计算机图形学

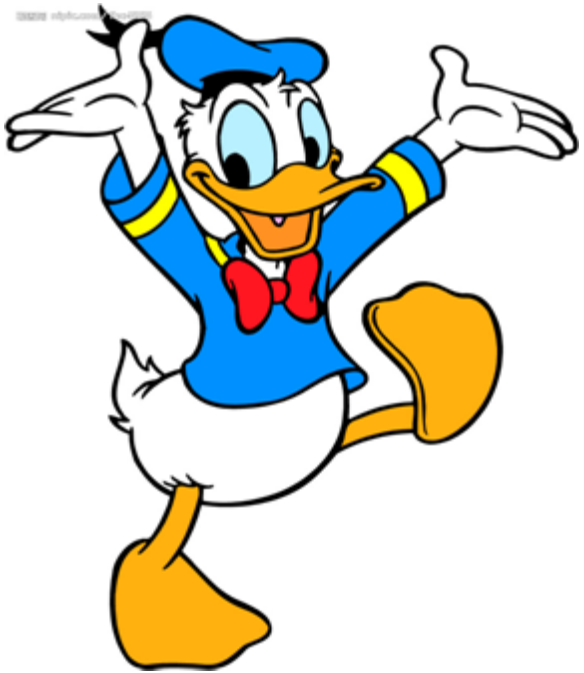
第二章：光栅图形学算法

光栅图形学算法的研究内容

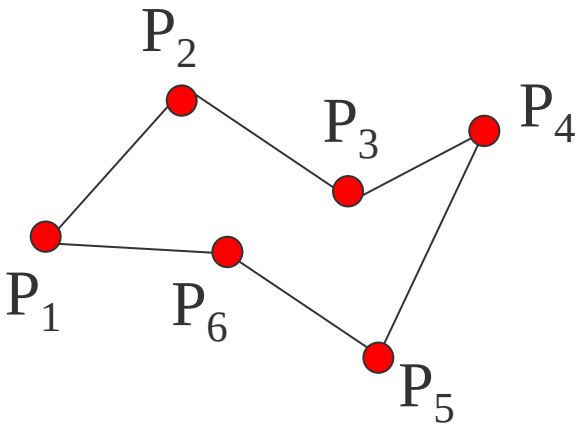
- 直线段的扫描转换算法
- 多边形的扫描转换与区域填充算法
- 裁剪算法
- 反走样算法
- 消隐算法

一、多边形的扫描转换

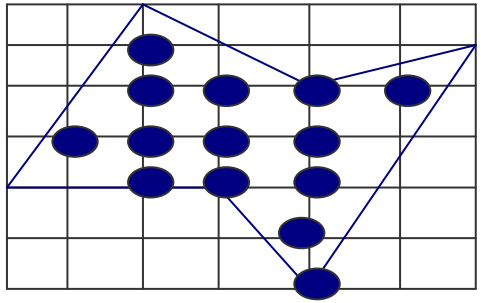
多边形的扫描转换和区域填充这个问题是怎么样在离散的像素集上表示一个连续的二维图形



多边形有两种重要的表示方法：**顶点**表示和**点阵**表示



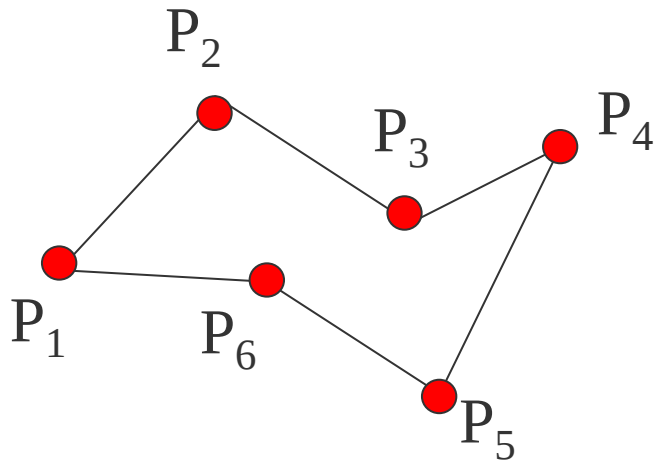
顶点表示



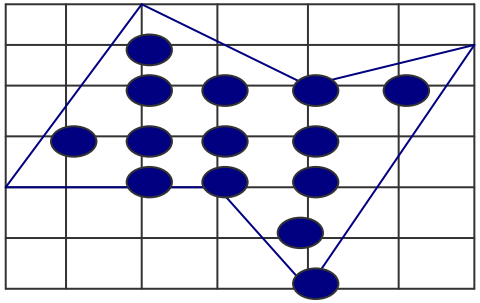
点阵表示

顶点表示是用多边形的顶点序列来表示多边形。这种表示直观、几何意义强、占内存少，易于进行几何变换

但由于它没有明确指出哪些象素在多边形内，故不能直接用于面着色

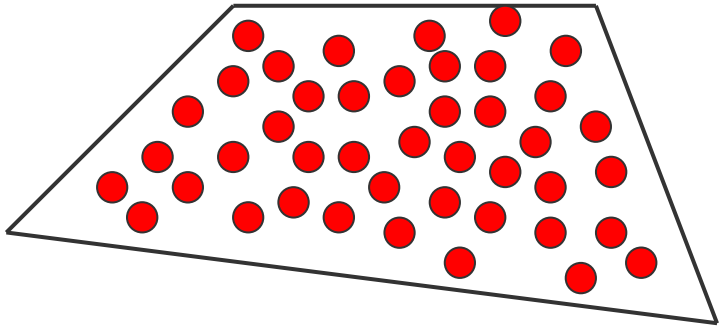


点阵表示是用位于多边形内的象素集合来刻画多边形。这种表示丢失了许多几何信息（如边界、顶点等），但它却是光栅显示系统显示时所需的表示形式。

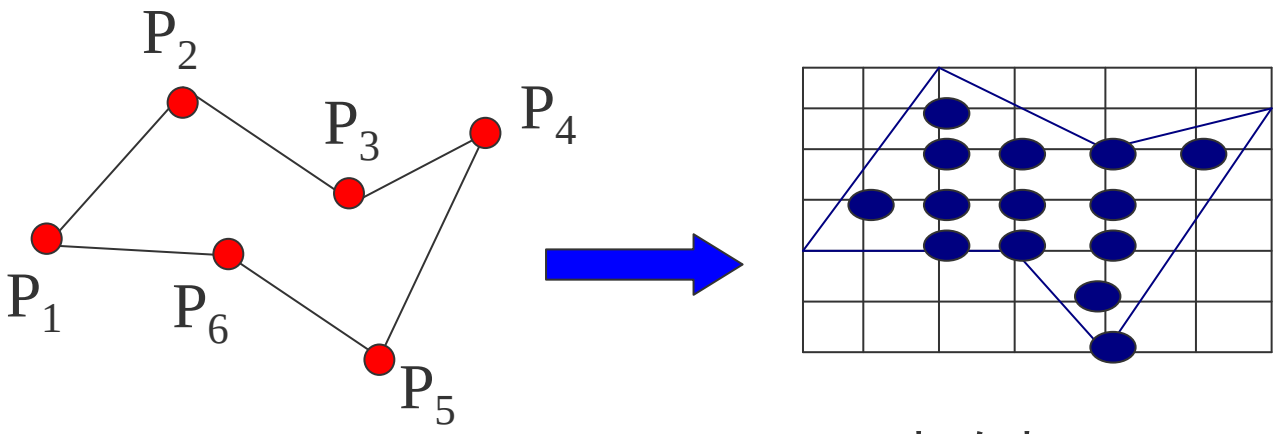


这涉及到两个问题：第一个问题是如果知道边界，能否求出哪些像素在多边形内?

