GIS专业主干课 ：21905001

计算机图形学

Computer Graphics

林伟华

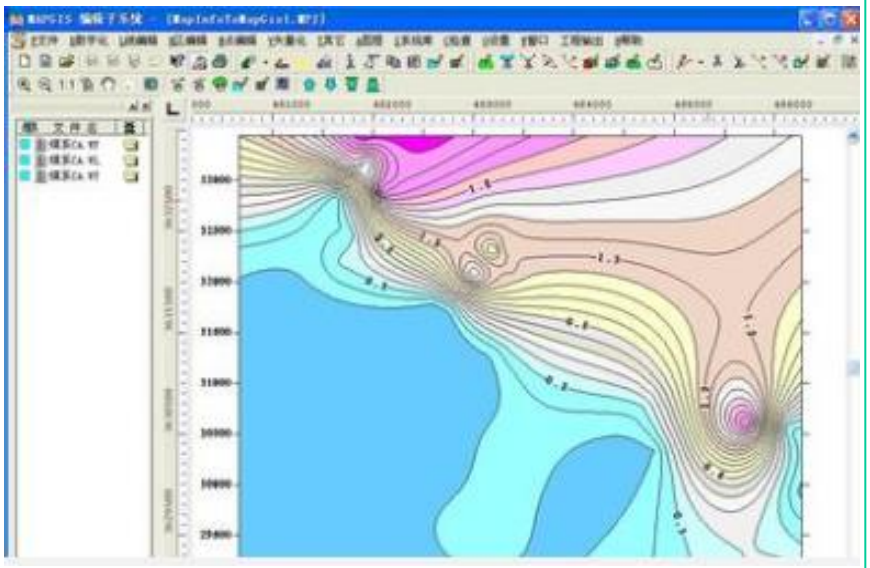
中国地质大学（武汉）信息工程学院

lwhcug@163.com

目录

区域填充GIS应用

- **MapGIS开发**
  - void _RegionFill(MyDC mdc,POINT *xy,int *ne, int regnum,int col=0); //区域填充
- **MapGIS应用**
  - 行政区划
  - 等值线分层……



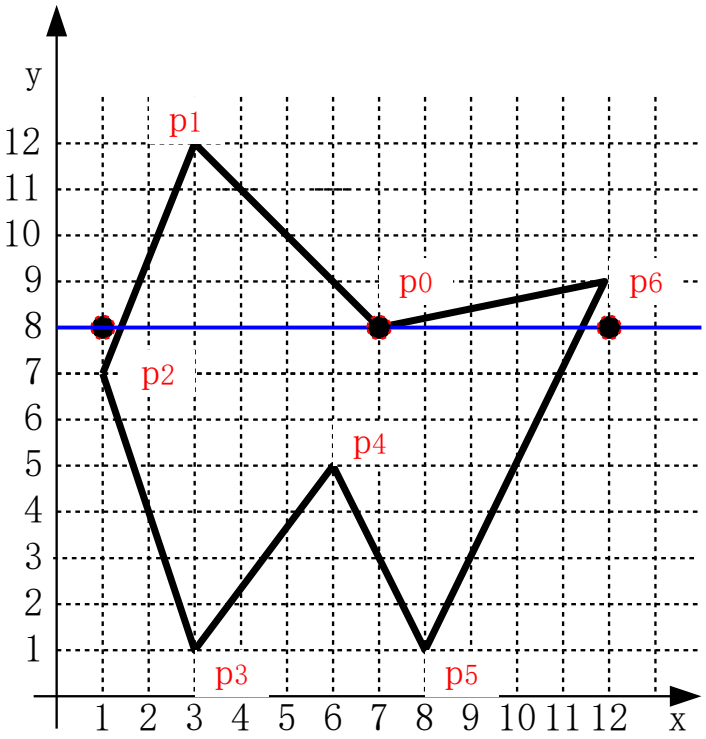
基本概念

- **区域：**指相互连通的一组象素的集合。区域通常由一个封闭的轮廓线来定义，处于一个封闭轮廓线内的所有象素点构成一个区域。
- **区域填充：**将区域内的象素置成新的颜色，新的颜色可以是常数，表示填以某种**颜色**；也可是变量，表示填充的是**图案**。
- **区域填充需解决的问题：**
  - 1）确定需要填充哪些象素；
  - 2）确定用什么颜色。

基本概念

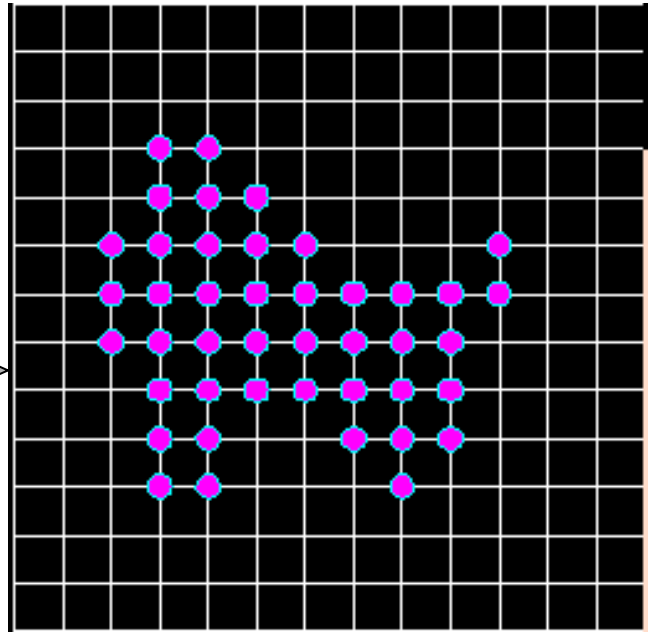
顶点表示：用多边形的顶点序列来刻划多边形

点阵表示：用位于多边形内的象素的集合来刻划多边形



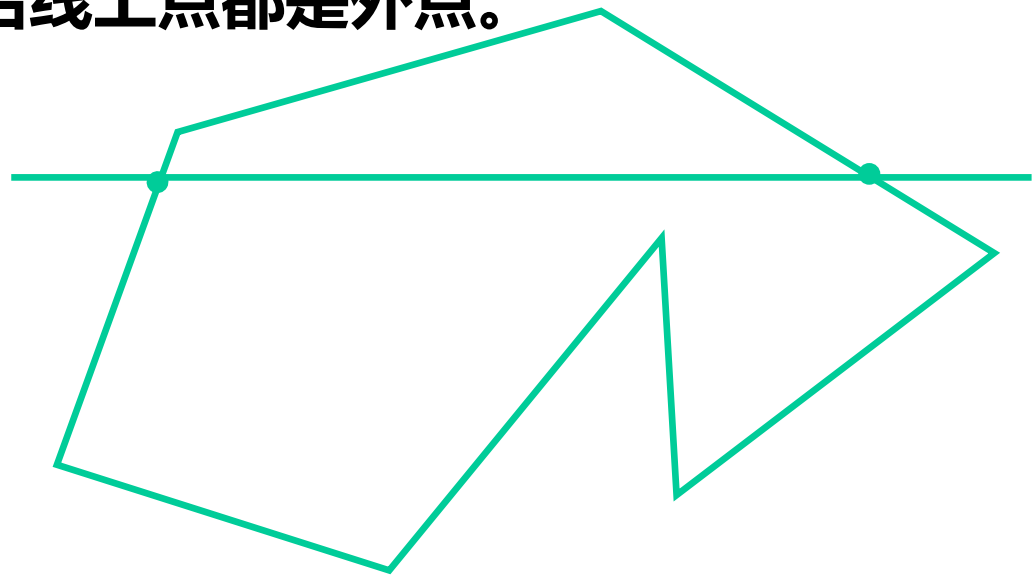
(a)　多边形$P_0P_1P_2P_3P_4P_5P_6P_0$

多边形的扫描转换或多边形的填充



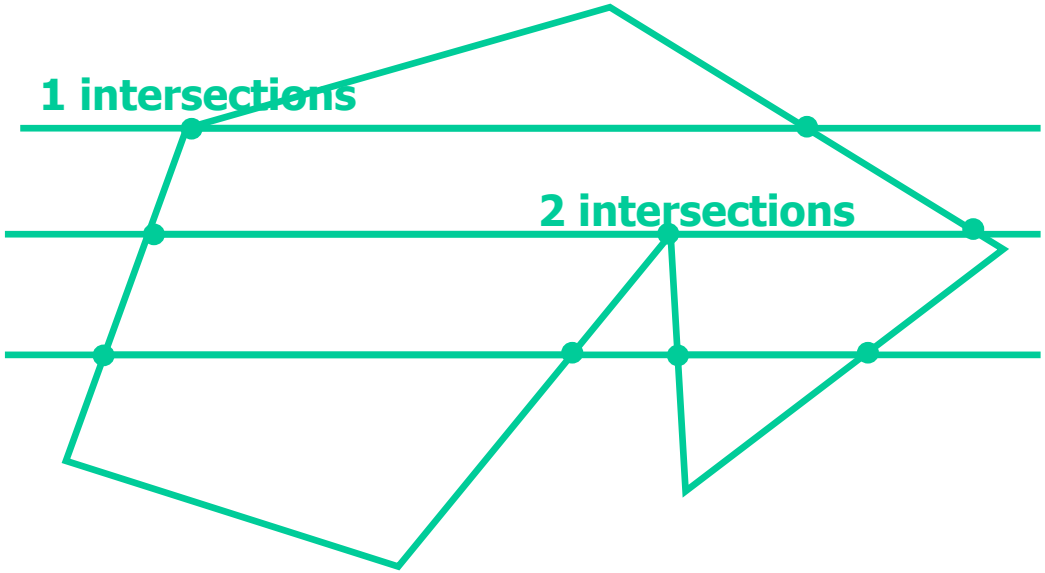
简单扫描线填充算法

- **多边形的内点和外点**
- 内点：填充成固定颜色或图案；
- 外点：不填充；
- 边界点：作为内点还是外点视要求而定。
- **推导：**当射线与多边形相交奇数次后，线上点都是内点，偶数次相交后线上点都是外点。



