

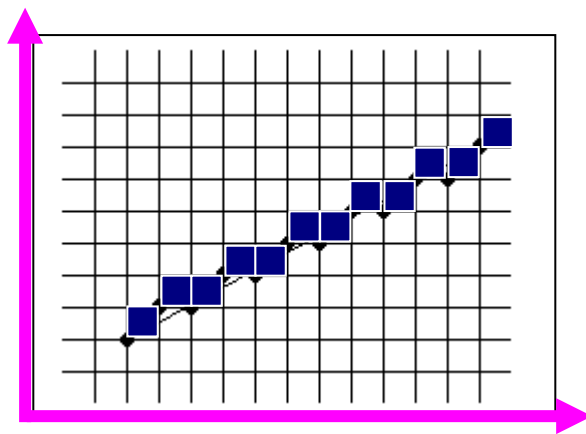
第三章 基本图形生成算法

内容提要

- ✓ 直线生成算法
- ✓ 圆弧生成算法
- ✓ 线宽和线型的处理
- ✓ 实区域填充算法
- ✓ 图形反走样技术

3.5 实区域填充算法

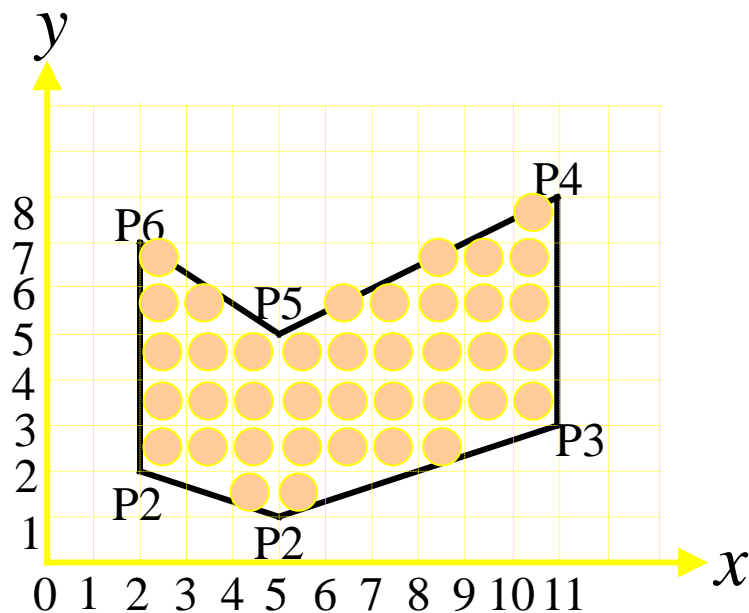
直线的扫描转换



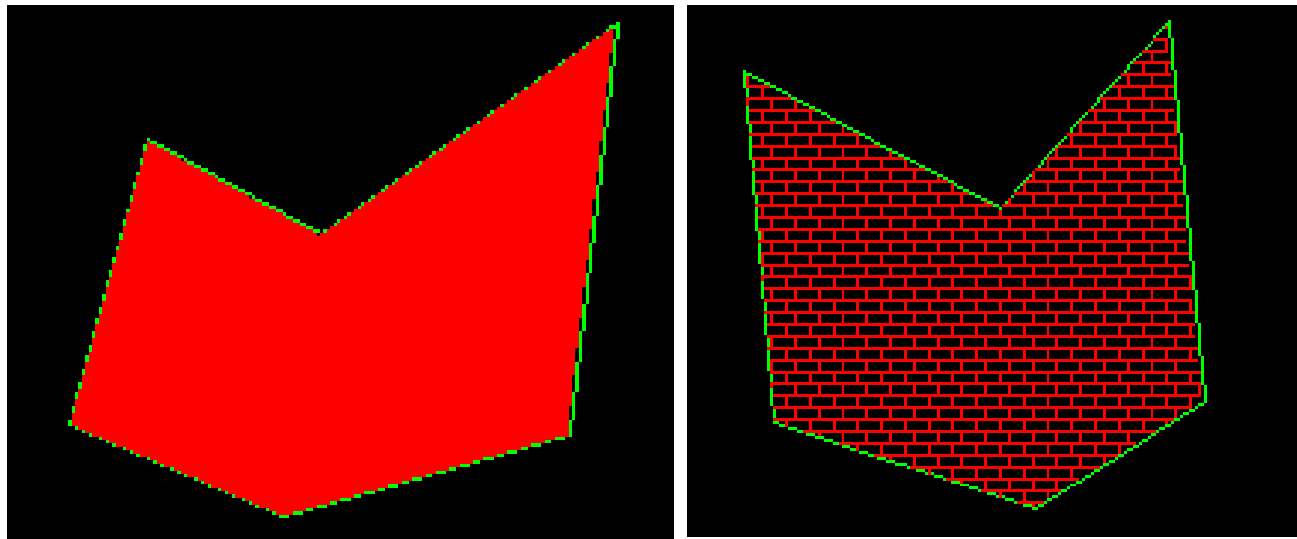
假设坐标原点位于左下角点
则像素由其左下角坐标表示

3.5 实区域填充算法

- ✓ 解决的主要问题是什么？
 - 确定待填充的像素
 - 即检查光栅屏幕上的每一像素是否位于多边形区域内



3.5 实区域填充算法



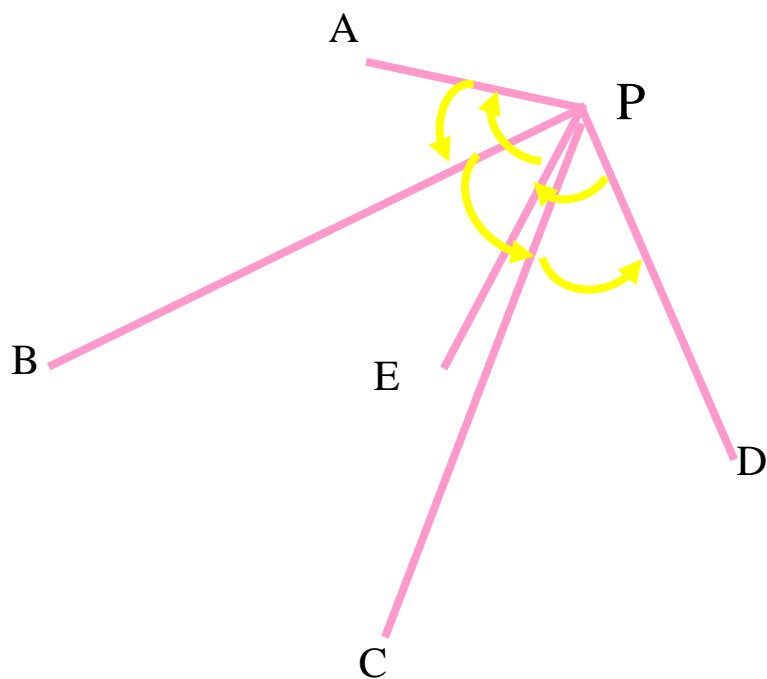
- ✧ 图案填充还有一个什么像素填什么颜色的问题
- ✧ 曲线围成的区域，可用多边形逼近

实区域填充算法

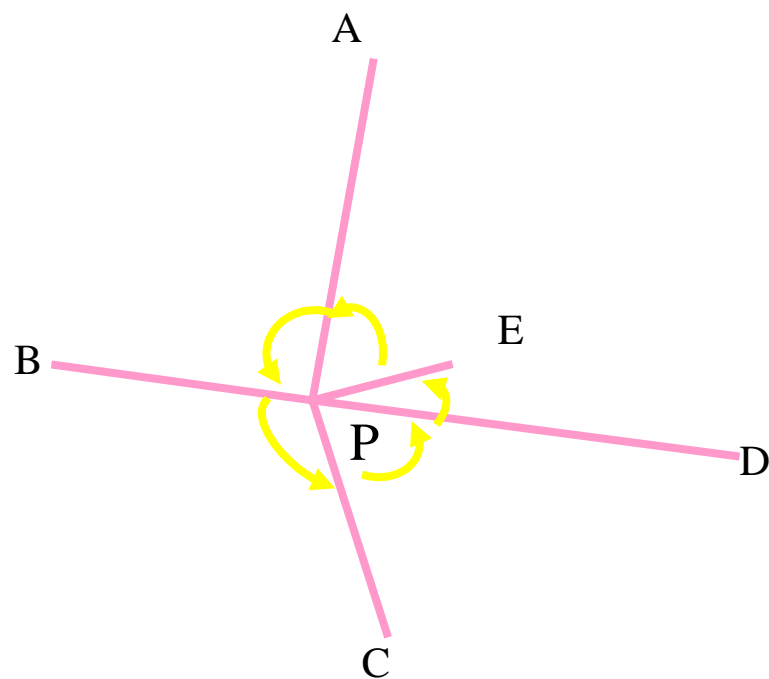
- ✧ 如何判断一个点是否位于多边形区域内？
- ✧ 点在多边形内的包含性检验
 - 检验夹角之和
 - 射线法检验交点数