第二个问题是知道多边形内部的像素，反过来如何求多边形的边界？



光栅图形的一个基本问题是把多边形的顶点表示转换为点阵表示。这种转换称为**多边形的扫描转换**

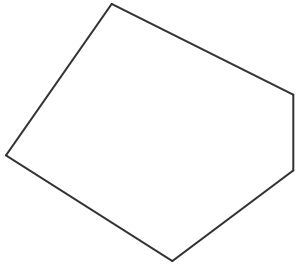


顶点表示　　点阵表示

多边形分为凸多边形、凹多边形、含内环的多边形等：

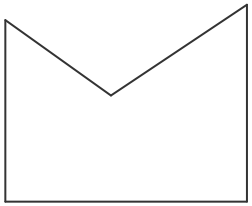
（1）凸多边形

任意两顶点间的连线均在多边形内



（2）凹多边形

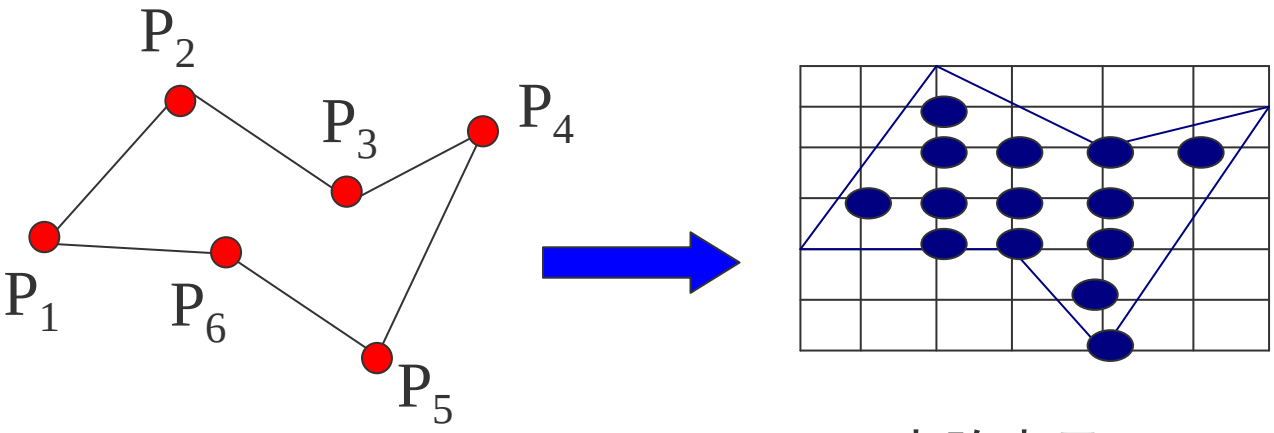
任意两顶点间的连线有不在在多边形内



（3）含内环的多边形

多边形内包含多边形

现在的问题是，知道多边形的边界，如何找到多边形内部的点，即把多边形内部填上颜色

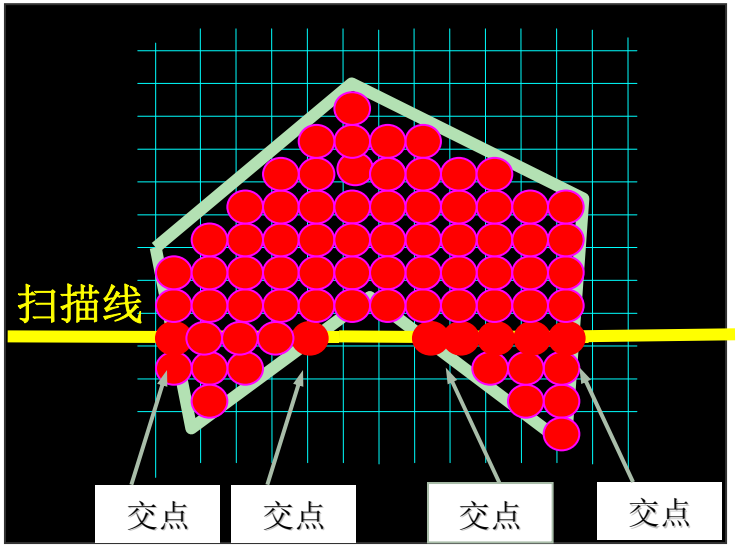


顶点表示　　点阵表示

1、X-扫描线算法

X-扫描线算法填充多边形的基本思想是按扫描线顺序，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像素，即完成填充工作

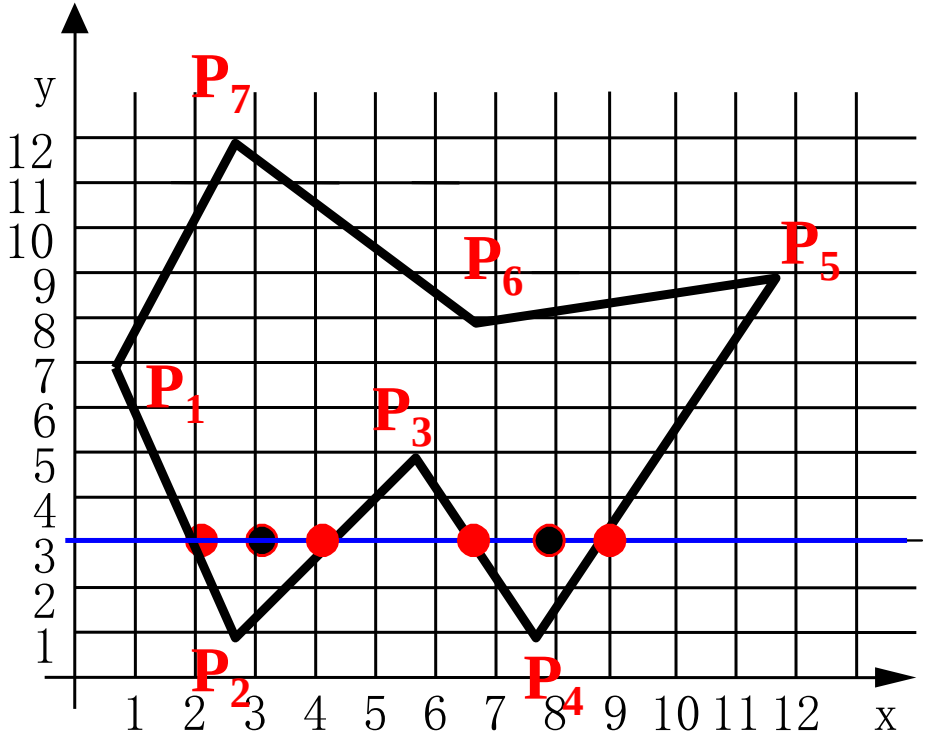
区间的端点可以通过计算扫描线与多边形边界线的交点获得



如扫描线y=3与多边形的边界相交于4点：

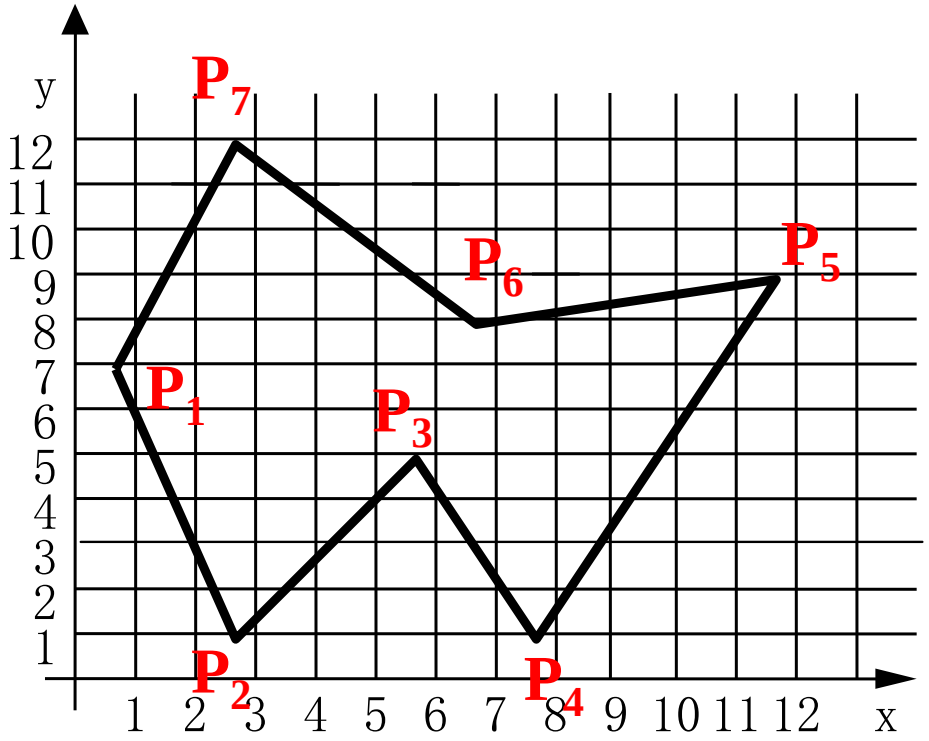
（2，3）、（4，3）、（7，3）、（9，3）。

这四点定义了扫描线从X=2到X=4，从X=7到X=9两个落在多边形内的区间，该区间内的像素应取填充色



算法的核心是按X递增顺序排列交点的X坐标序列。由此，可得到X-扫描线算法步骤如下：

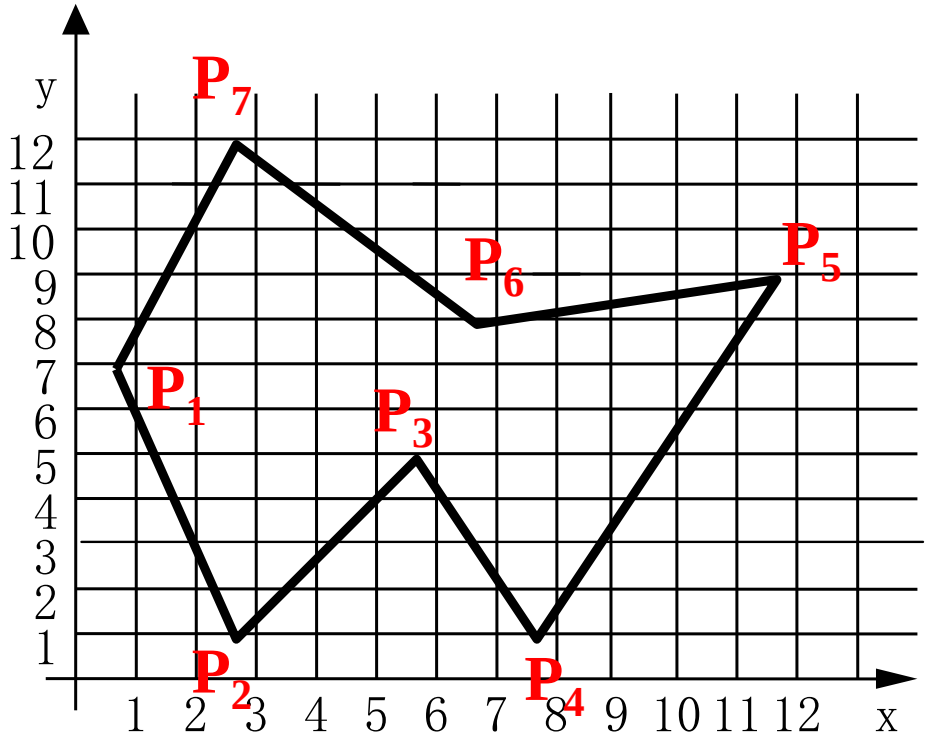
（1）确定多边形所占有的最大扫描线数，得到多边形顶点的最小和最大y值（y_{min}和y_{max}）