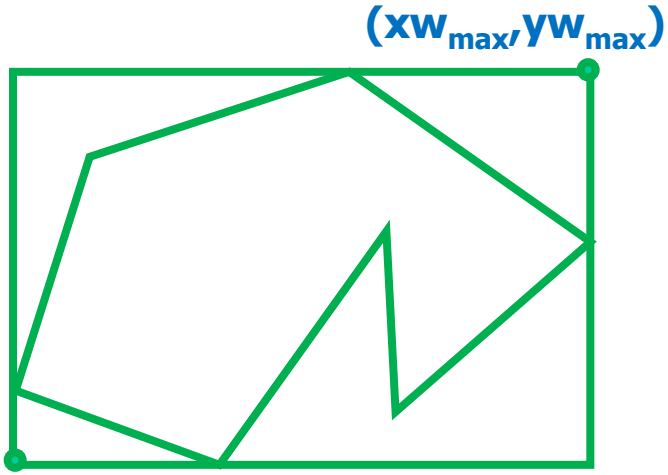
简单扫描线填充算法

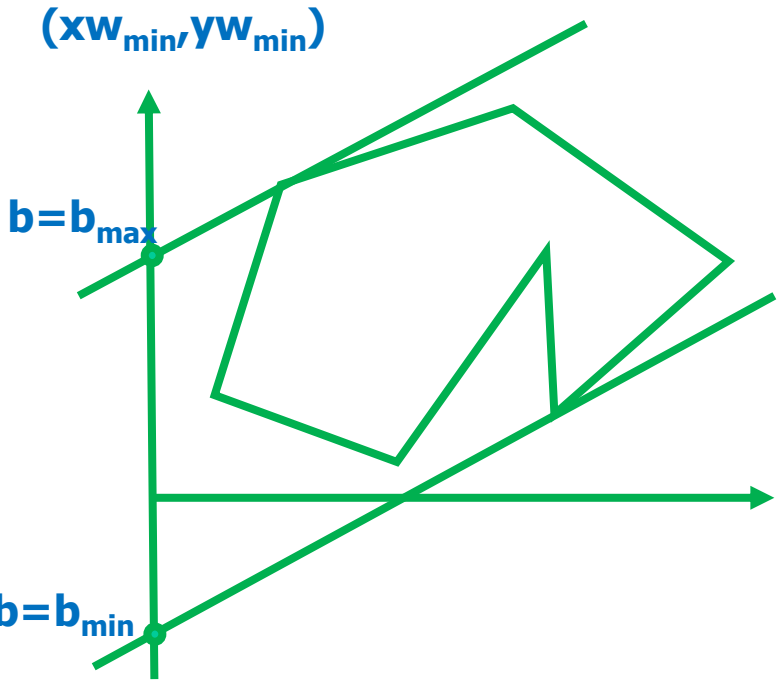
- **特例：顶点是交点**
- **a）共享顶点的两条边分别位于扫描线的两边，交点算一个。**
- **b）共享顶点的两条边都位于扫描线的下边，交点算二个。**
- **c）共享顶点的两条边都位于扫描线的上边，交点算零个。**



简单扫描线填充算法

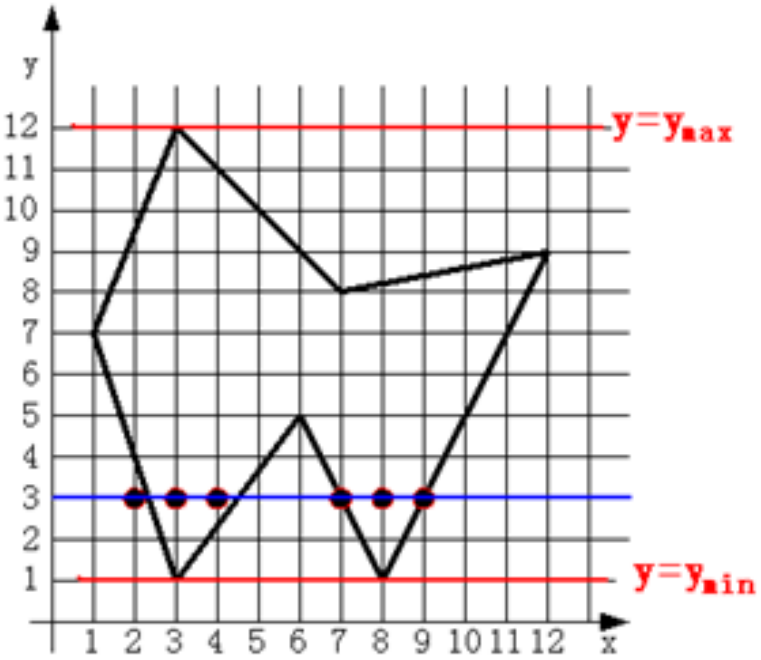
- **算法思路：**
- 1）计算多边形顶点范围
- 2）设置扫描线起始位置b=bmin
- 3）绘制扫描线
  - 计算扫描线与多边形的交点
  - 将交点集进行排序
  - 每一交点对之间部分用填充颜色绘制
- 4）b = b+ △b
- 5）If b<bmax goto 3);
  否则退出程序。