实区域填充算法

检验夹角之和



若夹角和为 0 ，
则点 p 在多边形外



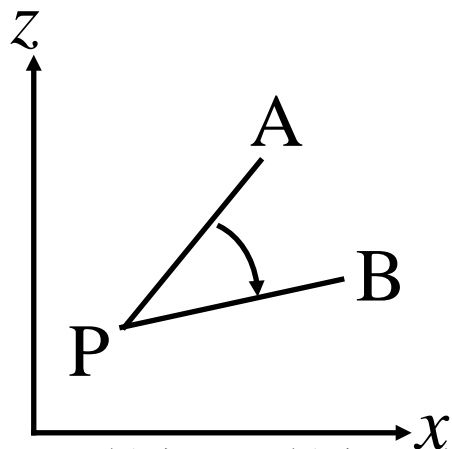
若夹角和为 360° ，
则点 p 在多边形内

实区域填充算法

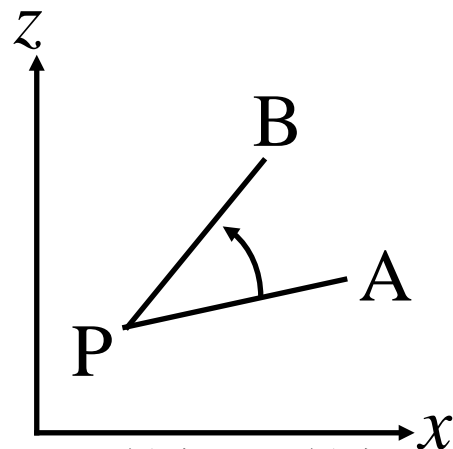
✧ 夹角如何计算？

- 大小：利用余弦定理
- 方向：令

$$T = \begin{vmatrix} x_A - x_P & z_A - z_P \\ x_B - x_P & z_B - z_P \end{vmatrix} = (x_A - x_P)(z_B - z_P) - (x_B - x_P)(z_A - z_P)$$