算法的核心是按X递增顺序排列交点的X坐标序列。由此，可得到X-扫描线算法步骤如下：

（2）从$y = y_{min}$到$y = y_{max}$，

每次用一条扫描线进行填充

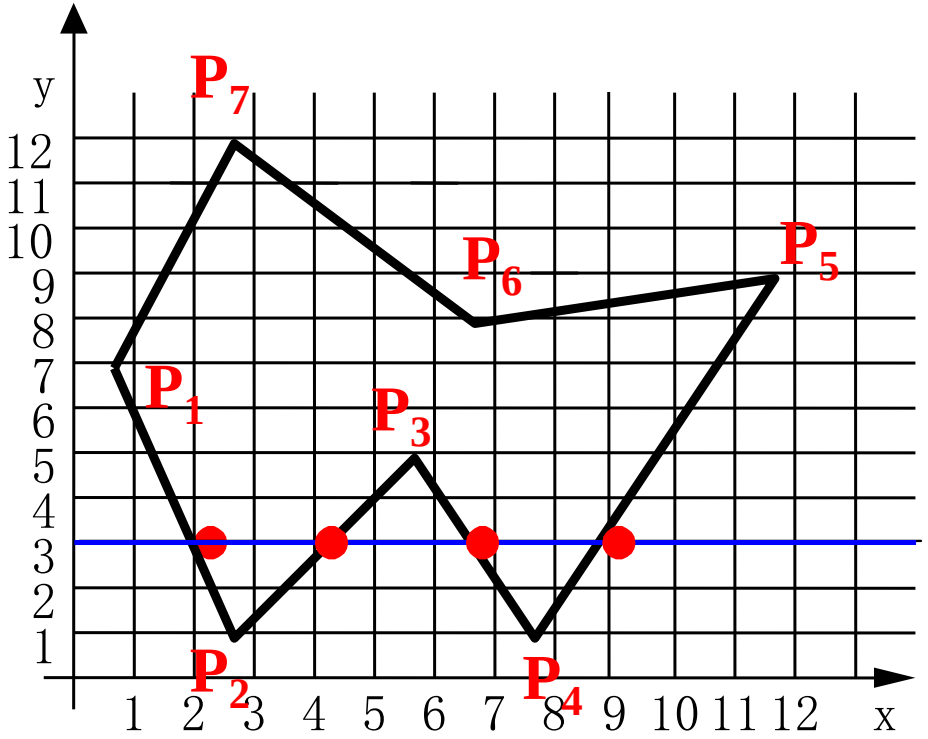


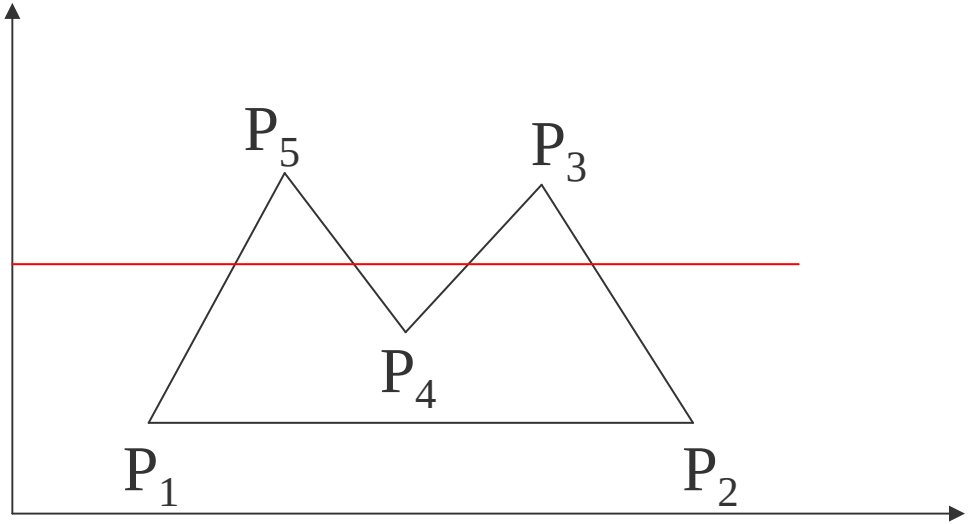
算法的核心是按X递增顺序排列交点的X坐标序列。由此，可得到X-扫描线算法步骤如下：