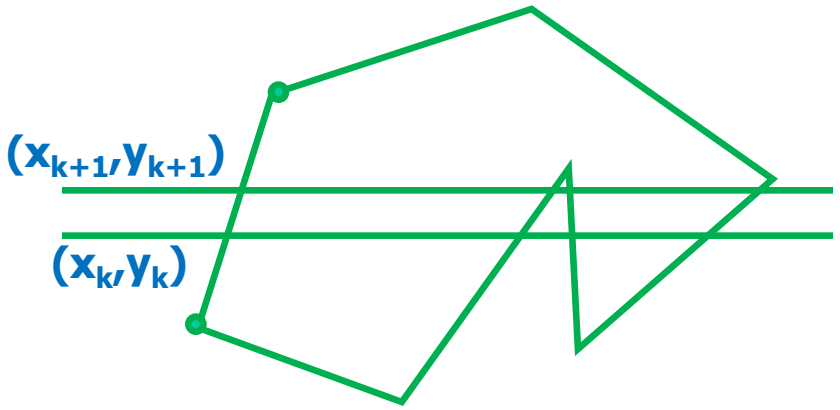




边相关扫描线填充算法

- **改进思路：**
- 每次计算边和扫描线交点，效率很低。
- 当前扫描线与各边的交点顺序，与下一条扫描线与各边的交点顺序很可能相同或类似。





$y_{k+1}=y_k+1$

$x_{k+1}=x_k+1/m$

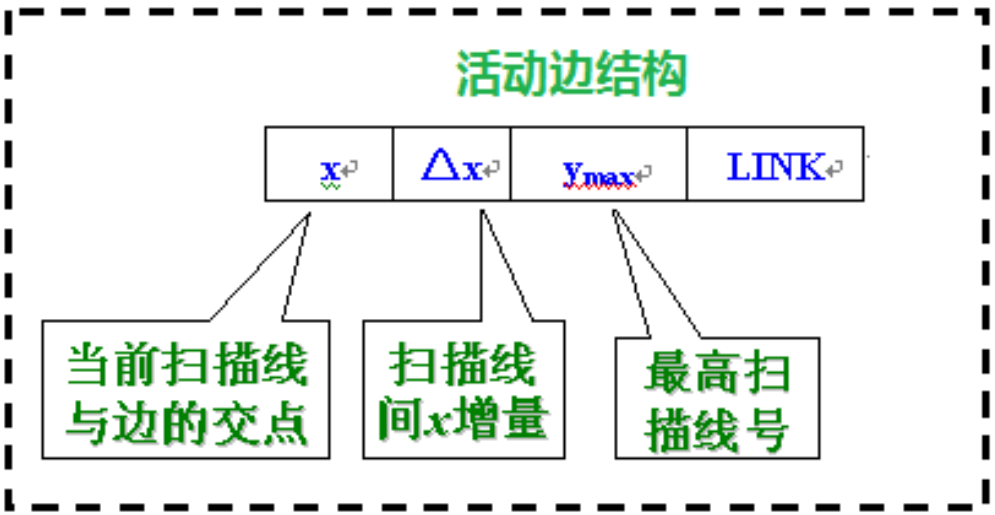
边相关扫描线填充算法

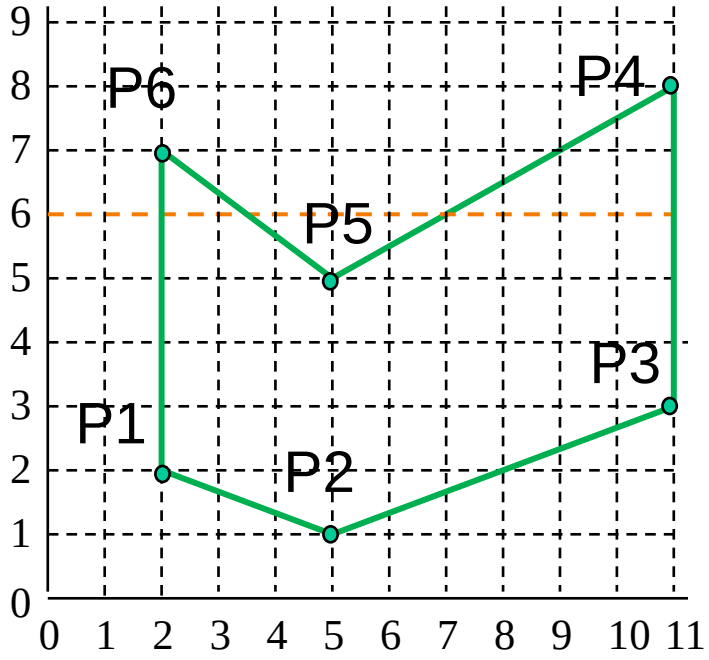
活动边表（AET, Active edge table)：

与当前扫描线相交的边称为活动边（active edge)，把它们按与扫描线交点x坐标递增的顺序存入一个链表中。

如： 扫描线 y=1， 活动边：边p1p2和边p2p3

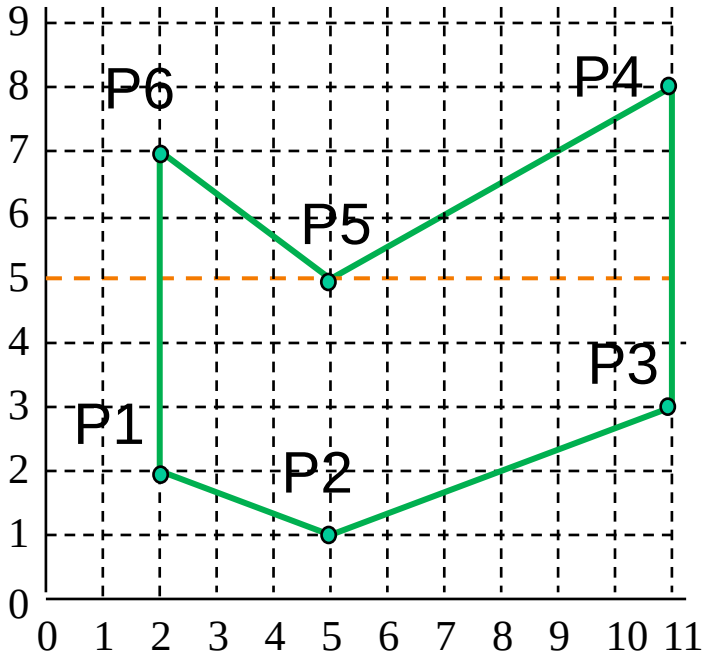
扫描线y=4，活动边：边p1p6和边p3p4

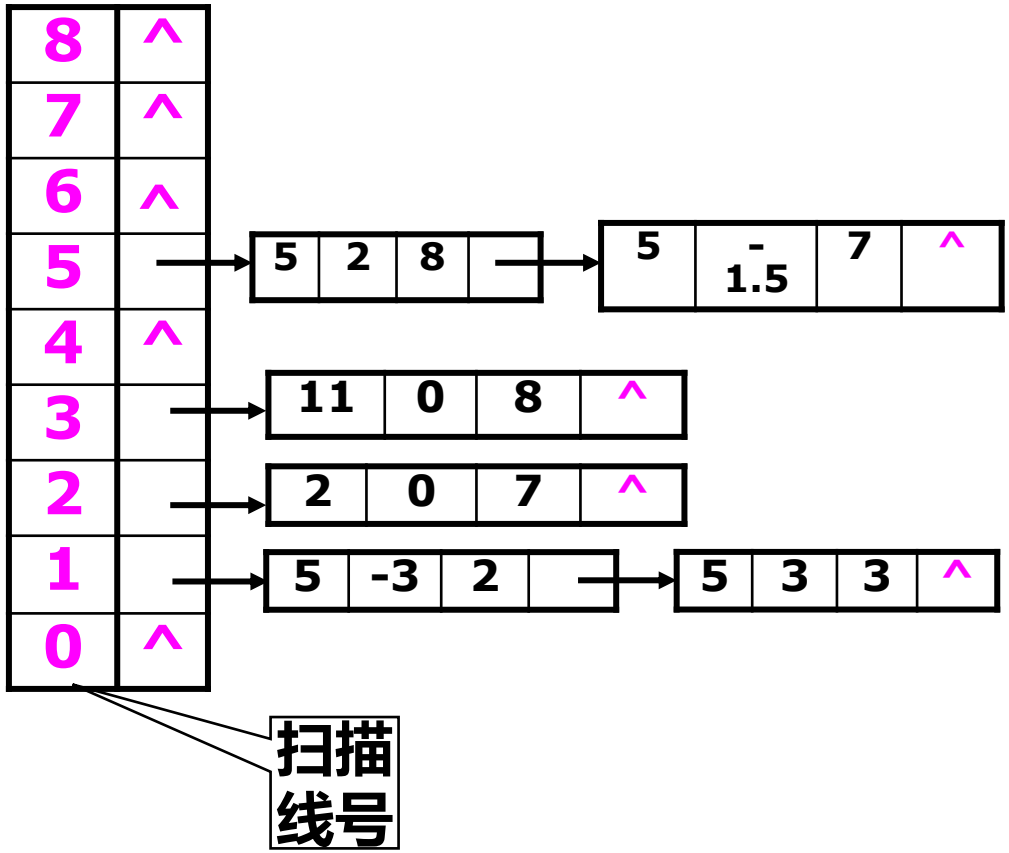




边相关扫描线填充算法

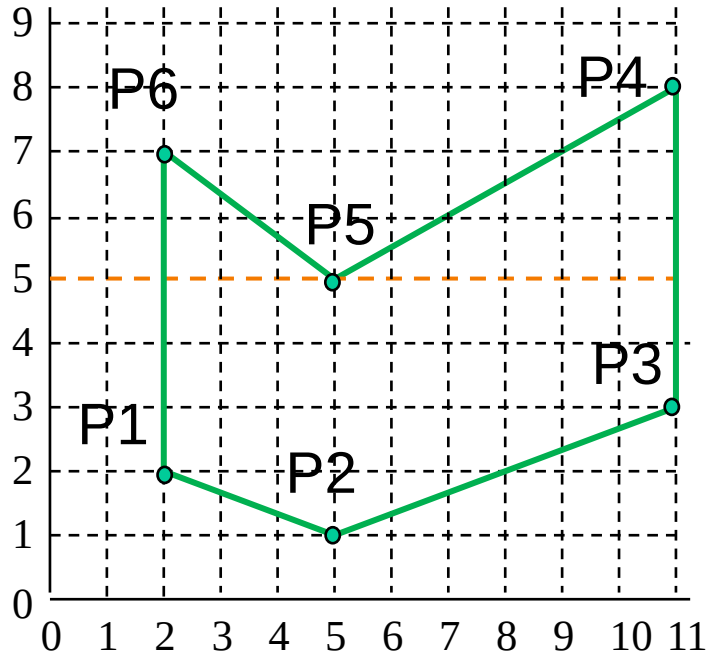
初始活动边表-链表结构





边相关扫描线填充算法

扫描过程中活动边表结构



扫描线	活动边列表	列表排序后	绘制区域
1	5 -3 2	5 -3 2	(5, 5)
	5 3 3	5 3 3	
2	2 0 7	2 0 7	(2, 8)
	8 3 3	8 3 3	
3	2 0 7	2 0 7	(2, 11)
	11 0 8	11 0 8	
4	2 0 7	2 0 7	(2, 11)
	11 0 8	11 0 8	
5	2 0 7	2 0 7	(2, 5) (5, 11)
	5 -1.5 7	5 -1.5 7	
	5 2 8	5 2 8	
	11 0 8	11 0 8	
6	2 0 7	2 0 7	(2, 3.5) (7, 11)
	3.5 -1.5 7	3.5 -1.5 7	
	7 2 8	7 2 8	
	11 0 8	11 0 8	
……			