当 $T < 0$ 时，AP斜率 $>$ BP斜率，为顺时针角

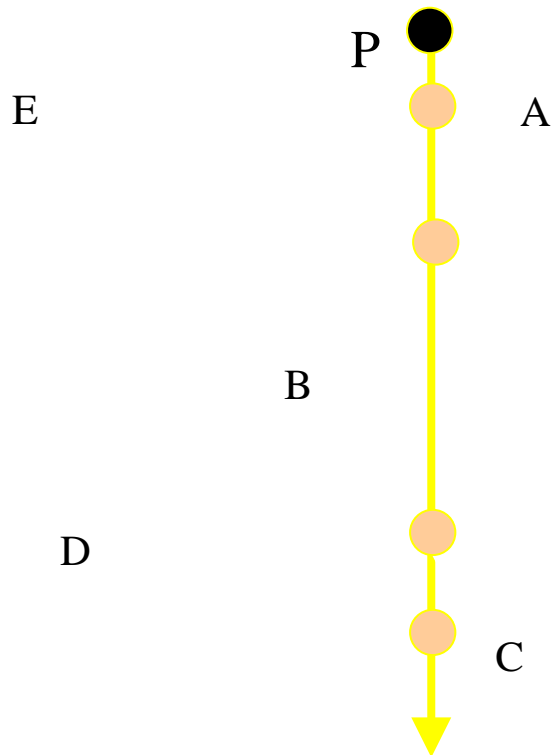


当 $T > 0$ 时，AP斜率 $<$ BP斜率，为逆时针角

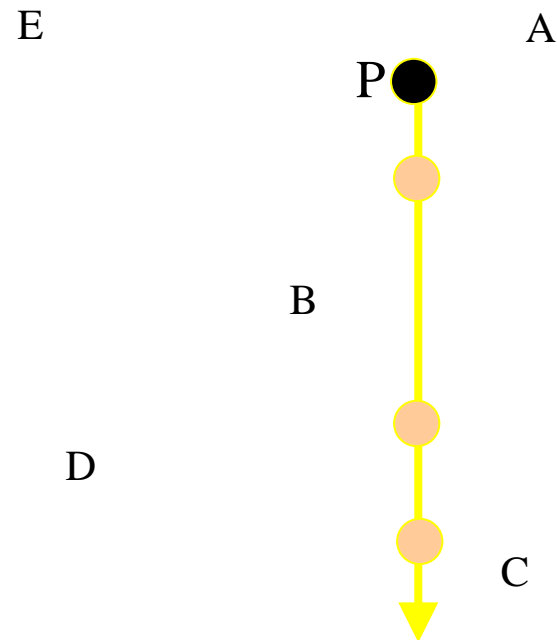
实区域填充算法

射线法检验交点数

交点是顶点怎么办？



交点数=偶数（包括0）
点 在多边形之外



交点数=奇数
点 在多边形之内

实区域填充算法

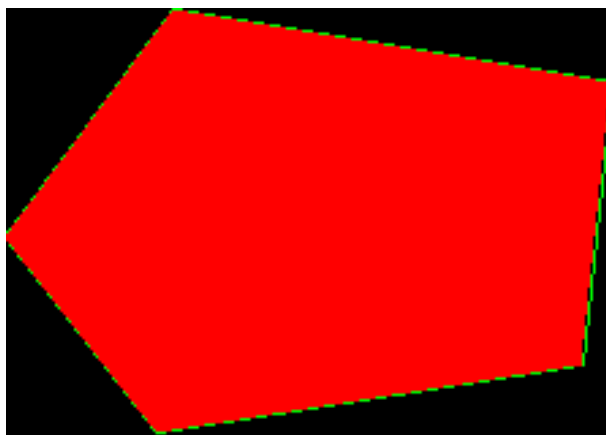
✧ 逐点测试的问题？

- 效率低不实用

✧ 一个简单的解决方法

- 包围盒法

测试效率仍然很低



凸多边形



凹多边形

实区域填充算法



✓换一种思路：

✓考虑图形的扫描方式特点

— 能否利用扫描线的连贯性？

✓考虑图形的特点

— 能否利用图形的空间连贯性？

实区域填充算法

✓分类：

✓扫描线填充算法

- 按扫描线顺序，测试点的连贯性

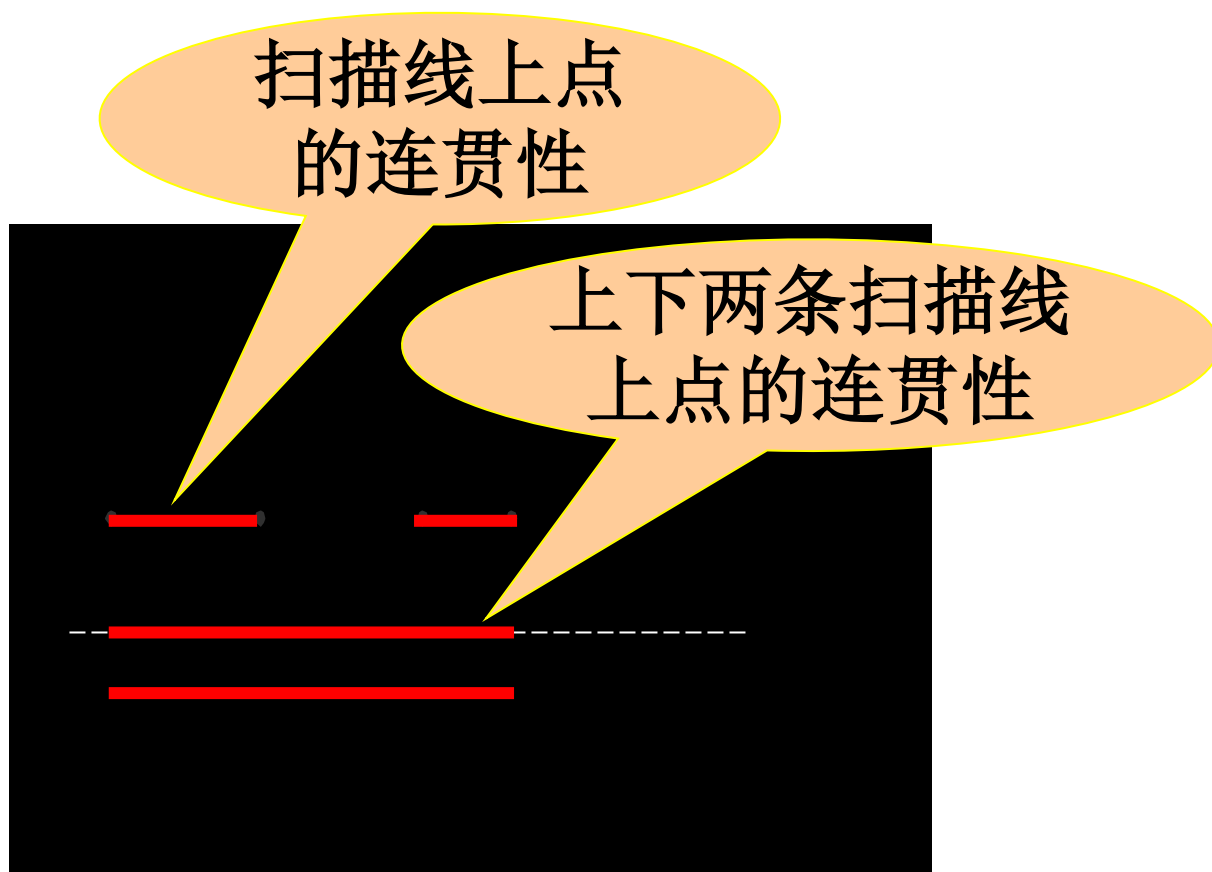
✓种子填充算法

- 从内部一个种子点出发

- 测试点的连贯性

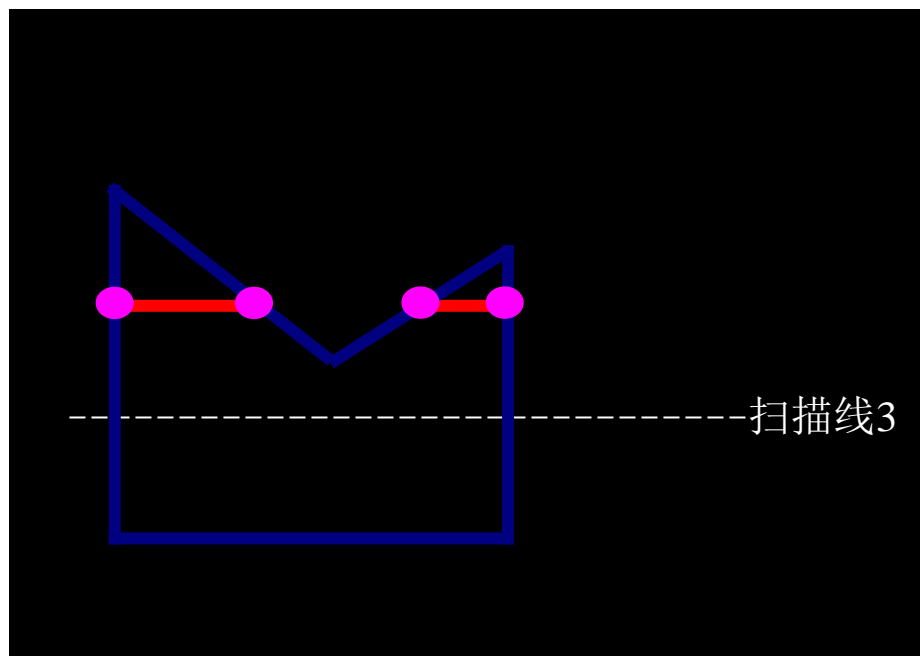
扫描线填充算法

扫描线的连贯性



扫描线填充算法

- ✓ 求交: I_4, I_3, I_2, I_1
- ✓ 排序: I_1, I_2, I_3, I_4
- ✓ 交点配对: $(I_1, I_2), (I_3, I_4)$
- ✓ 区间填色