（3）对一条扫描线填充的过程可分为四个步骤：

a、求交：计算扫描线与多边形各边的交点

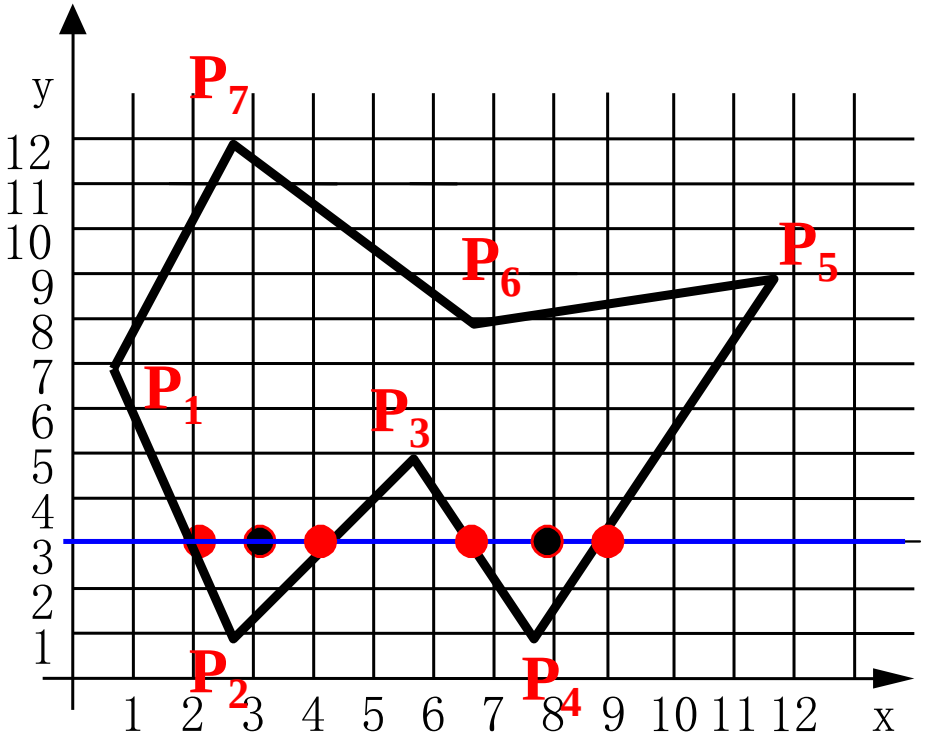
b、排序：把所有交点按递增顺序进行排序




P5
P3
P4
P1
P2

算法的核心是按X递增顺序排列交点的X坐标序列。由此，可得到X-扫描线算法步骤如下：

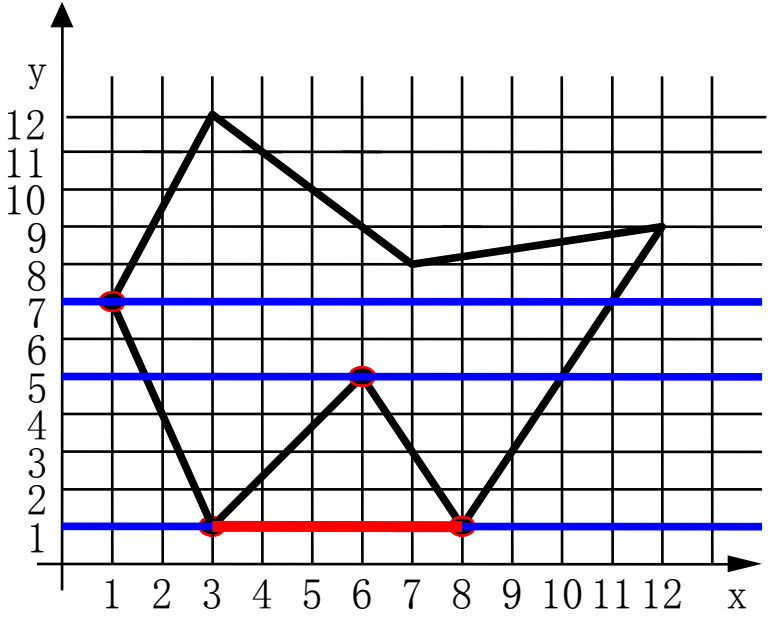
（3）对一条扫描线填充的过程可分为四个步骤：



c、交点配对：第一个与第二个，第三个与第四个

d、区间填色：把这些相交区间内的像素置成不同于背景色的填充色

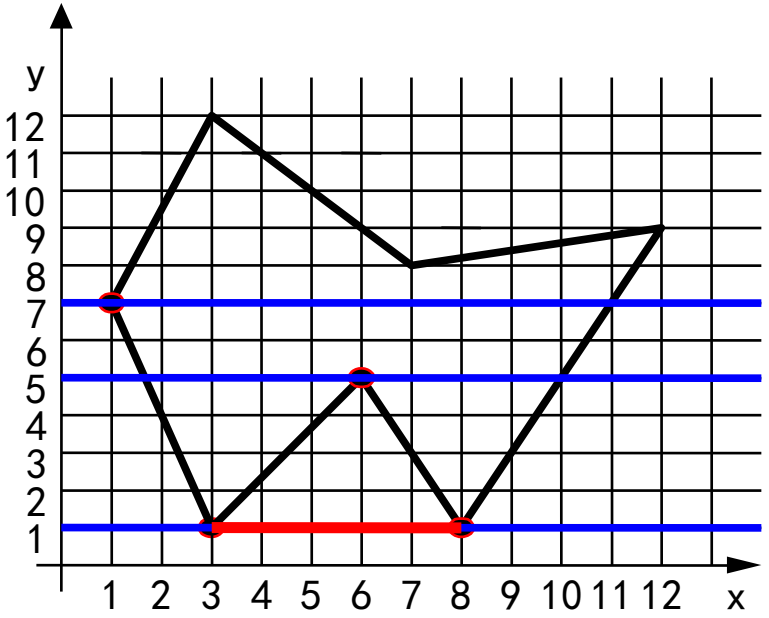
当扫描线与多边形顶点相交时，交点的取舍问题（交点的个数应保证为偶数个）



解决方案：

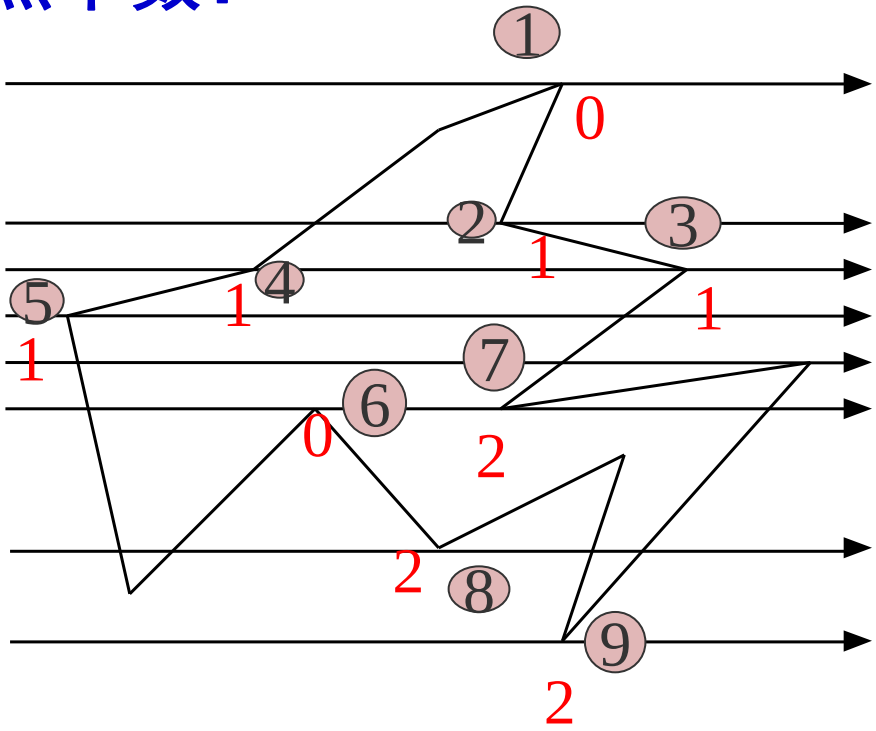
（1）若共享顶点的两条边分别落在扫描线的两边，交点只算一个

（2）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



检查共享顶点的两条边的另外两个端点的y值，按这两个y值中大于交点y值的个数来决定交点数

举例计算交点个数：



为了计算每条扫描线与多边形各边的交点，最简单的方法是把多边形的所有边放在一个表中。在处理每条扫描线时，按顺序从表中取出所有的边，分别与扫描线求交

这个算法效率低，为什么？

关键问题是求交！而求交是很可怕的，求交的计算量是非常大的

1、X-扫描线算法

X-扫描线算法填充多边形的基本思想是按扫描线顺序，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像素，即完成填充工作

关键问题是求交！求交的计算量是非常大的

排序、配对、填色总是要的！

最理想的算法是不求交！

2、多边形的扫描转换算法的改进

扫描转换算法重要意义是提出了图形学里两个重要的思想：

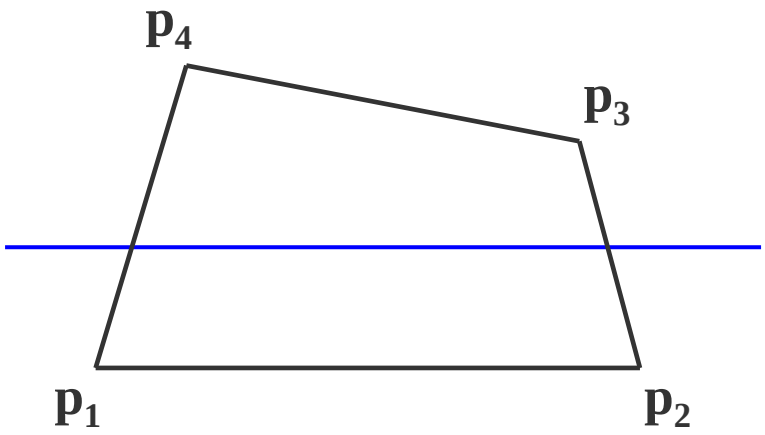
（1）扫描线：当处理图形图像时按一条条扫描线处理

（2）增量的思想

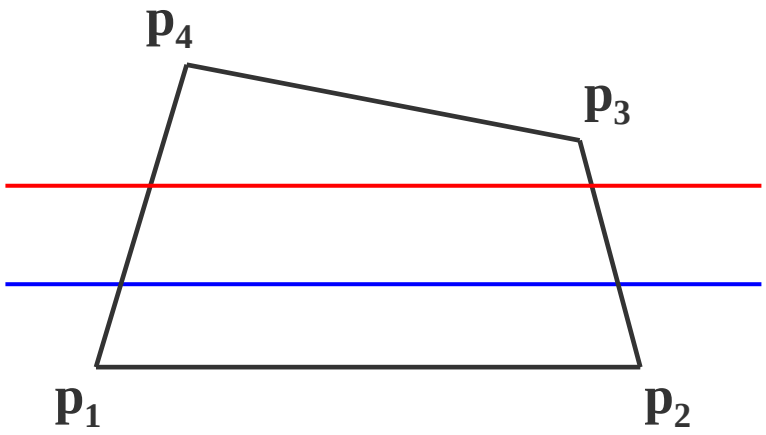
求交点的时候能不能也采取增量的方法？每条扫描线的y值都知道，关键是求x的值。x是什么？