边相关扫描线填充算法

算法思路：

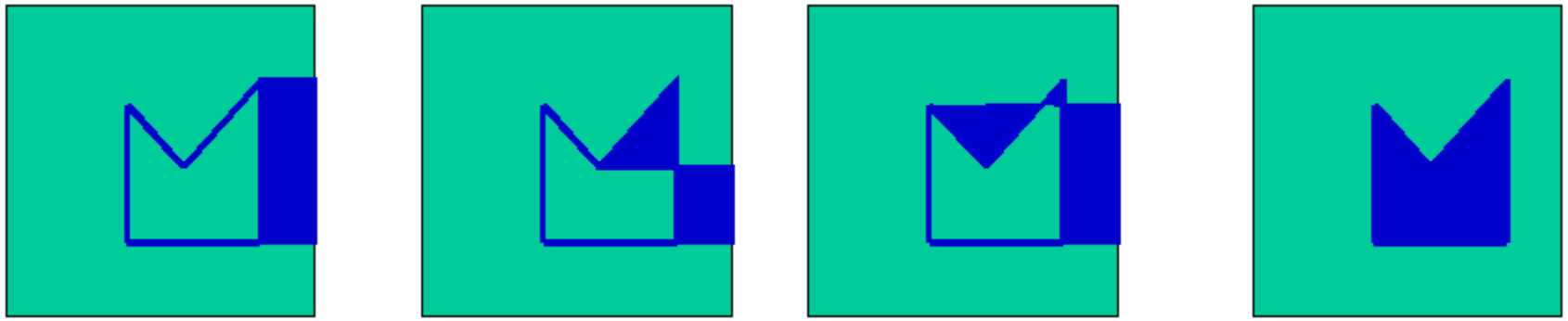
1）存储多边形边在经排序后（XY排序）活动边表链表中，每边包含元组信息（交点x坐标，斜率倒数，Ymax，Link）。

2）扫描线从多边形底到上扫描：

- 基于扫描线插入和修改活动边表链表
- 交点两两配对填充
- 删除y=ymax边

边填充算法

- **基本思想：**对于每条多边形与每一条扫描线的交点，将该扫描线上交点右方所有的象素取补。对多边形的每一条边作此处理，多边形各边的处理顺序随意。



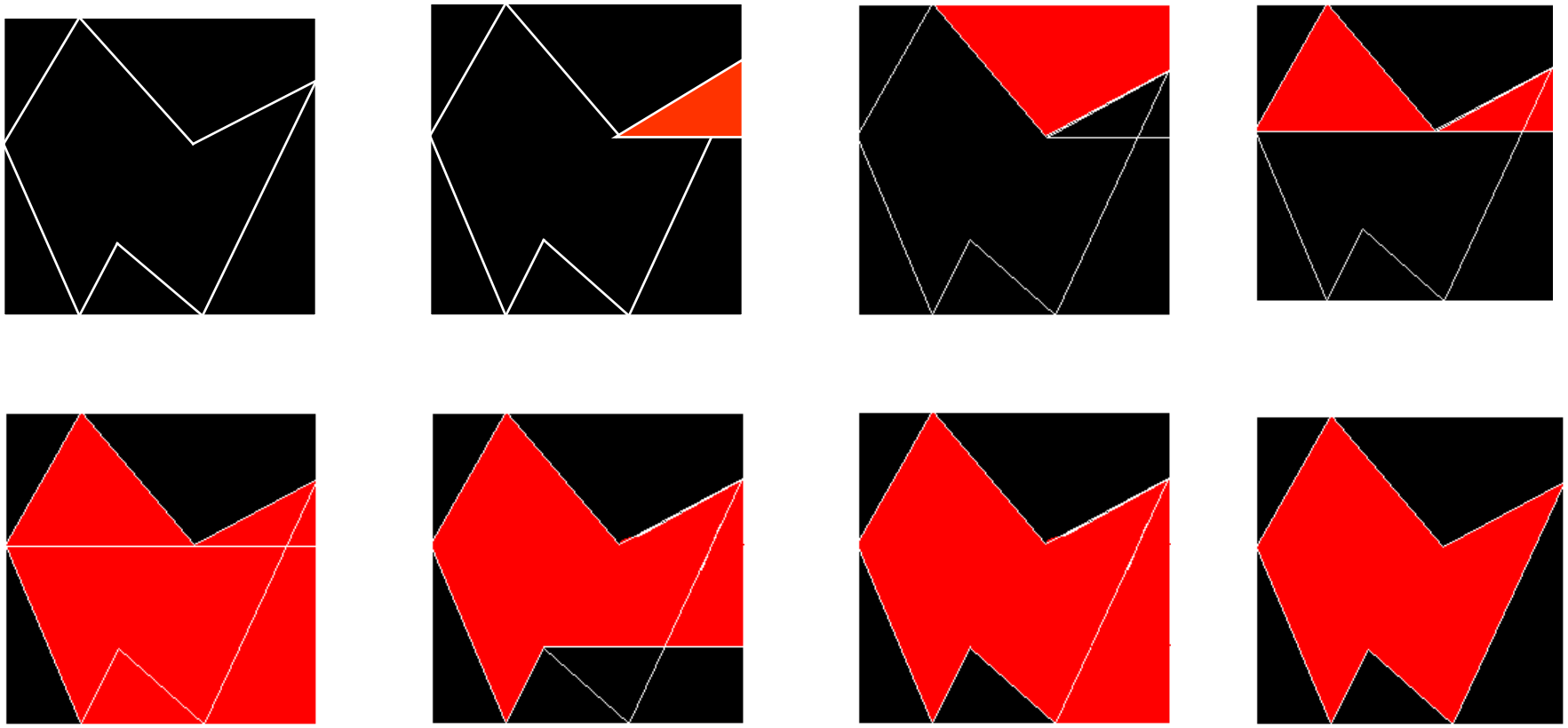
优点：算法简单

缺点：1）每个象素可能被访问多次；

2）输入/输出量大。

边填充算法

- 再例：

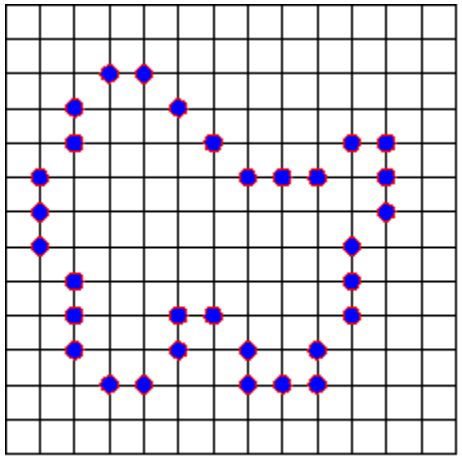


简单种子填充算法

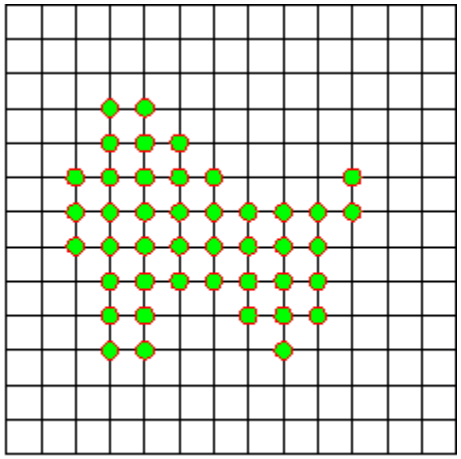
1、原理：假设在多边形区域内部有一象素已知，由此出发找到区域内的所有象素。

2、区域分类：

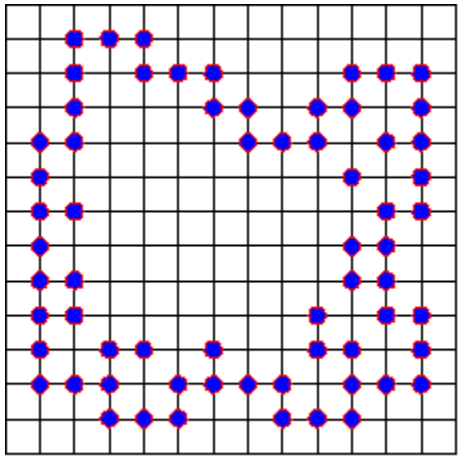
- 四向连通：从区域上任意一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意象素。
- 八向连通：区域内的每个象素，可以通过左、右、上、下、左上、右上、左下、右下这八个方向的移动的组合来到达.