扫描线填充算法



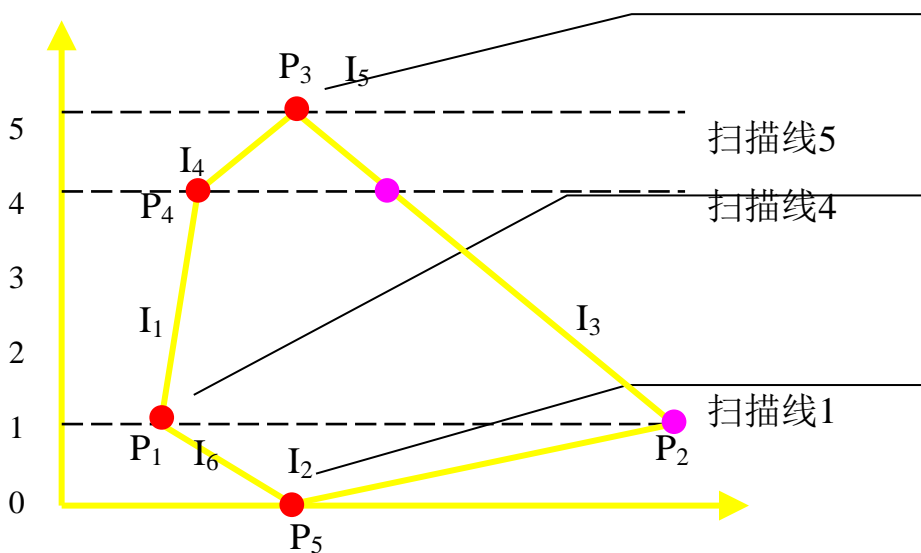
✓ 思考第一个问题：

✓ 交点配对时可能出现的问题？

— 会不会出现奇数个交点？

扫描线填充算法

✓ 顶点交点的计数问题



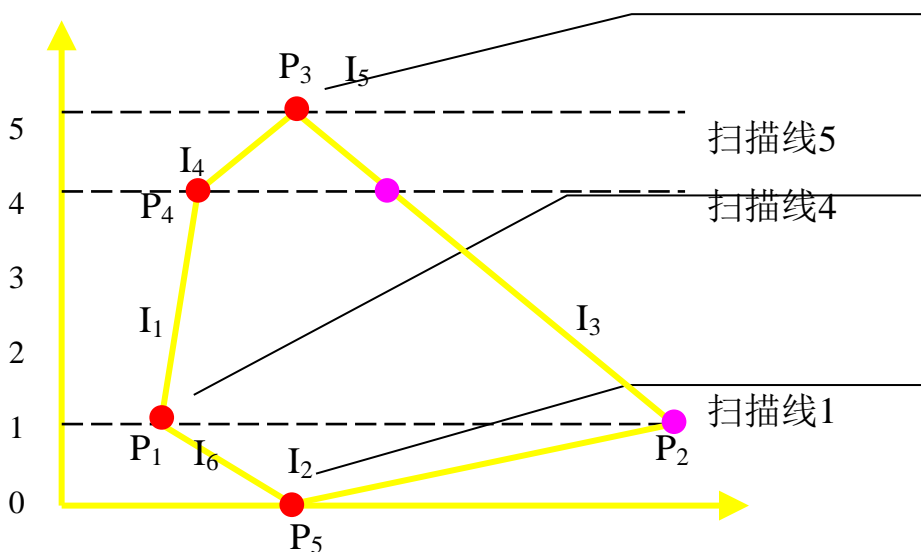
也计数1次吗？

应该计数1次

也计数1次吗？

扫描线填充算法

✓ 顶点交点的计数问题



也计数1次吗？

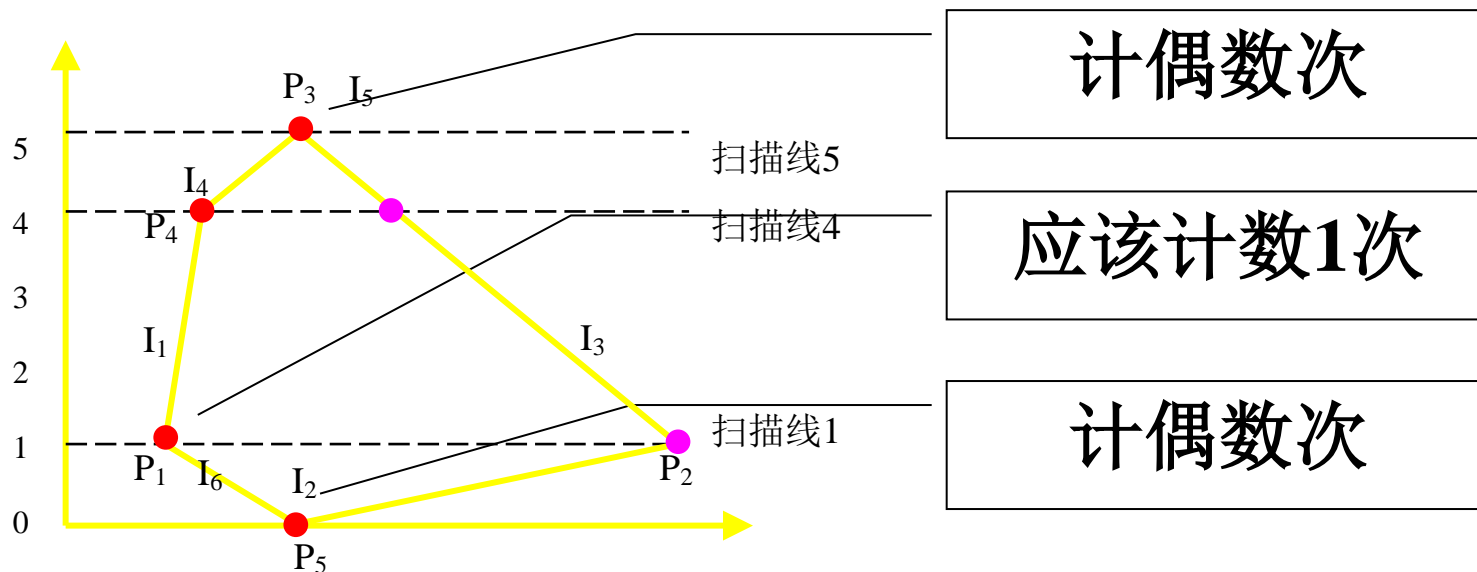
应该计数1次

也计数1次吗？

- 局部最高点和局部最低点的交点计数问题？
- 如何判断局部最高点和局部最低点？

扫描线填充算法

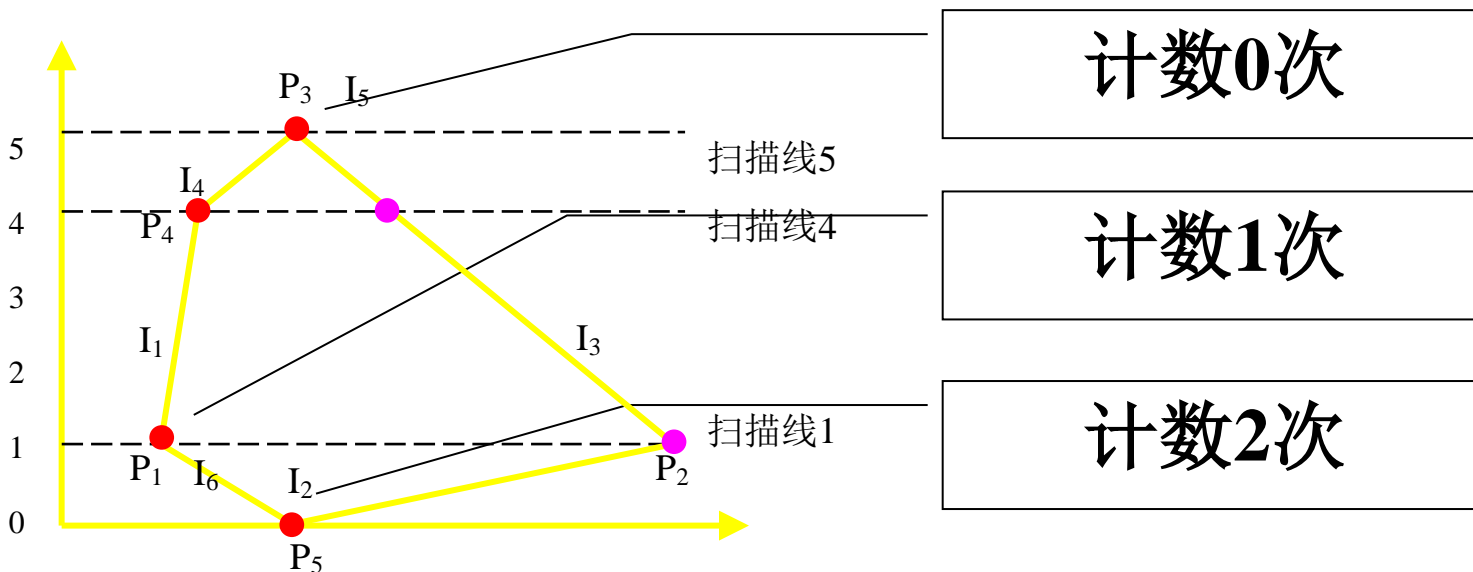
✓ 顶点交点的计数问题



– 局部最高点和局部最低点
计偶数次交点

扫描线填充算法

✓ 顶点交点的计数问题



- 检查交于该顶点的两条边的另外两个端点的y坐标值大于该顶点y坐标值的个数

扫描线填充算法



✓ 思考第2个问题

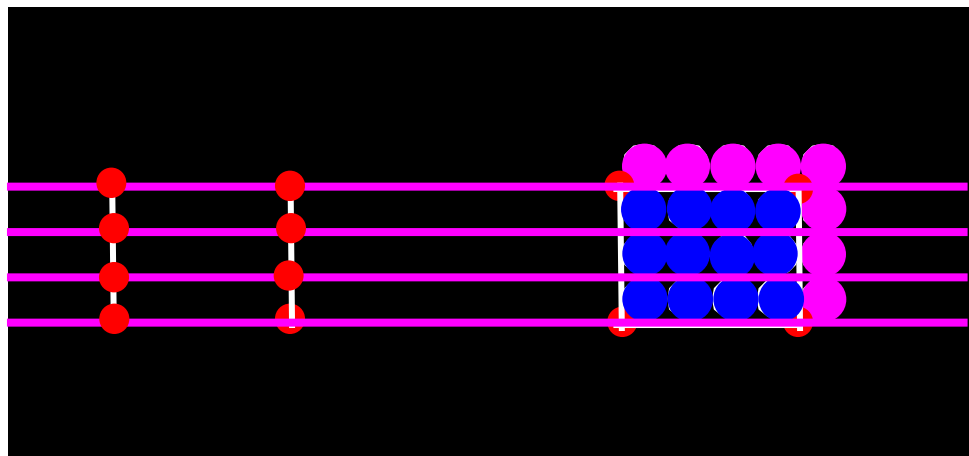
✓ 区间填色时可能出现的问题？

— 会不会填充到区域之外呢？

扫描线填充算法

✓ 填充扩大化问题

- 求出交点的坐标为: $x_l=1, x_r=5$
- 有5个点在配对区间内
- 即满足: $x_l \leq x \leq x_r$



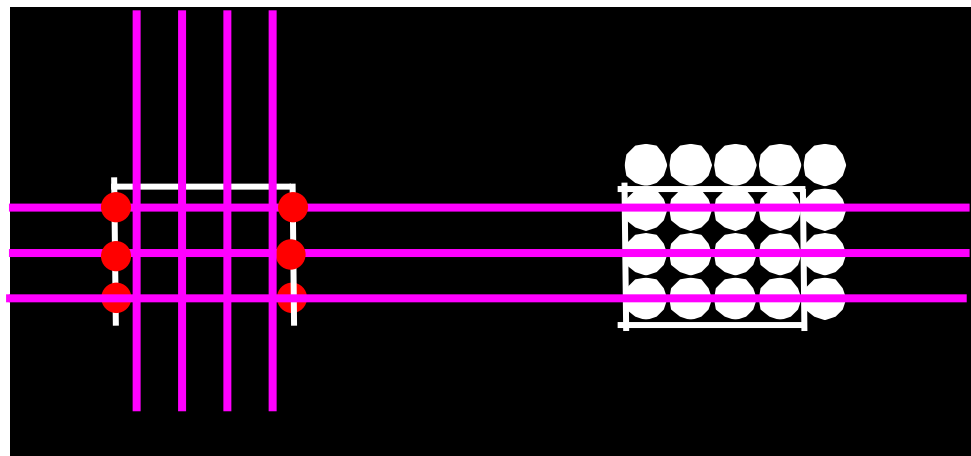
扫描线填充算法



✓ 如何解决填充扩大化问题？

- 取中心扫描线 $y+0.5$
- 检查交点右方像素的中心是否落在区间内

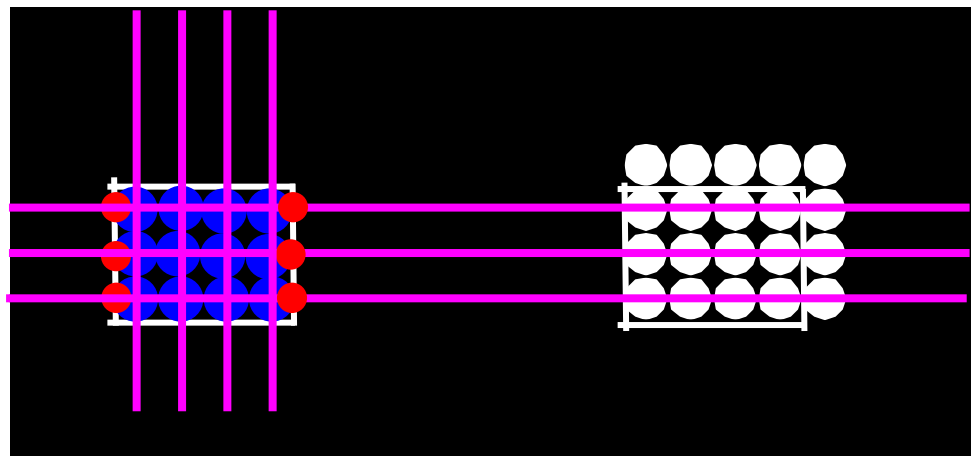
$$x_l \leq x+0.5 \leq x_r$$



扫描线填充算法

✓ 填充过程

- 求出交点的坐标为： $x_l=1, x_r=5$
- 有4个点在配对区间内
- 即满足： $x_l \leq x+0.5 \leq x_r$



扫描线填充算法



✎ 思考第3个问题：

✎ 效率问题

扫描线填充算法



- ✓ 影响算法效率的因素是什么？
 - 求交和交点排序
 - 把多边形所有边放在一个表中
 - 按顺序取出
 - 分别计算与当前扫描线求交点

扫描线填充算法

- ✓ 并非所有的边都与当前扫描线有交点？
- ✓ 如何提高效率？
 - 尽量减少和简化求交计算

扫描线填充算法

- ✓ 如何减少和简化求交计算？
- ✓ 对每条扫描线，建立一个活性边表
 - 何谓活性边？
 - 仅与当前扫描线有交点的边
- ✓ 活性边表
 - 把所有与当前扫描线有交点的边放到一个表中存储

扫描线填充算法

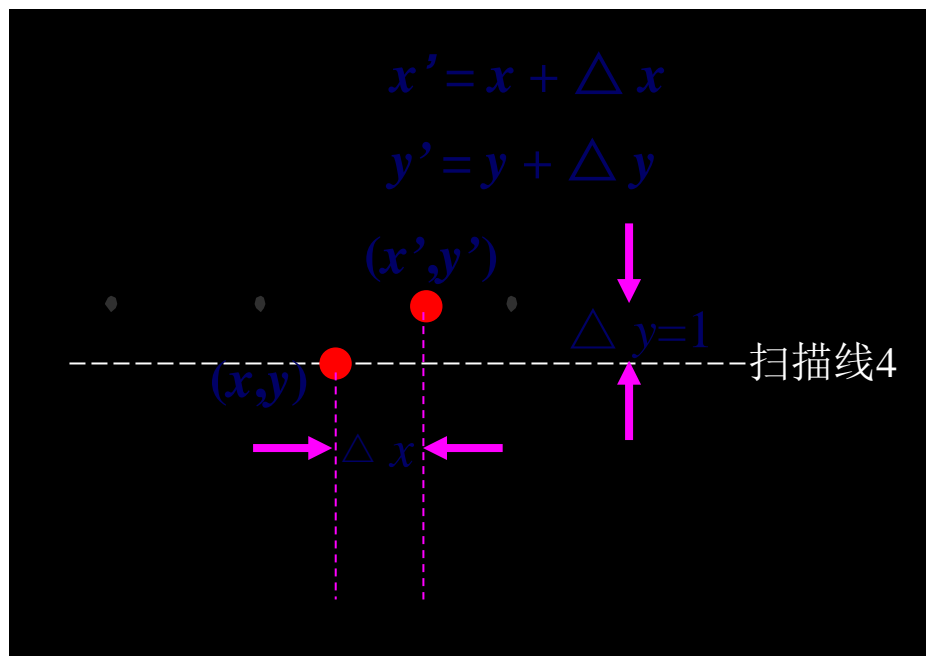
✓ 存储活性边的哪些信息呢？

- 目的是要保证存储的这些信息，
对计算活性边与扫描线的交点有用
- 最有用的信息莫过于
当前扫描线与活性边的交点 x
- 为了下一条扫描线与活性边求交方便
- 我们还需要知道什么信息呢？