可以从三方面考虑加以改进：

（1）在处理一条扫描线时，仅对与它相交的多边形的边（有效边）进行求交运算

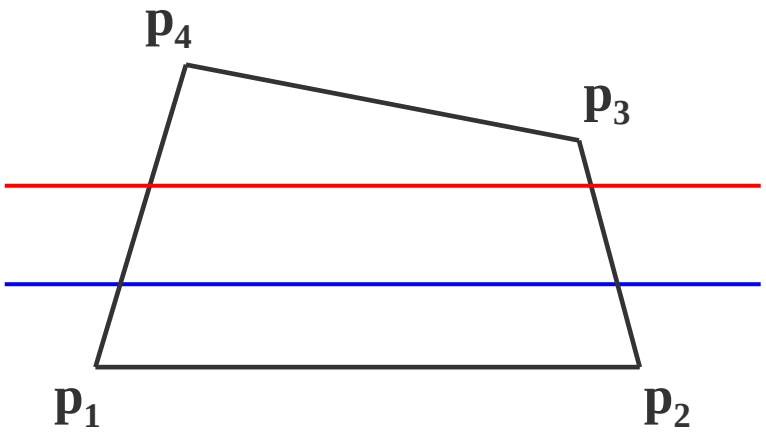


（2）考虑扫描线的连贯性



即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非常相似

（3）最后考虑多边形的连贯性



即当某条边与当前扫描线相交时，它很可能也与下一条扫描线相交

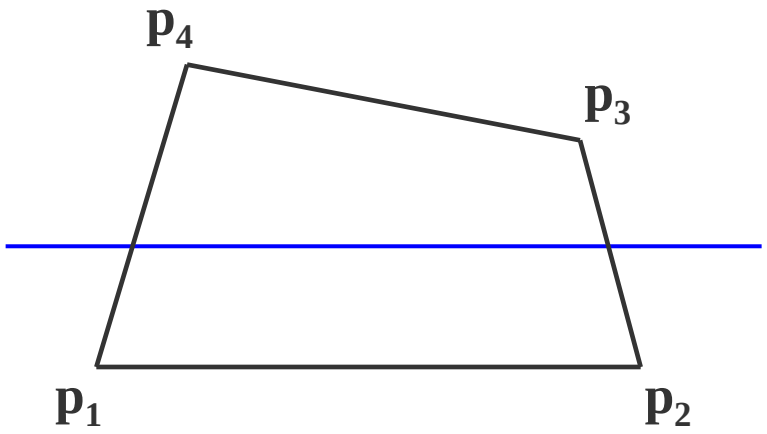
为了避免求交运算，需要引进一套特殊的数据结构

2、多边形的扫描转换算法的改进

为了避免求交运算，需要引进一套特殊的数据结构

数据结构：

（1）活性边表(AET)：把与当前扫描线相交的边称为活性边，并把它们按与扫描线交点x坐标递增的顺序存放在一个链表中。



（2）结点内容（一个结点在数据结构里可用结构来表示）

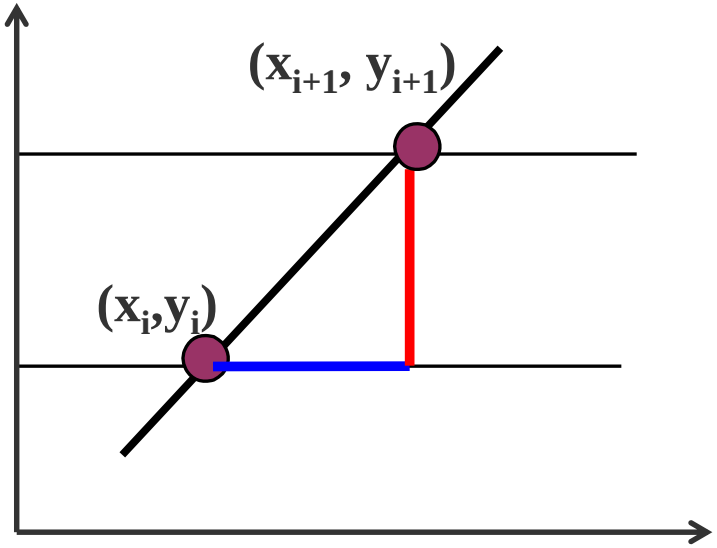
x：　当前扫描线与边的交点坐标

△x：　从当前扫描线到下一条扫描线间x的增量

y_{max}：　该边所交的最高扫描线的坐标值y_{max}

x	△x	y_{max}	next

随着扫描线的移动，扫描线与多边形的交点和上一次交点相关：



设边的直线斜率为k

$$k = \frac{\Delta y}{\Delta x} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

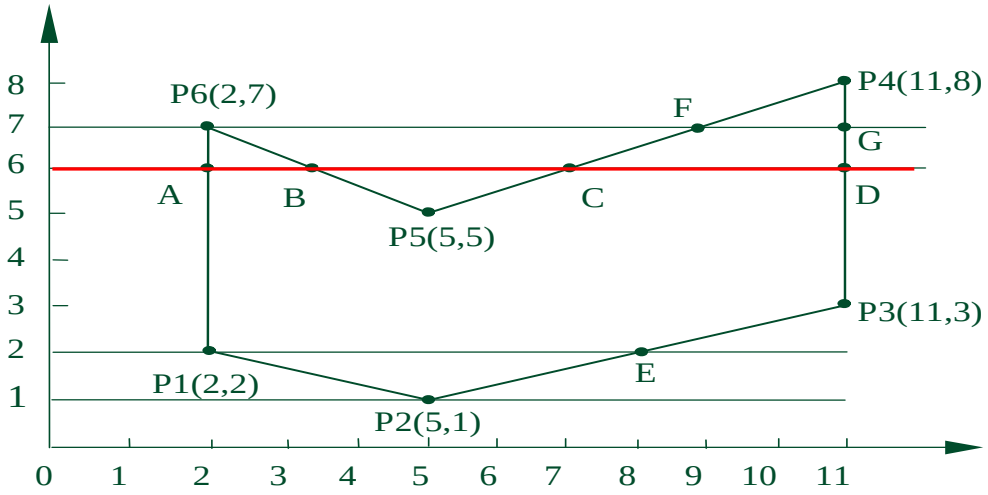
$$x_{i+1} - x_i = \frac{1}{k} \Rightarrow x_{i+1} = x_i + \frac{1}{k}$$

即： $\Delta x = \frac{1}{k}$

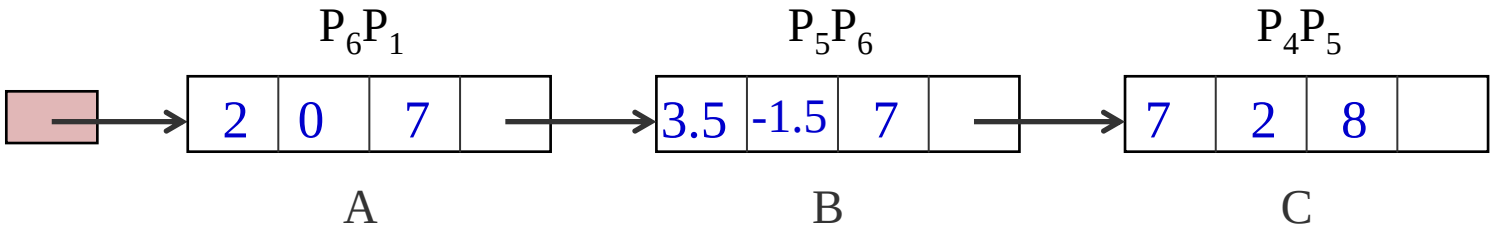
另外，需要知道一条边何时不再与下一条扫描线相交，以便及时把它从有效边表中删除出去，避免下一步进行无谓的计算

x	△x	y_{max}	next

其中x为当前扫描线与边的交点，y_{max}是边所在的最大扫描线值，通过它可以知道何时才能“抛弃”该边，△x表示从当前扫描线到下一条扫描线之间的x增量即斜率的倒数。next为指向下一条边的指针


8
7
6
5
4
3
2
1
0 1 2 3 4 5 6 7 8 9 10 11
P6(2,7)
P4(11,8)
F
G
A
B
C
D
P5(5,5)
P3(11,3)
P1(2,2)
E
P2(5,1)