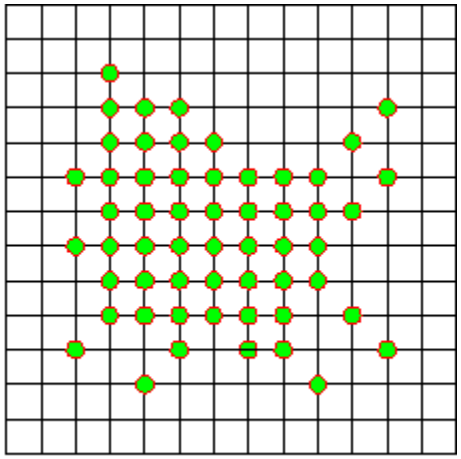
(a)以边界表示的4-连通区域



(b)以内点表示的4-连通区域



(c)以边界表示的8-连通区域

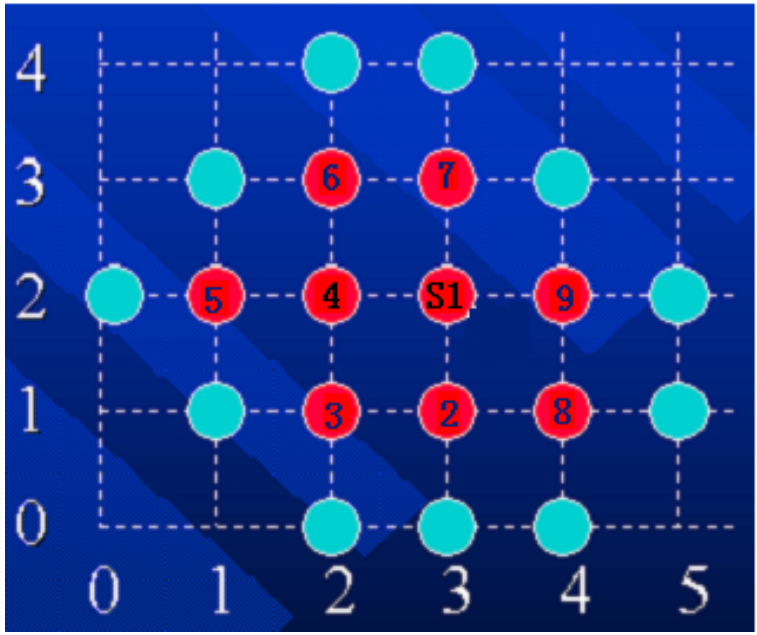


(d)以内点表示的8-连通区域

简单种子填充算法

算法思路：

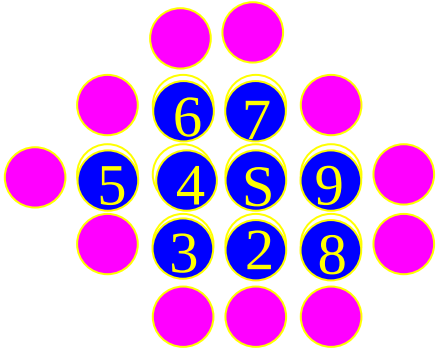
- 1）初始化：种子像素入栈，当栈非空时，重复2-4的步骤。
- 2）栈顶像素出栈。
- 3） 将出栈像素置为填充颜色。
- 4） 按右、上、左、下顺序依次检查与出栈像素相邻的四个像素，若其中某个像素不在边界上且未填充，则该像素入栈。
- 5）当堆栈为空时，算法终止。

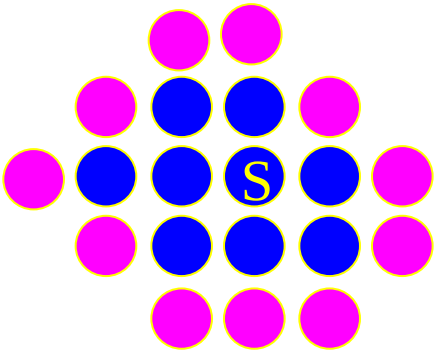


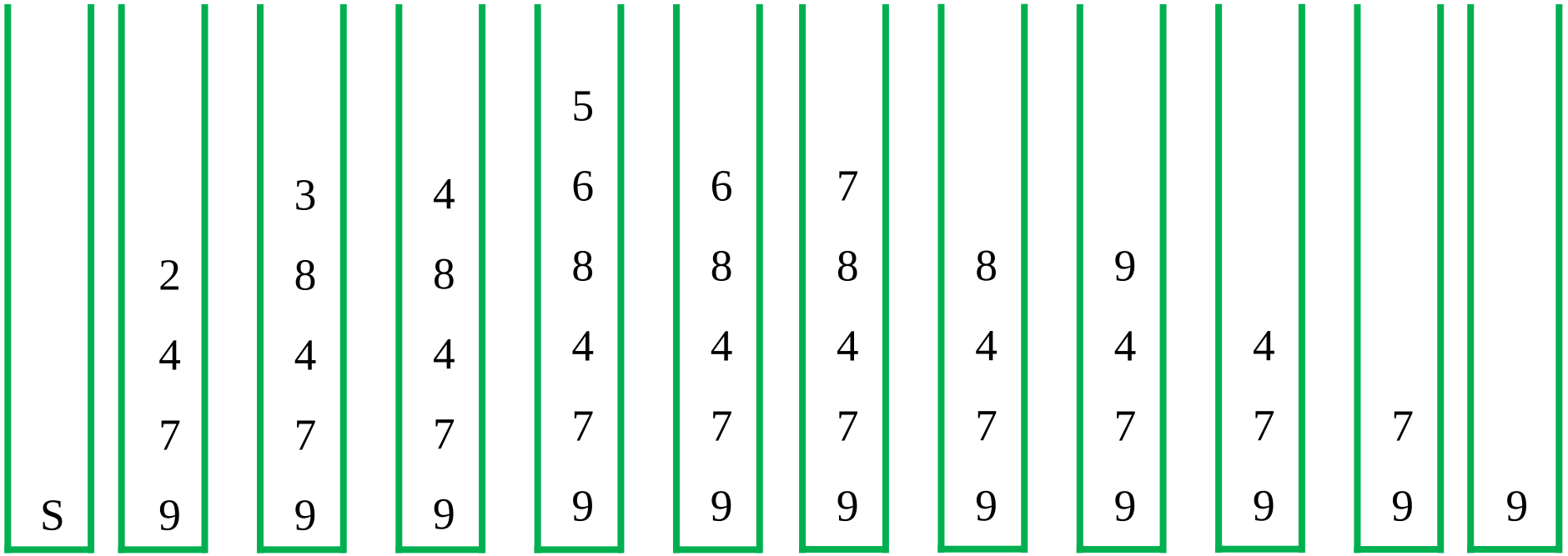
则各元素入栈顺序：
S1，9，7，4，2，8，3，4
6，5

简单种子填充算法

简单种子填充算法-演示：







简单种子填充算法

简单种子填充—递归算法

```
void BoundaryFill4(int x,int y,int boundarycolor,int newcolor)
{   int color;
        if(color!=newcolor && color!=boundarycolor)
        {       drawpixel(x,y,newcolor);
                BoundaryFill4 (x,y+1, boundarycolor,newcolor);
                BoundaryFill4 (x,y-1, boundarycolor,newcolor);
                BoundaryFill4 (x-1,y, boundarycolor,newcolor);
                BoundaryFill4 (x+1,y, boundarycolor,newcolor);
        }
}
void FloodFill4(int x,int y,int oldcolor,int newcolor)
{   if(getpixel(x,y)==oldcolor)   //属于区域内点oldcolor
        {       drawpixel(x,y,newcolor);
                FloodFill4(x,y+1,oldcolor,newcolor);
                FloodFill4(x,y-1,oldcolor,newcolor);
                FloodFill4(x-1,y,oldcolor,newcolor);
                FloodFill4(x+1,y,oldcolor,newcolor);
        }
}
```