扫描线填充算法

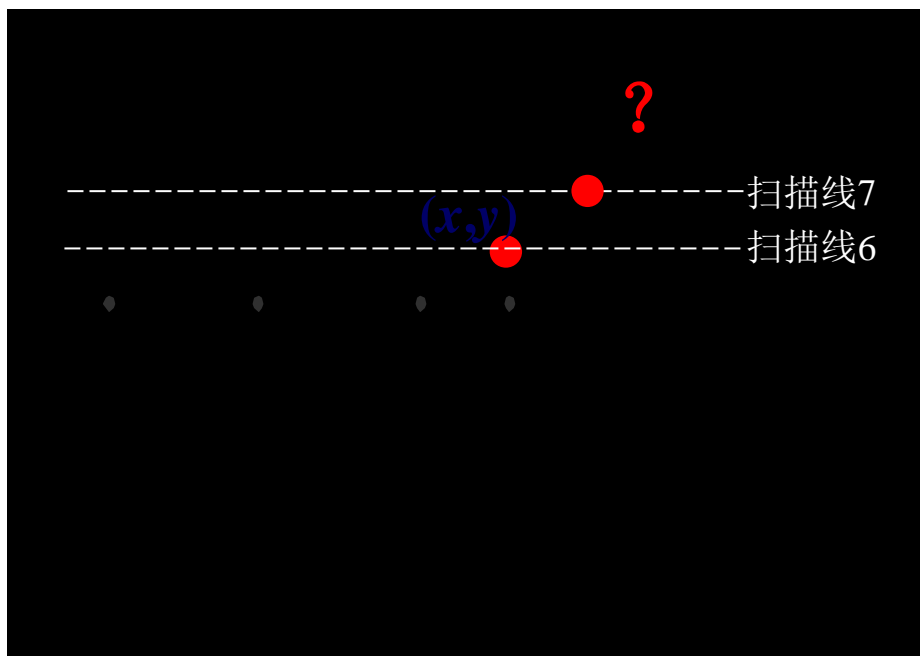
- 为下一条扫描线与活性边求交方便
 - 从当前扫描线到下一条扫描线的 x 增量
 - $\Delta x = ?$

$$\Delta x = 1/k$$



扫描线填充算法

- ✓ 下一条扫描线是否与活性边总有交点呢？
 - 记录活性边所交的最高扫描线号 y_{max}
 - 当 $y > y_{max}$ 时，将活性边从活性边表中删除



扫描线填充算法

✓ 活性边表的建立

✓ 结点信息

x : 当前扫描线与边的交点

Δx : 从当前扫描线到下一条扫描线之间的 x 增量

y_{max} : 边所交的最高扫描线号

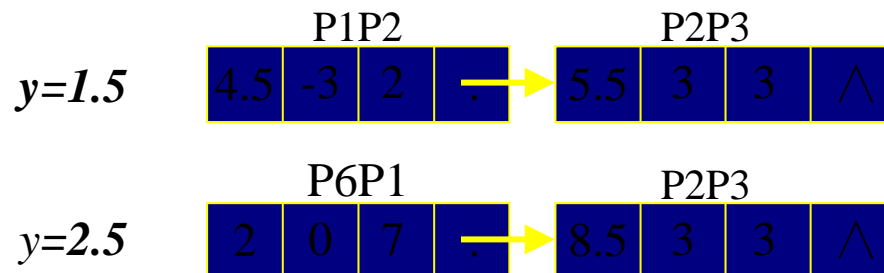
x	Δx	y_{max}	
-----	------------	-----------	--

扫描线填充算法

- ✓ 活性边表的更新
- ✓ 结点信息的更新
- ✓ 旧边的删除
- ✓ 新边的插入

x	Δx	y_{max}	
x'	Δx	y_{max}	

$$x' = x + \Delta x$$



扫描线填充算法

- ✎ 如何解决新边插入的问题？
- ✎ 对每条扫描线建立一个新边表
- ✎ 新边表需要什么结点信息？
- ✎ 以方便建立活性边表的结点信息为目的
- ✎ 最好是可以将新边表中的信息直接插入到活性边表中