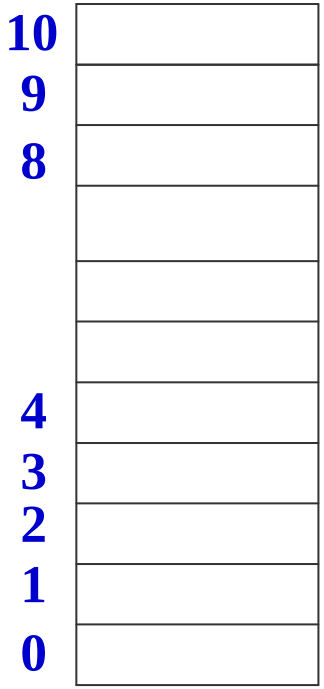
x	△x	y_{max}	next

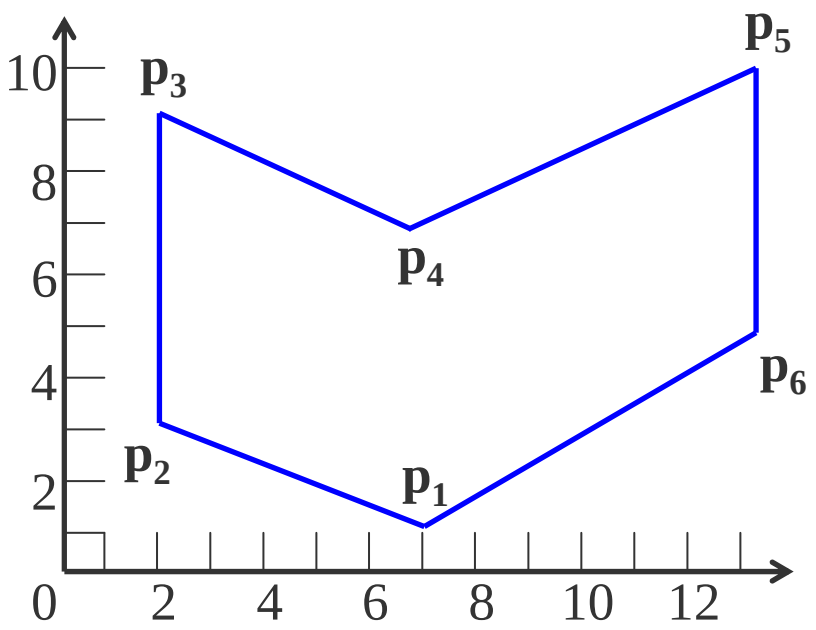

P6P1
P5P6
P4P5
2 0 7
3.5 -1.5 7
7 2 8
A
B
C

为了方便活性边表的建立与更新，需构造一个新边表（NET），用来存放多边形的边的信息，分为4个步骤：

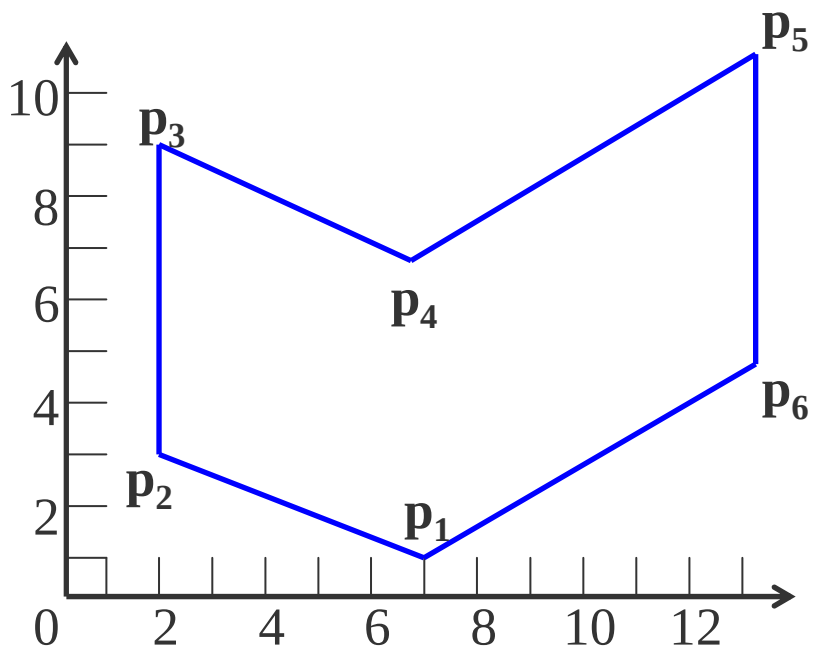
（1）首先构造一个纵向链表，链表的长度为多边形所占有的最大扫描线数，链表的每个结点，称为一个吊桶，对应多边形覆盖的每一条扫描线

（1）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



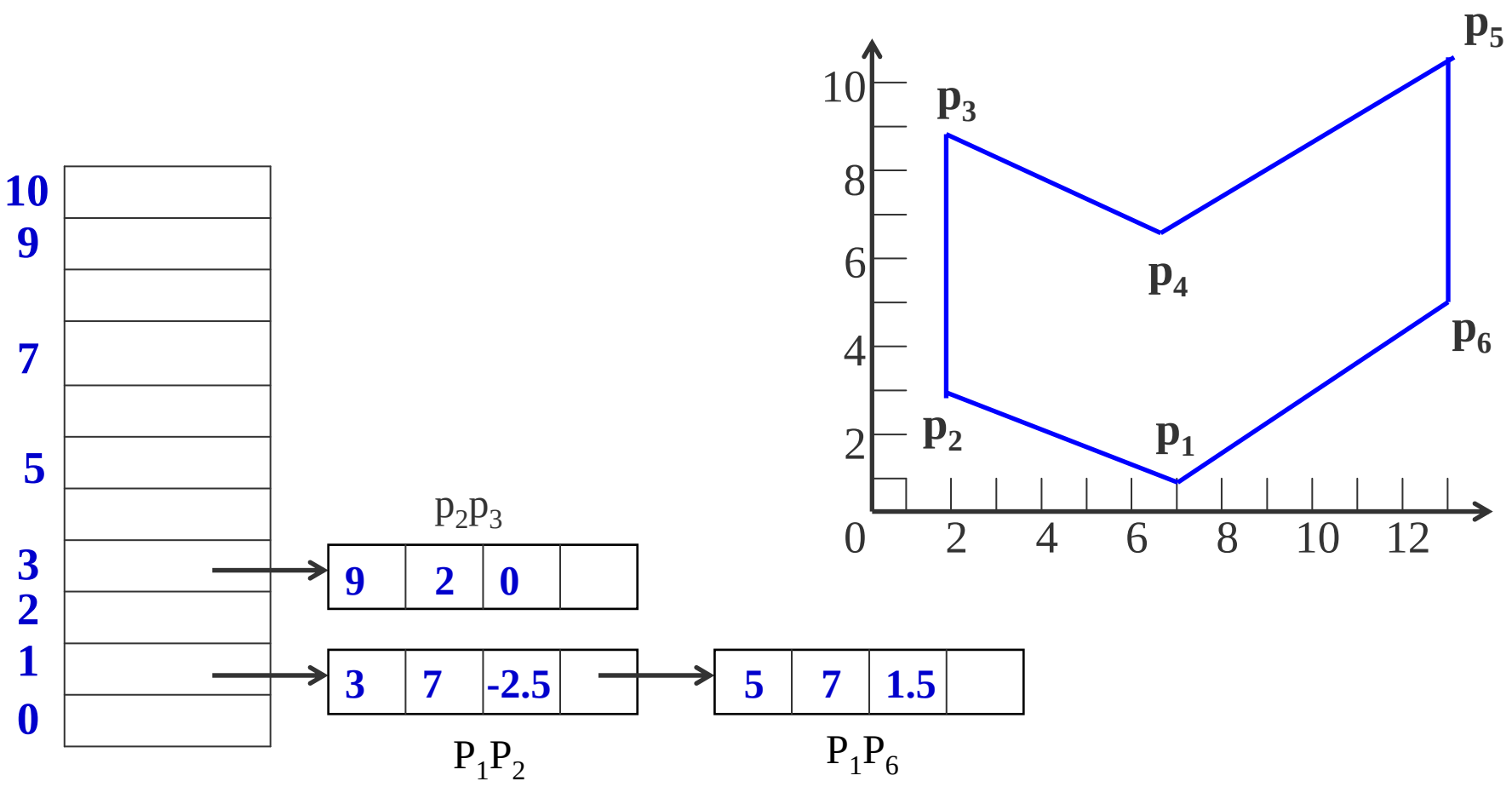


（2）NET挂在与该边低端y值相同的扫描线桶中。也就是说，存放在该扫描线第一次出现的边



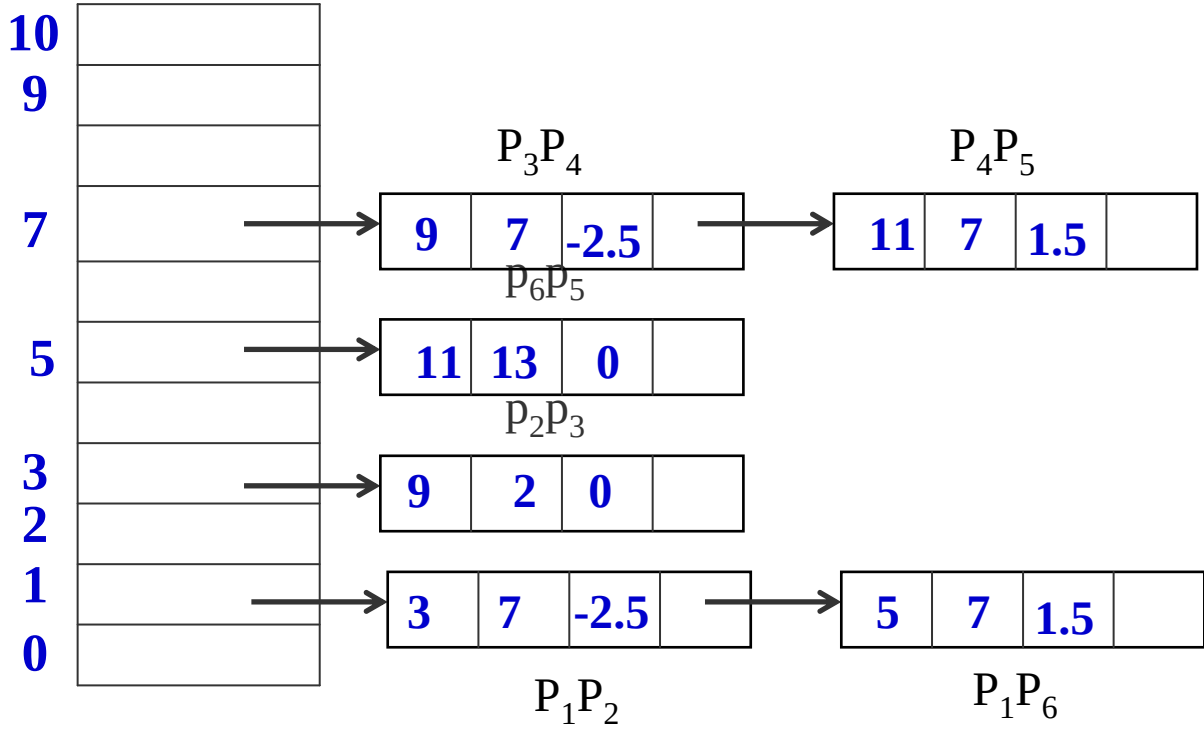
- 该边的y_{max}
- 该边较低点的x坐标值x_{min}
- 该边的斜率1/k
- 指向下一条具有相同较低端y坐标的边的指针

y_{max}	x_{min}	1/k	next


10
9
7
5
3
2
1
0
p_2p_3
9
2
0
3
7
-2.5
5
7
1.5
P_1P_2
P_1P_6
10
8
6
4
2
0
2
4
6
8
10
12
p_3
p_4
p_5
p_6
p_2
p_1