简单种子填充算法

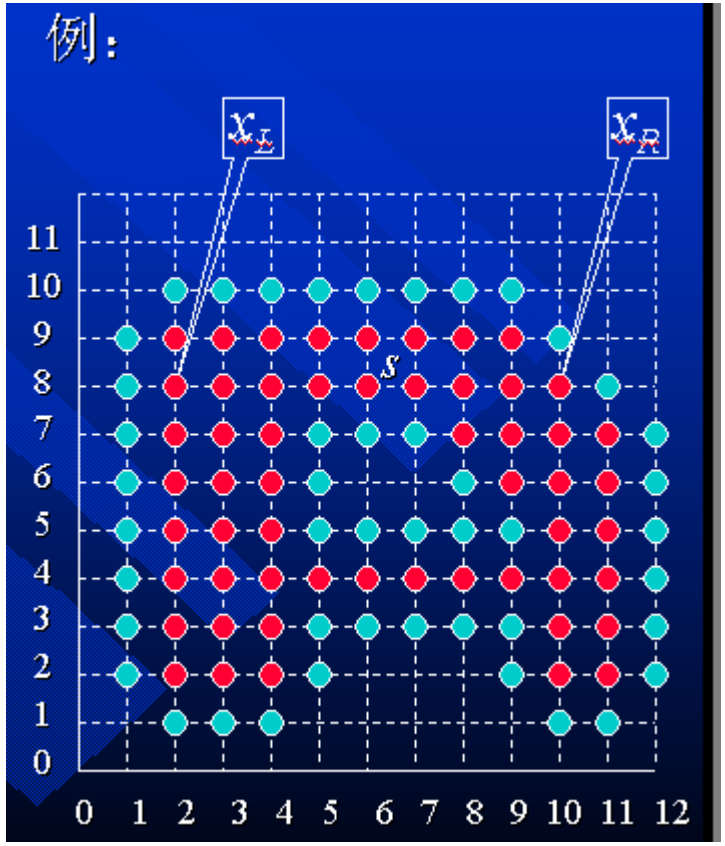
简单种子填充—非递归算法

```
void boundaryfill4(int x, int y, int fill, int boundary)
{
    push(x,y);
    while(stack not empty)
    {
        pop(x,y);
        setpixel(x,y,fill);
        If((getpixel(x+1,y)!=fill)&&(getppixel(x+1,y)!= boundary)
            push(x+1,y);
         If((getpixel(x,y+1)!=fill)&&(getppixel(x,y+1)!= boundary)
            push(x,y+1);
         If((getpixel(x-1,y)!=fill)&&(getppixel(x-1,y)!= boundary)
            push(x-1,y);
         If((getpixel(x,y-1)!=fill)&&(getppixel(x,y-1)!= boundary)
            push(x,y-1);
     }
}
```

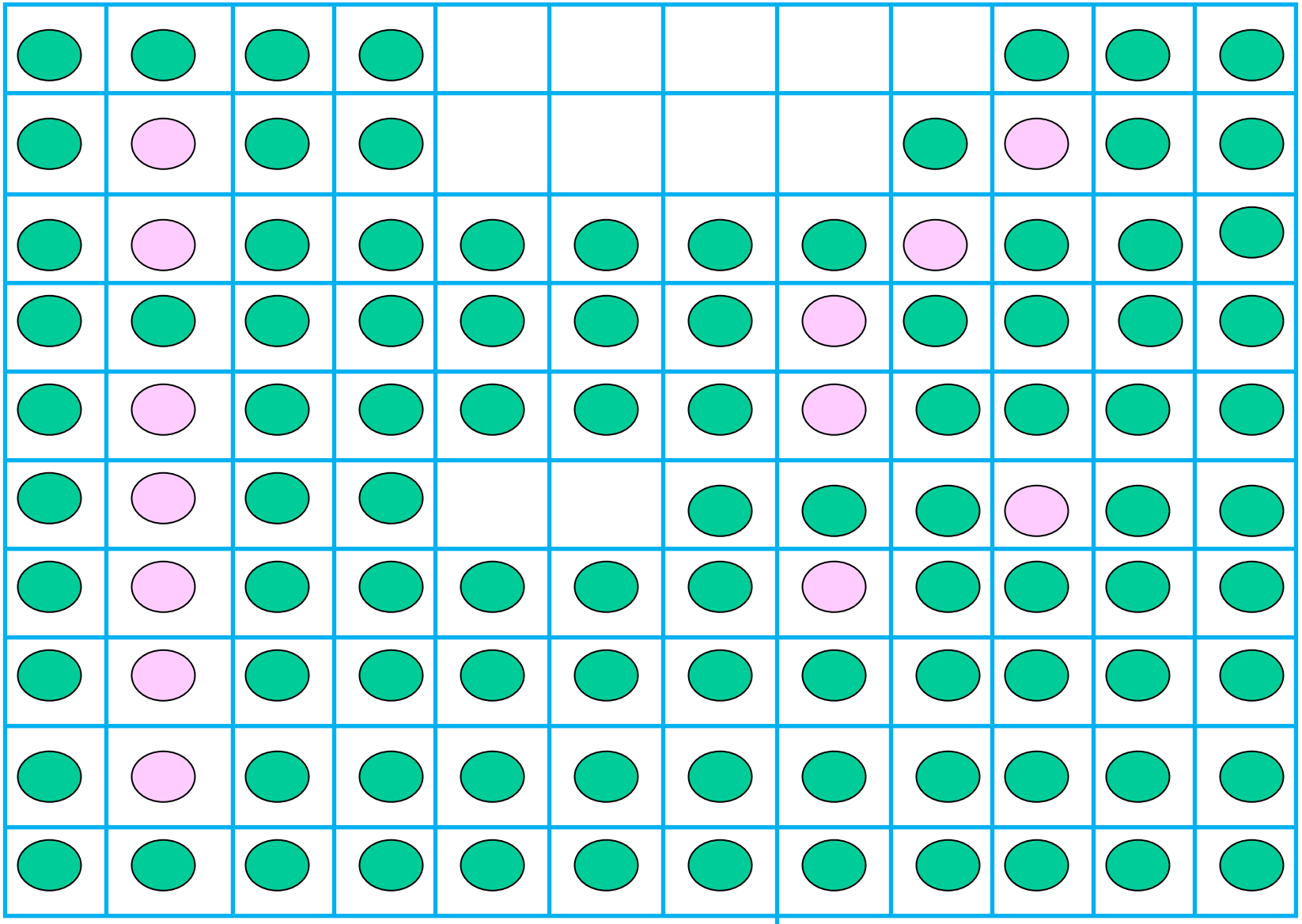
扫描种子填充算法

- **改进：** 为减少像素重复入栈，限定任一扫描线与多边形相交区间，只取一个种子像素

1. 初始化：种子像素入栈，当栈非空时，重复2-6的步骤。
2. 栈顶像素出栈。
3. 沿扫描线对种子像素的左右像素进行填充，直至边界，从而填满该区间。
4. 区间内最左和最右像素分别记为 x_L 和 x_R 。
5. 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或已经填充，若存在非边界、未填充的像素，将检查区间的最左像素作种子入栈。
6. 当堆栈为空时，算法终止。



扫描种子填充算法



直线线宽

- 线刷子：垂直刷子、水平刷子

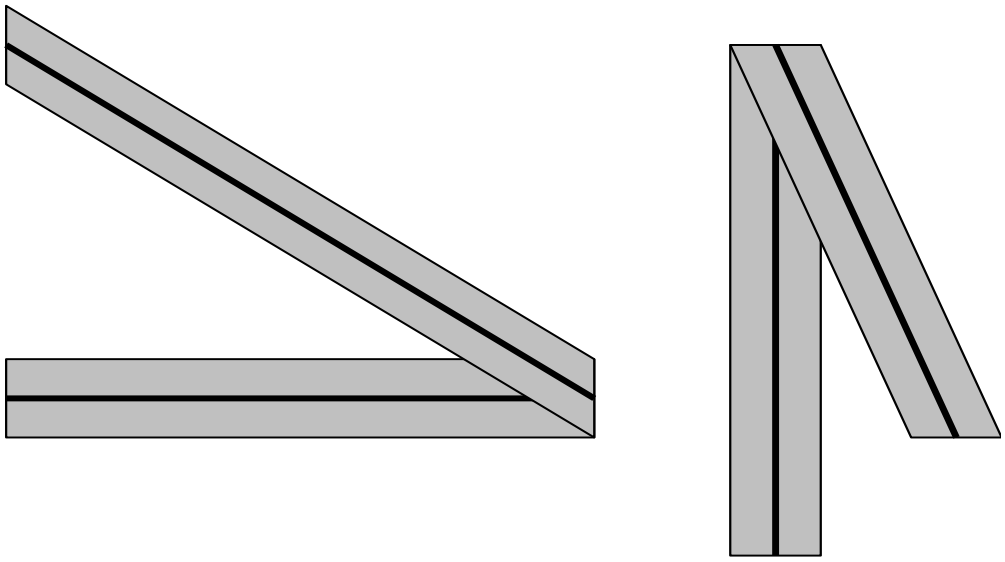
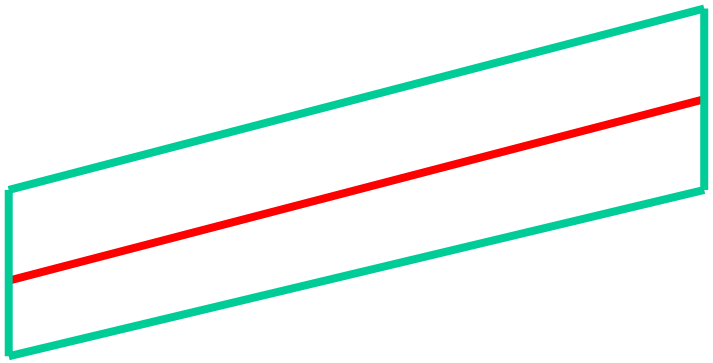


