处理4连通或8-连通相邻点。
- 扫描线边界填充过程
 - 1)从种子点开始，首先填该像素所在扫描行的连续像素段；
 - 2)将相邻扫描线上各段的起始位置进栈
 - 3)从栈顶逐步取出开始点(先进后出)，填充该水平段像素；
 - 4)回到3操作，直到堆栈空结束