扫描线填充算法

- ✓ 由于是新边，扫描线与边的交点应为
- ✓ *扫描线与边的初始交点*

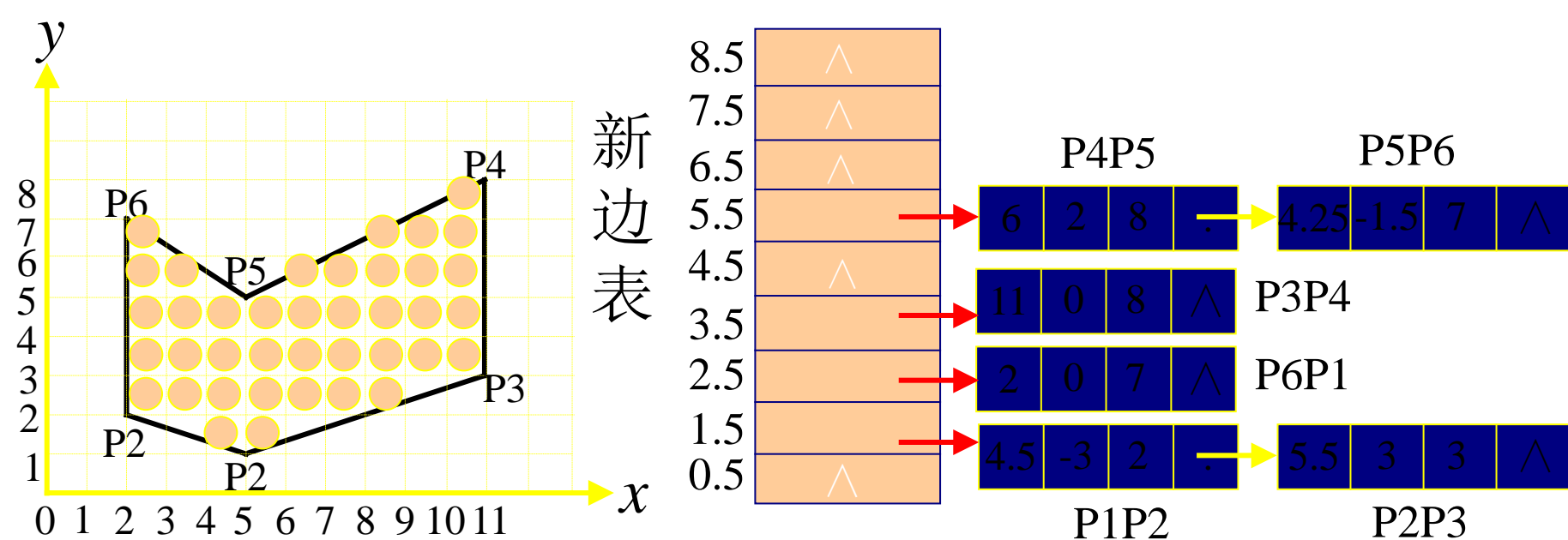
✓ 结点信息

x_0 : *扫描线与边的初始交点*

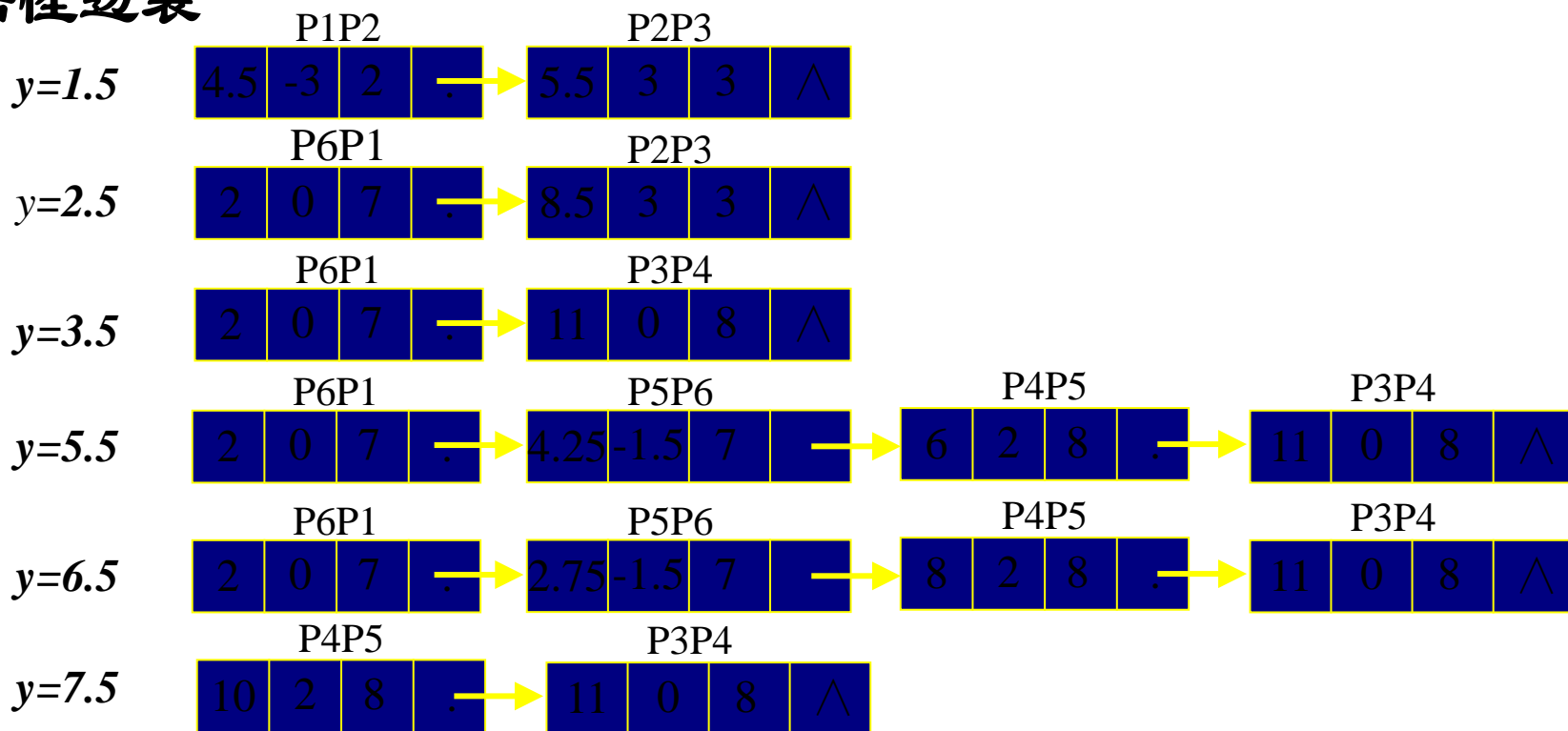
Δx : *从当前扫描线到下一条扫描线之间的x增量*

y_{max} : *边所交的最高扫描线号*

x_0	Δx	y_{max}	
-------	------------	-----------	--



活性边表



扫描线填充算法

✓ 称为：有序边表算法

✓ 优点：

- 对每个像素只访问一次
- 与设备无关

✓ 缺点：

- 数据结构复杂，表的维护、排序开销大
- 只适合软件实现

边填充算法

Edge Fill Algorithm

✔ 优点:

- 无需复杂的链表结构

✔ 涉及到屏幕像素的异或写操作

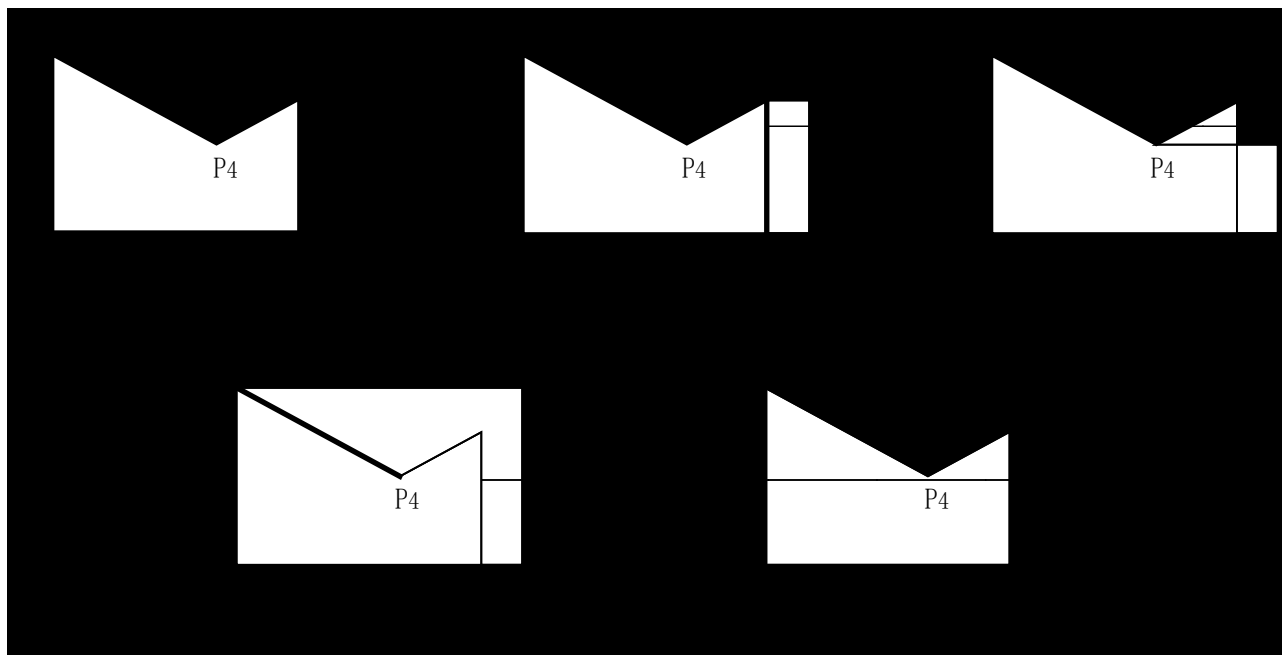
✔ 异或写操作有什么特点?

- 第一次异或写操作，像素被置成前景色
- 第二次异或写操作，像素恢复为背景色

边填充算法

✓ 基本思想

- 对每一条与多边形相交的中心扫描线
- 将像素中心位于交点右方的全部像素取补（异或写）



算法
演示

边填充算法

✔ 优点：

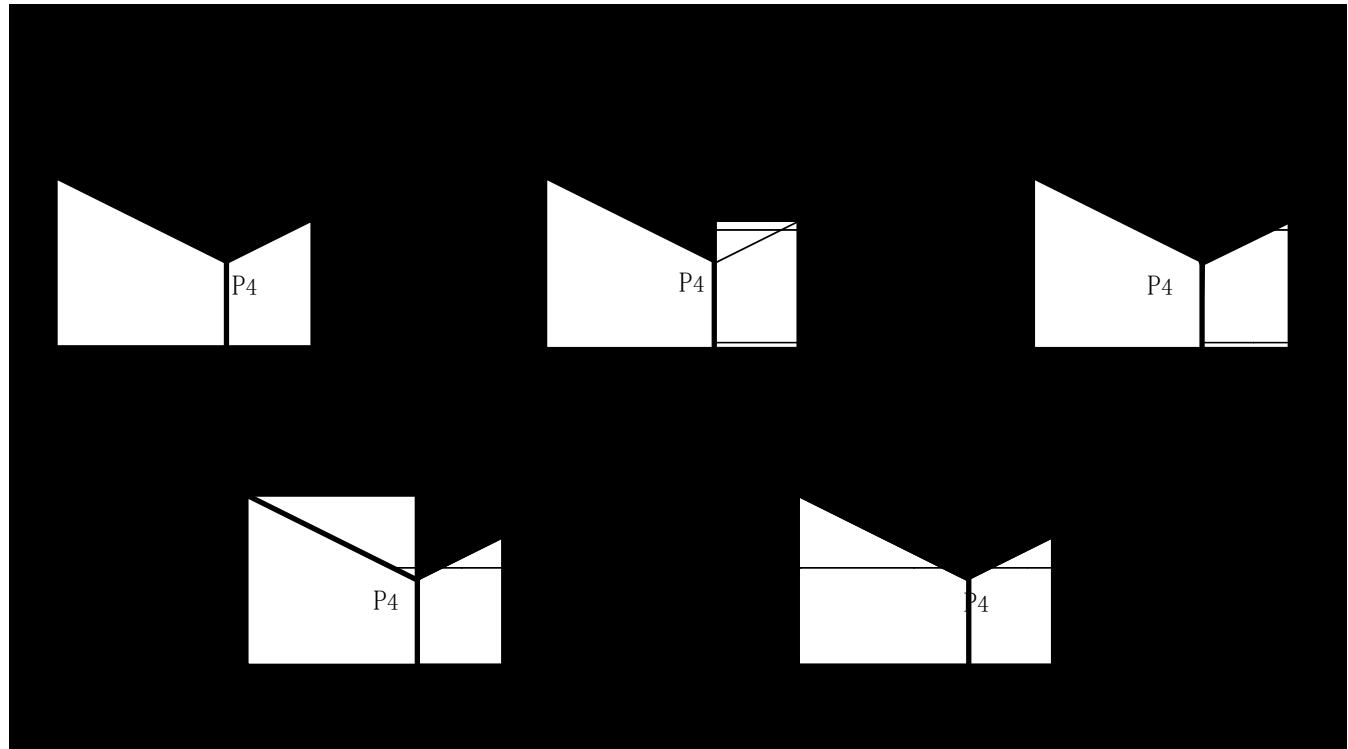
- 最适合于有帧缓存的显示器
- 可按任意顺序处理多边形的边
- 仅访问与该边有交点的扫描线上右方的像素，算法简单