图5-39　线刷子

直线线宽

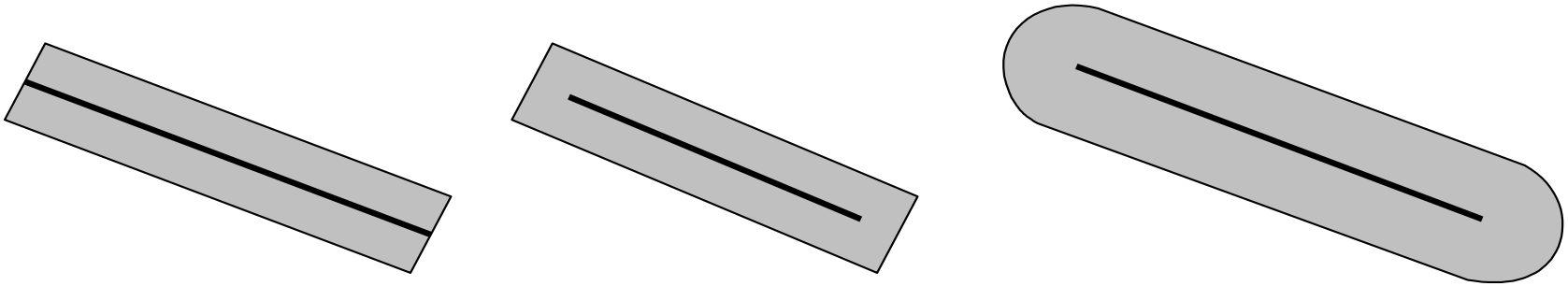
- 实现思路：
  - 直线斜率在【-1，1】之间，把刷子置成垂直方向，让刷子中心往直线的另一端移动即可。
  - 若直线斜率不在【-1，1】之间，把刷子置成水平方向即可。
- 实现方法：
  - 修改直线的扫描转换算法即可。
  - 例：当直线斜率在【-1，1】之间时，把每步迭代的点的上下方半线宽之内的象素全部置为直线颜色即可。



直线线宽

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

解决：添加“线帽（line cap）”



(a)方帽　(b)突方帽　(c)圆帽

图5-40　线“帽子”

直线线宽

- 缺口处理：
  - 当较接近水平的线与较接近垂直的线汇合时，汇合处外角有缺口。

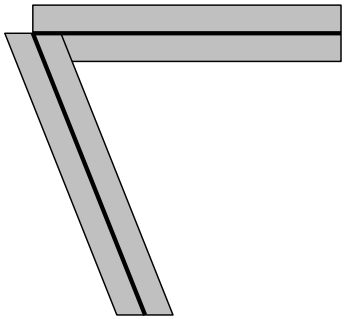
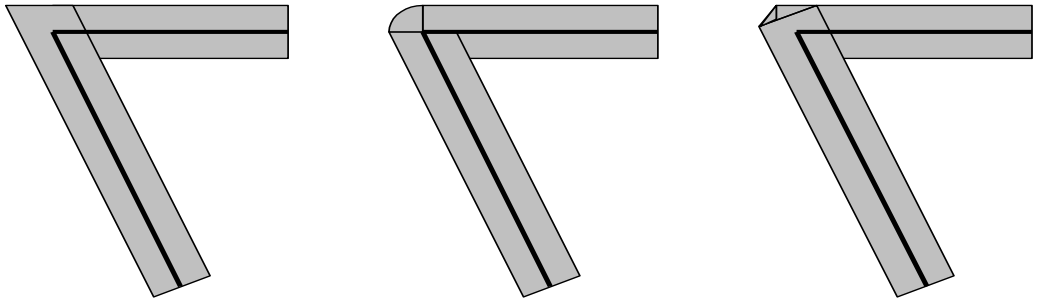


图5-41　线刷子产生的缺口

  - 解决：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5-42　线刷子产生的缺口

直线线宽

- **方刷子：**把边宽为指定线宽的正方形的中心对准单象素宽的线条上的各个象素，并沿直线作平行移动即可。

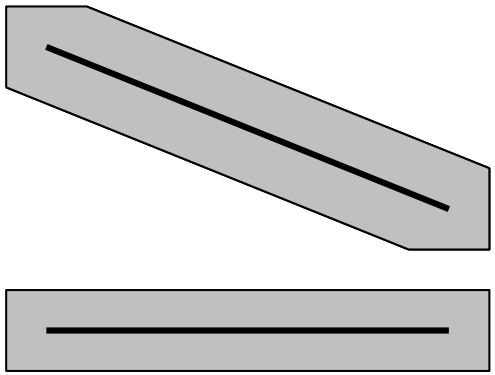


图5-43　方刷子

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

直线线宽

- **具体实现：采用区域填充的算法**
- ① **建立空表**：为每条扫描线建一个表，存放该扫描线与线条的相交区间左右端点位置；
- ② **确定每条扫描线左右端点**：用该方刷子方形与各扫描线的相交区间端点坐标去更新原表内端点数据，得到线条各角点；
- ③ **确定填充多边形**：用直线段把相邻角点连接起来，最后调用多边形填充算法把所得的四边形进行着色。

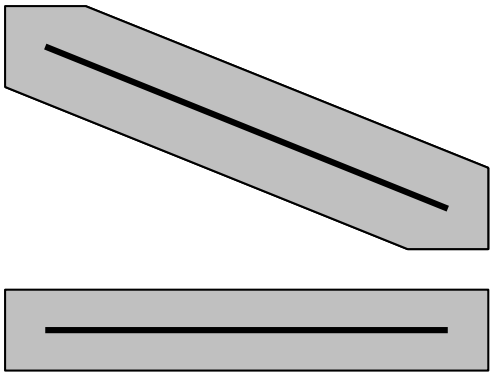


图5-43　方刷子

常见线条

- **常用线条：**
  - 点线：表示立体线框中可见的轮廓线；
  - 虚线：表示立体线框中不可见的轮廓线；
  - 点划线：用于表示中心线。
- **具体画法：**
  - 虚线：可通过在实线段之间插入与实线段等长的空白段来显示；
  - 划线：由用户指定各段划线的长度和空白段的长度；
  - 点线：可通过生成很短的划线和等于和大于划线大小的空白段来显示；
  - 点划线：点与等长实线段的交错出现。

常见线条

- **实现方法：**
  - 实心段和中间空白段的长度（象素数目）可用象素模板(pixel mask)指定
    - **存在问题**：如何保持任何方向的划线长度近似地相等
    - **解决**：可根据线的斜率来调整实心段和中间空白段的象素数目。