从右边这个NET表里就知道多边形是从哪里开始的

在这个表里只有1、3、5、7处有边，从y=1开始做，而在1这条线上有两条边进来了，然后就把这两条边放进活性边表来处理



每做一次新的扫描线时，要对已有的边进行三个处理：

1、是否被去除掉；

2、如果不被去除，第二就要对它的数据进行更新。所谓更新数据就是要更新它的x值，即：x+1/k

3、看有没有新的边进来，新的边在NET里，可以插入排序插进来。

这个算法过程从来没有求交，这套数据结构使得你不用求交点！避免了求交运算。

```
void polyfill (polygon, color)
       int color;  多边形   polygon;
       {  for (各条扫描线i )
           { 初始化新边表头指针NET[i];
             把y_min = i 的边放进边表NET[i];
           }
          y = 最低扫描线号;
          初始化活性边表AET为空;
          for (各条扫描线i )
               {
                  把新边表NET[i] 中的边结点用插入排序法插入AET表,
                      使之按x坐标递增顺序排列;
                  遍历AET表, 把配对交点区间(左闭右开)上的象素(x,y)
                     , 用putpixel(x,y,color) 改写象素颜色值;
                  遍历AET表, 把y_max= i 的结点从AET表中删除, 并把y_max>i
                    结点的x值递增Δx;
                  若允许多边形的边自相交, 则用冒泡排序法对AET表重新排序;
               }
        }  /* polyfill */
```

多边形扫描转换算法小结

扫描线法可以实现已知任意多边形域边界的填充。该填充算法是按扫描线的顺序，计算扫描线与待填充区域的相交区间，再用要求的颜色显示这些区间的像素，即完成填充工作

为了提高算法效率：

（1）增量的思想

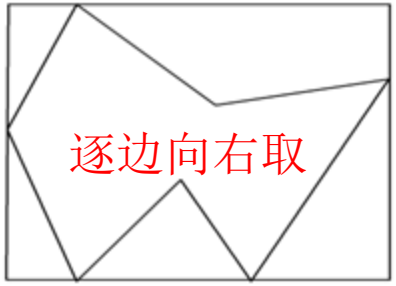
（2）连贯性思想

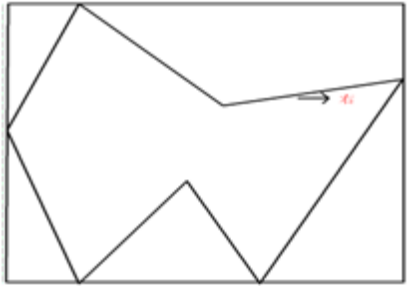
（3）构建了一套特殊的数据结构

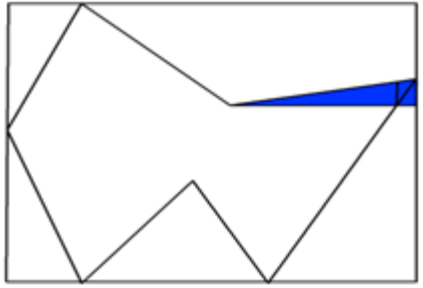
这里区间的端点通过计算扫描线与多边形边界的交点获得。所以待填充区域的边界线必须事先知道，因此它的缺点是无法实现对未知边界的区域填充

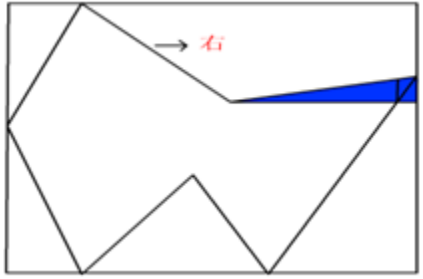
3、边缘填充算法

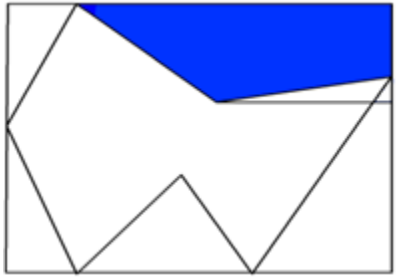
其基本思想是按任意顺序处理多边形的每条边。在处理每条边时，首先求出该边与扫描线的交点，然后将每一条扫描线上交点右方的所有像素取补。多边形的所有边处理完毕之后，填充即完成。

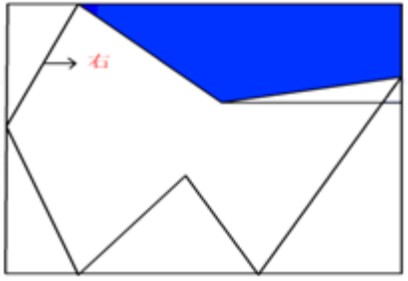

逐边向右取

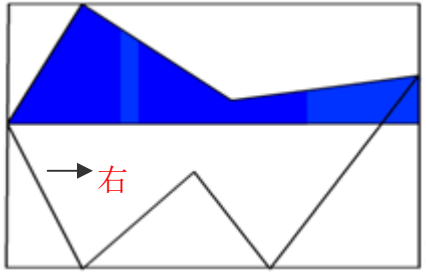

右

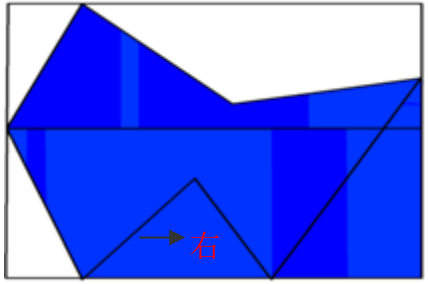


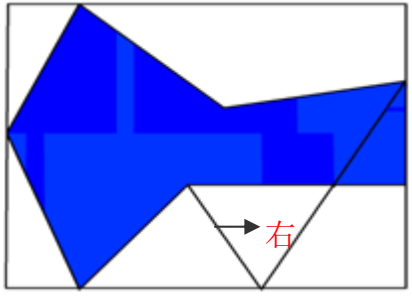

右

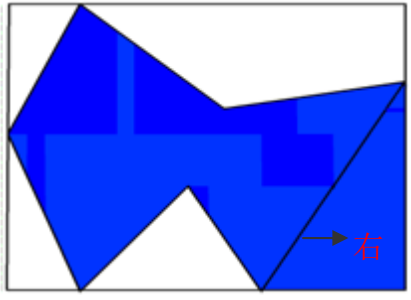


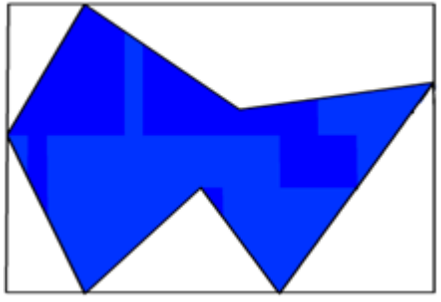

右


右


右


右


右



算法简单，但对于复杂图型，每一象素可能被访问多次。输入和输出量比有效边算法大得多。

为了减少边缘填充法访问像素的次数，可采用栅栏填充算法

4、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。在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补

5、边界标志算法

帧缓冲器中对多边形的每条边进行直线扫描转换，亦即对多边形边界所经过的象素打上标志

然后再采用和扫描线算法类似的方法将位于多边形内的各个区段着上所需颜色

由于边界标志算法不必建立维护边表以及对它进行排序，所以边界标志算法更适合硬件实现，这时它的执行速度比有序边表算法快一至两个数量级。

第二章：光栅图形学算法

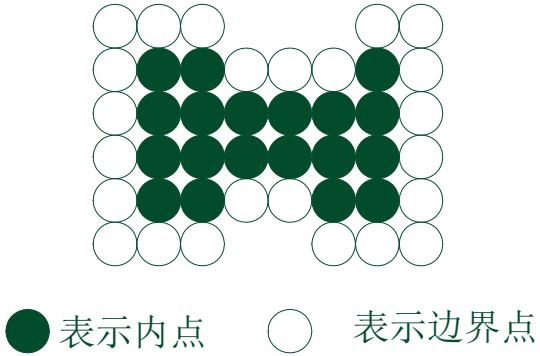
多边形的扫描转换与区域填充

二、区域填充

区域---指已经表示成点阵形式的填充图形，是象素的集合

区域填充是指将区域内的一点(常称种子点)赋予给定颜色，然后将这种颜色扩展到整个区域内的过程。

区域可采用**内点**表示和**边界**表示两种表示形式

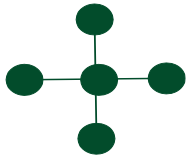


内点表示：枚举出区域内部的所有像素，内部的所有像素着同一个颜色，边界像素着与内部像素不同的颜色

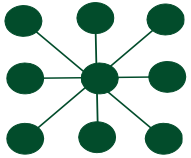
边界表示：枚举出边界上的所有像素，边界上的所有像素着同一个颜色，内部像素着与边界像素不同的颜色