✔ 缺点：

- 对复杂图形，每一像素可能被访问多次，输入/输出量大
- 图形输出不能与扫描同步进行，只有全部画完才能打印

栅栏填充算法 Fence Fill Algorithm

✓ 算法的改进 – 引入栅栏线



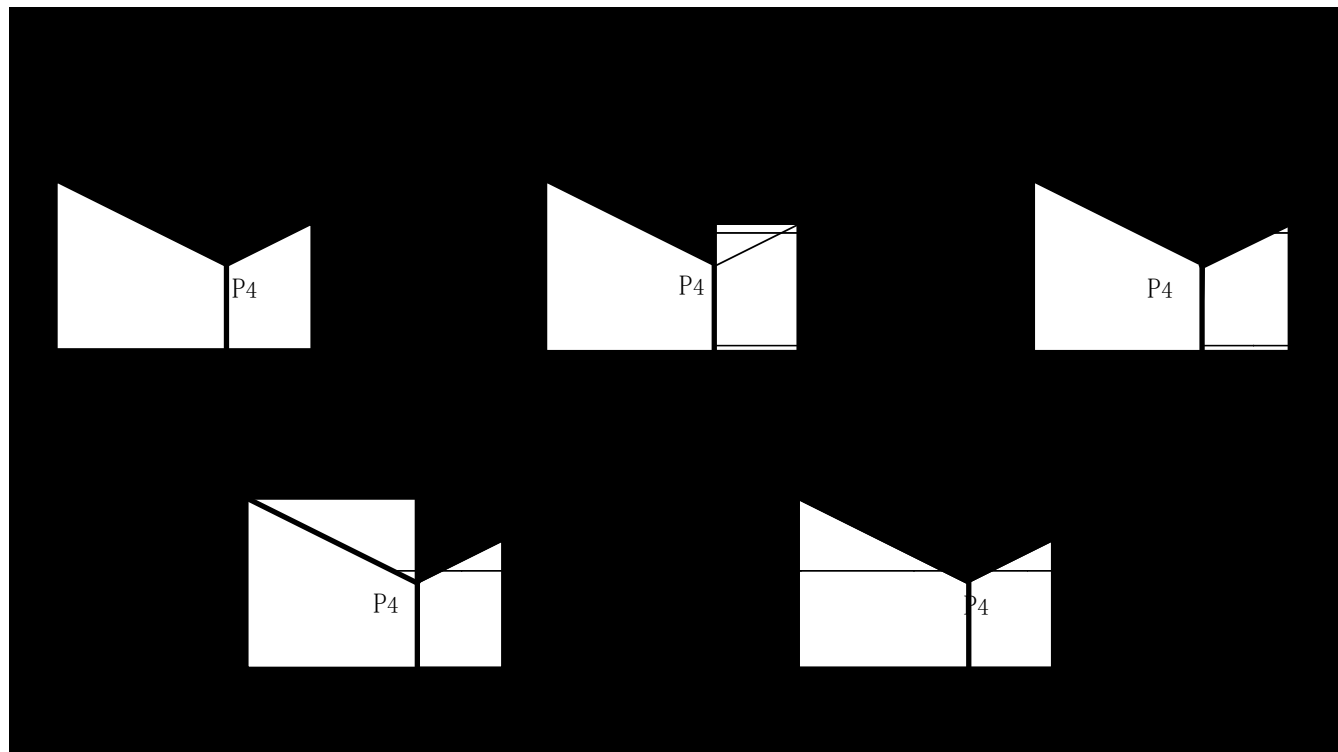
栅栏填充算法

✓ 基本思想

- 将像素中心位于交点和栅栏线之间的全部像素取补

算法
演示1

算法
演示2

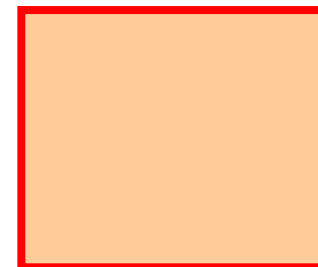


种子填充算法

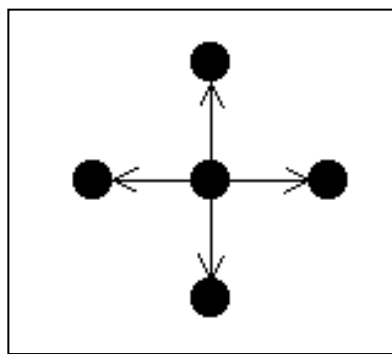
- ✓ **Seed Fill Algorithm**
- ✓ **另外一种思路:**
- ✓ **假设多边形区域内至少有一个像素已知**
- ✓ **由该像素出发找出区域内部的所有像素**

种子填充算法

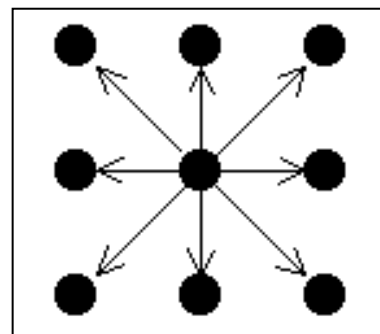
✓ 假定区域采用边界定义法



✓ 区域连通方式:



4-connected

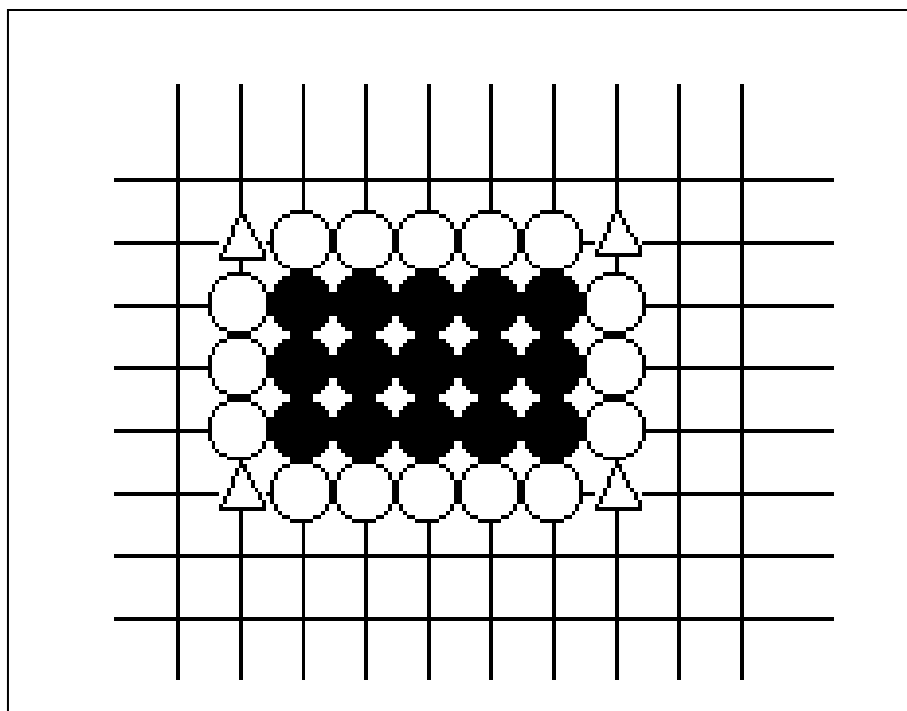


8-connected

种子填充算法

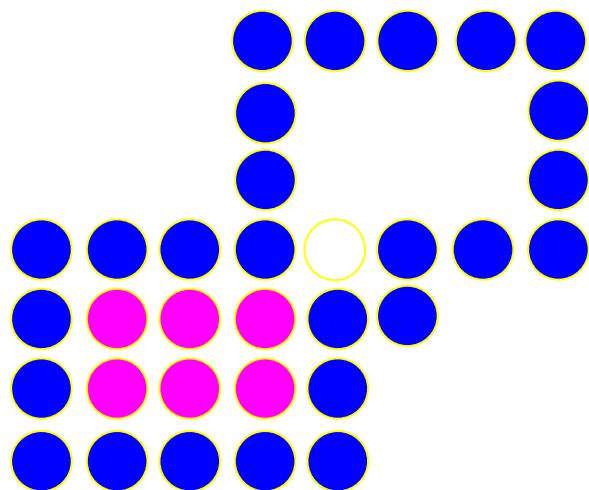
✓ 4连通与8连通区域的区别

- 连通性
- 对边界的要求

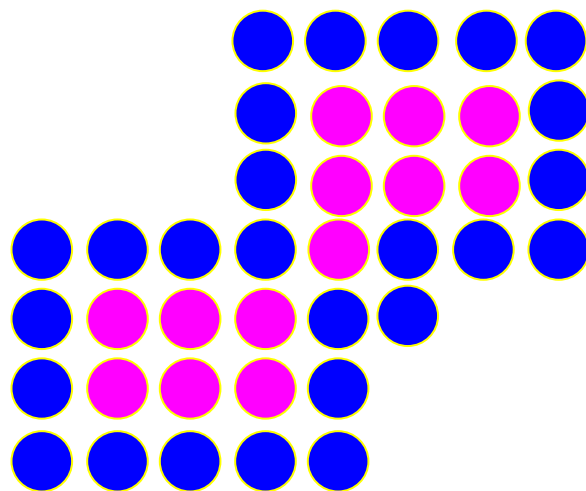


种子填充算法

区域连通方式对填充结果的影响



4连通区域边界填充算法的填充结果



8连通区域边界填充算法的填充结果

简单的种子填充算法

- ✓ 4连通边界算法步骤：
- ✓ 种子像素入栈
- ✓ 当栈非空时，重复以下步骤：
 - 栈顶像素出栈
 - 将出栈像素置成填充色
 - 按左、上、右、下顺序检查与出栈像素相邻的四像素，若其中某像素不在边界上且未被置成填充色，则将其入栈

种子填充算法的递归实现

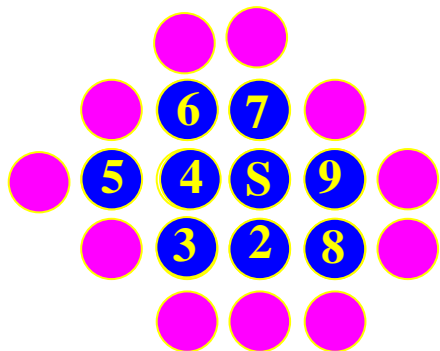
4-connected boundary-fill

```
void BoundaryFill4(int x,int y,int fill,int
    boundary)
{
    int current;
    current = getpixel(x, y);
    if ((current != boundary) && (current != fill))
    {
        putpixel(x, y, fill);
        BoundaryFill4(x+1, y, fill, boundary);
        BoundaryFill4(x-1, y, fill, boundary);
        BoundaryFill4(x, y+1, fill, boundary);
        BoundaryFill4(x, y-1, fill, boundary);
    }
}
```