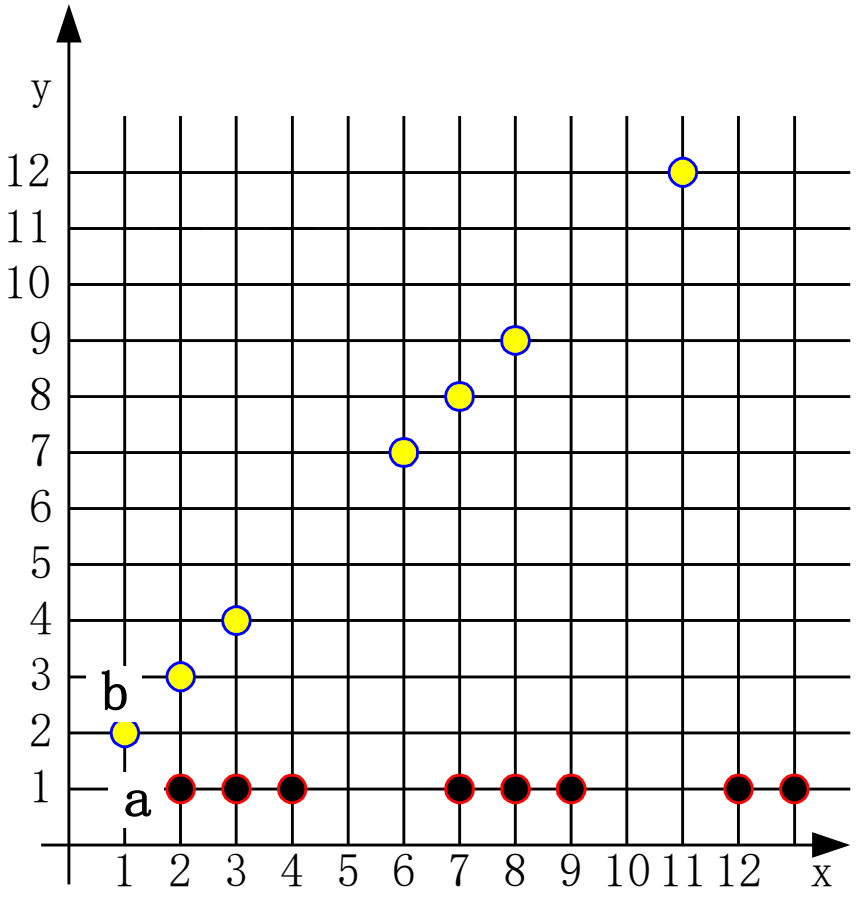


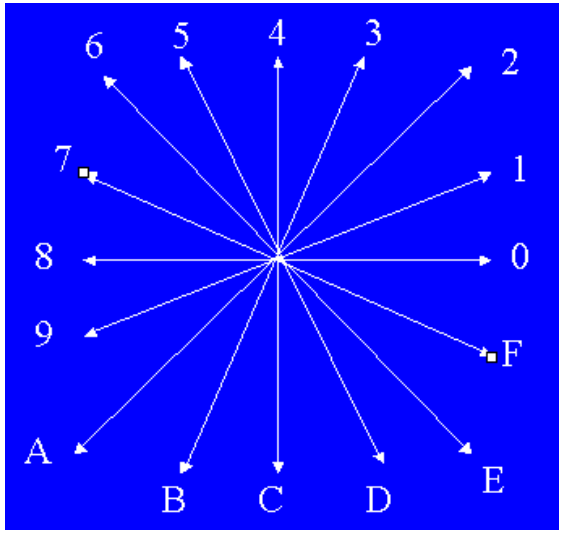
图5-38　相同数目象素显示的不等长划线

字符

- **字符库：**
  - 字符库中存储了每个字符的形状信息。为了在终端显示器和绘图仪上输出字符。
- **矢量型字符库：**
  - 采用矢量代码序列表示字符的各个笔画，适用笔式绘图仪绘制字符。
- **点阵型字符库：**
  - 为每个字符产生一个字符掩膜，即表示该字符的象素图案的一个点阵，适用于终端显示器显示字符。

矢量字符

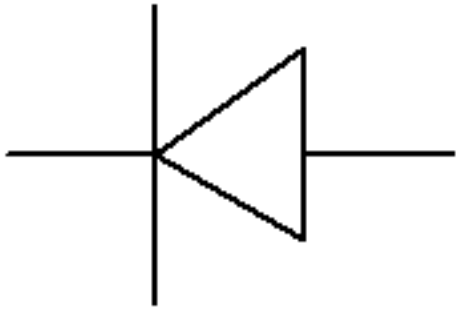
- **AutoCAD中所采用的定义：**
  - **<形编号>，<字节数>，<形名称>**
  - **<字节1>，<字节2>，…… 0**



① 标志位：带有前缀0的字节是十六进制，无前缀0的字节是 十六进制。

② 高四位表示矢量的长度，低四位表示矢量方向。

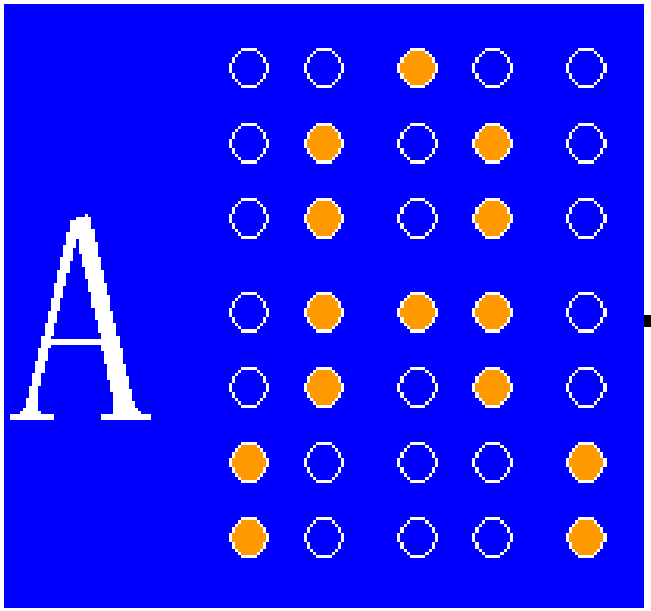
- 例1：下图中所示的二极管符号的形定义为：



- 133,11,DIODE
- 040,044,04C,042,04C,040,
- 048,04C,046,04C,0

点阵字符

- **西文字符：**至少用5*7的点阵描述,存储需35位，4个多字节
- **汉文字符：**至少用16*16的点阵描述,存储需256位，32个字节特殊字符：按需要确定。



问题：信息量大，

解决：压缩，矢量。

二进制数表示的矩阵

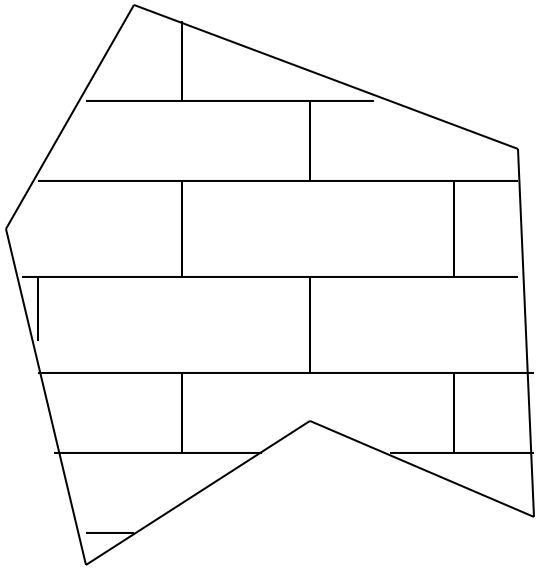
↓

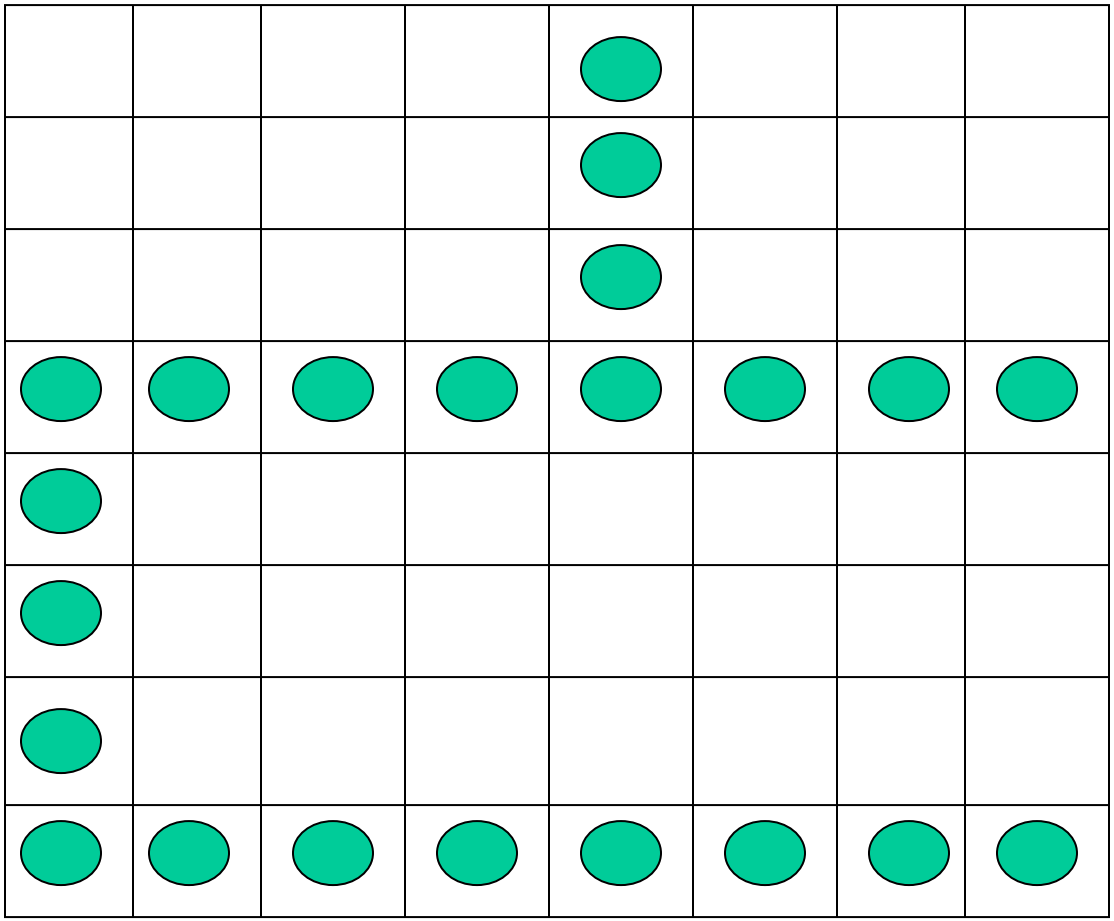
编码（ASCII）

图案填充

```
unsigned char maskcode[8]=
{ 0xFF,0x80,0x80,0x80,0xFF,0x08,0x08,0x08};
unsigned char maskbit[8]=
{ 0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
… …

 … …
i=fmod(x,8); j=fmod(y,8);
If(maskcode[j]&maskbit[I])
putpixel(x,y,value);
```





思考

1、阐述简单扫描线填充和与边相关扫描线填充算法区别。

2、扫描线种子填充相对简单种子填充改进的地方是什么？

谢　谢！