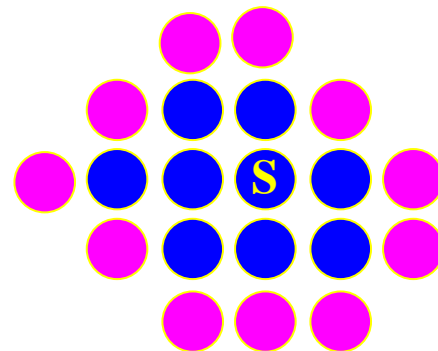
4-connected flood-fill

```
void FloodFill4(int x,int y,int fillColor,int
    oldColor)
{
    int current;
    current = getpixel(x, y);
    if (current == oldColor)
    {
        putpixel(x, y, fillColor);
        BoundaryFill4(x+1, y, fillColor, oldColor);
        BoundaryFill4(x-1, y, fillColor, oldColor);
        BoundaryFill4(x, y+1, fillColor, oldColor);
        BoundaryFill4(x, y-1, fillColor, oldColor);
    }
}
```


简单的种子填充算法



缺点？

[illegible]

扫描线种子填充算法

✧基本思想：

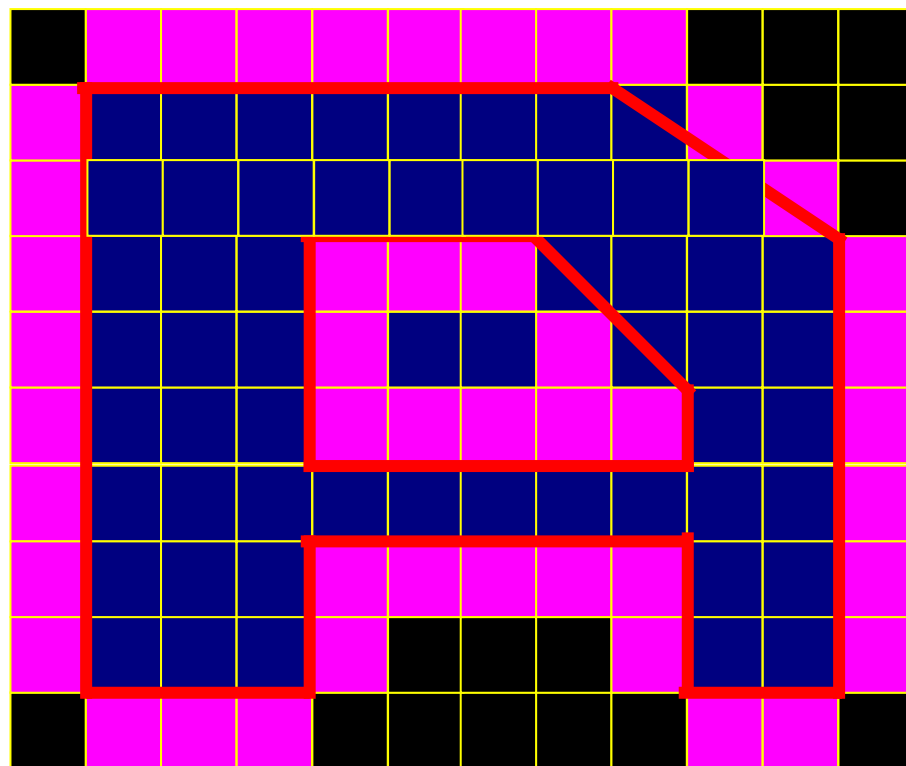
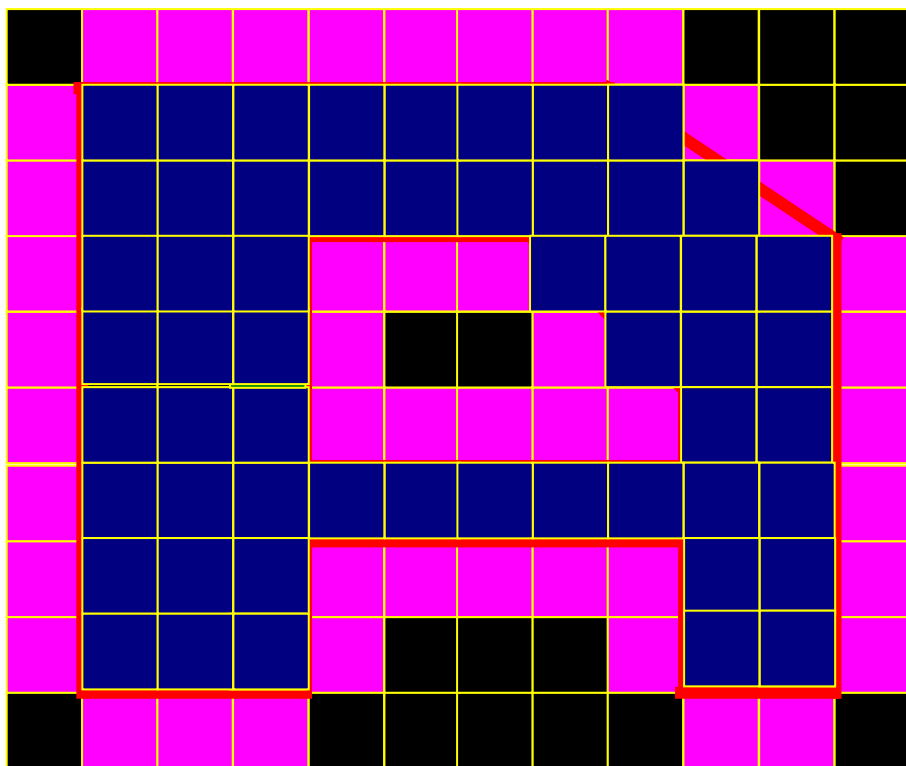
- 利用扫描线的连贯性，每次填充一行像素
- 减少压入堆栈的像素数目

扫描线种子填充算法

- ✓ 种子像素入栈
- ✓ 当栈非空时，重复以下步骤：
 - 栈顶像素出栈
 - 沿扫描线对出栈像素的左右像素进行填充，直到遇到边界像素为止
 - 将上述区间内最左、最右像素记为 x_l 和 x_r
 - 在区间 $[x_l, x_r]$ 中检查与当前扫描线相邻的上下两条扫描线是否全为边界像素、或已填充的像素，若为非边界、未填充的像素，则把每一区间的最右像素取为种子像素入栈

扫描线种子填充算法

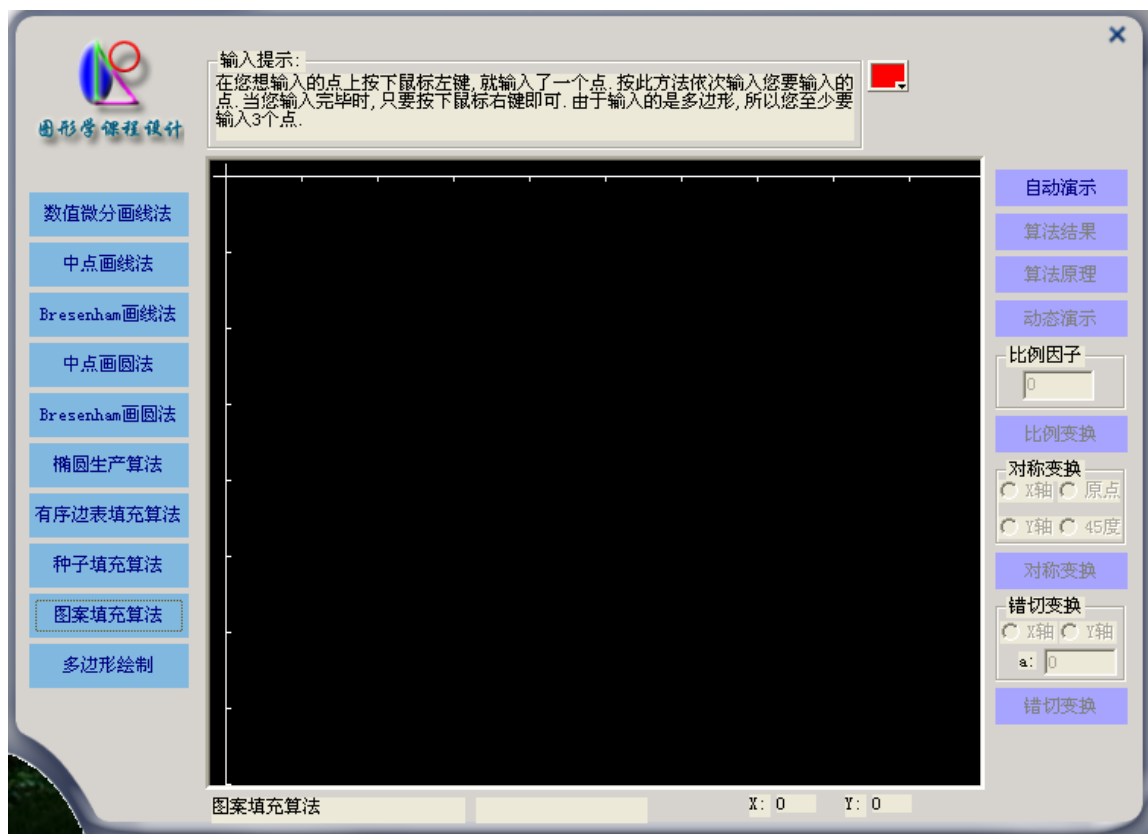
如何填充这个有孔的多边形？



优点是什么？

算法演示

扫描线种子填充算法



程序演示