区域填充算法要求区域是连通的，因为只有在连通区域中，才可能将种子点的颜色扩展到区域内的其它点。

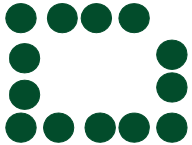
区域可分为4向连通区域和8向连通区域



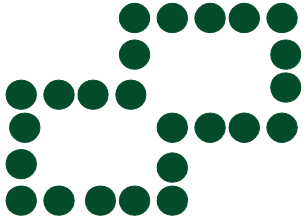
四个方向运动



八个方向运动



四连通区域

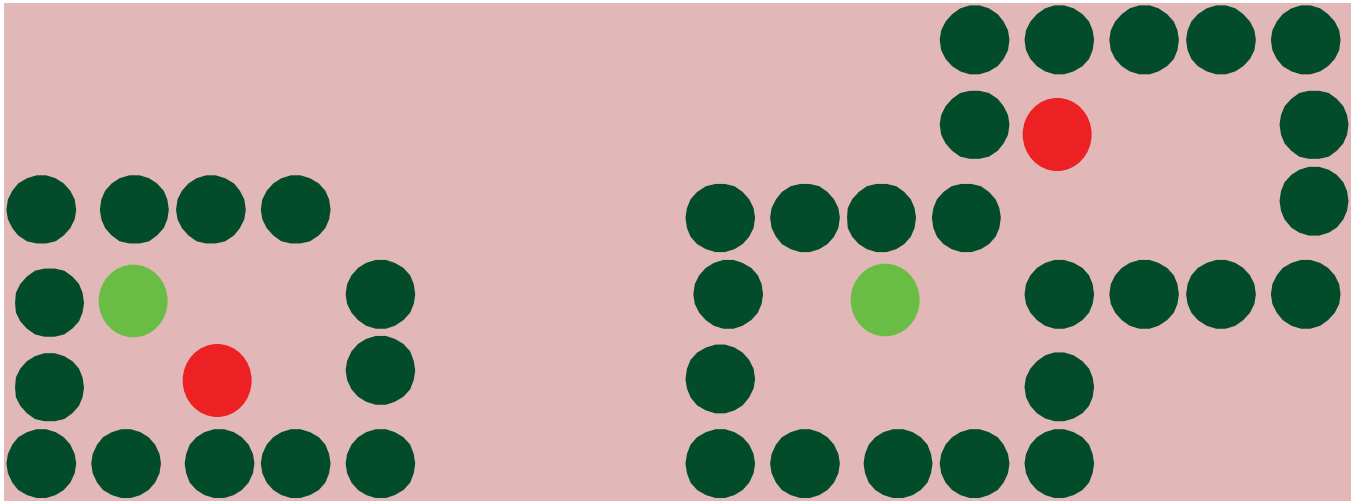


八连通区域

4向连通区域指的是从区域上一点出发，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

8向连通区域指的是从区域内每一象素出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合来到达

种子填充



四连通区域

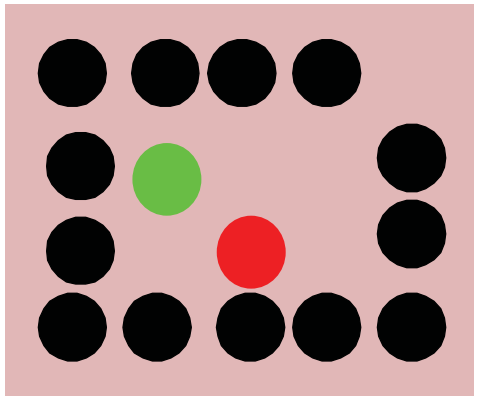
八连通区域

简单四连通种子填充算法（区域填充递归算法）

种子填充算法的原理是：假设在多边形区域内部有一像素已知，由此出发找到区域内的所有像素，用一定的颜色或灰度来填充

假设区域采用边界定义，即区域边界上所有像素均具有某个特定值，区域内部所有像素均不取这一特定值，而边界外的像素则可具有与边界相同的值

考虑区域的四向连通，即从区域上一点出发，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，到达区域内的任意像素。



使用栈结构来实现简单的种子填充算法

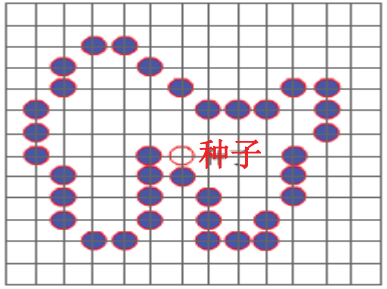
算法原理如下：

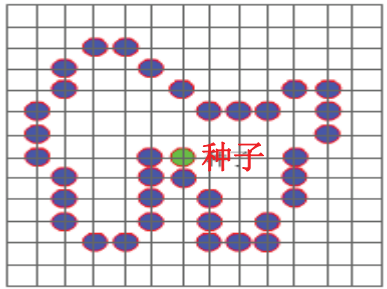
种子像素入栈，当栈非空时重复执行如下三步操作：

（1）栈顶像素出栈

（2）将出栈像素置成要填充色

（3）按左、上、右、下顺序检查与栈像素相邻的四个像素，若其中某个像素不在边界且未置成填充色，则把该像素入栈



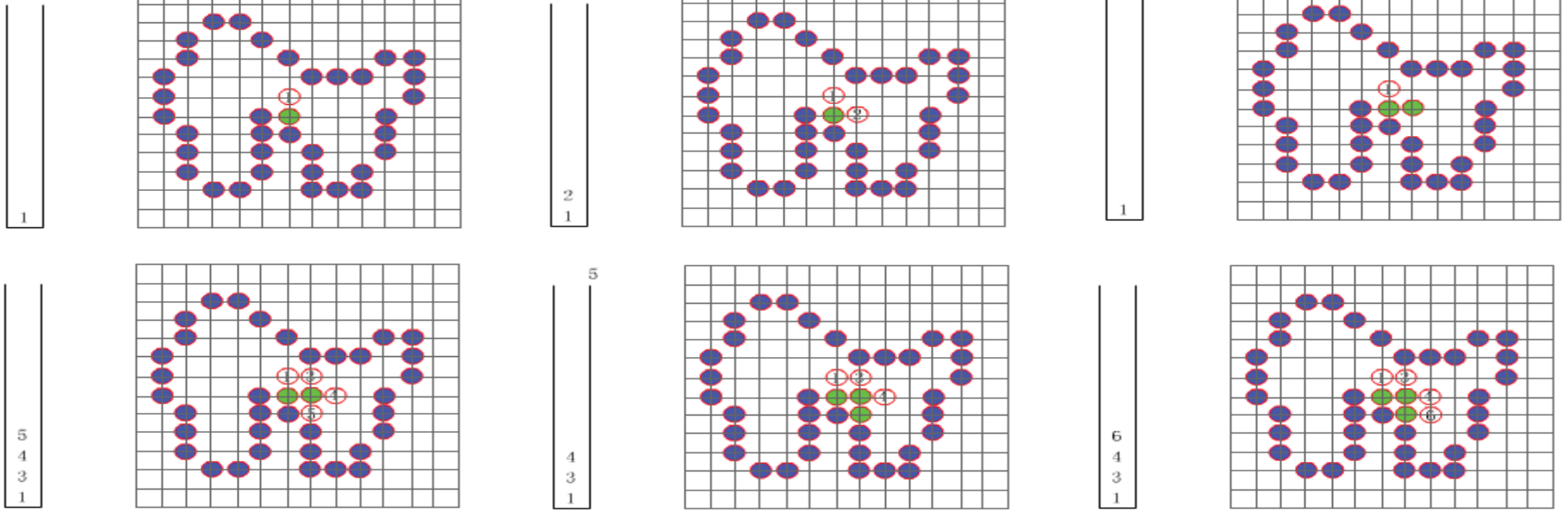


种子像素入栈，栈非空时重复执行如下三步：

（1）栈顶像素出栈

（2）将出栈像素置成要填充色

（3）按左、上、右、下顺序检查与栈像素相邻的四个像素，若其中某个像素不在边界且未置成填充色，则把该像素入栈



种子填充算法的不足之处

（1）有些像素会入栈多次，降低算法效率；栈结构占空间

（2）递归执行，算法简单，但效率不高。区域内每一像素都引进一次递归，进/出栈，费时费内存

（3）改进算法，减少递归次数，提高效率

可以采用区域填充的扫描线算法

三、多边形的扫描转换与区域填充算法小结

- 基本思想不同
  - 多边形扫描转换是指将多边形的顶点表示转化为点阵表示
  - 区域填充只改变区域的填充颜色，不改变区域表示方法

- 基本条件不同
  - 在区域填充算法中，要求给定区域内一点作为种子点，然后从这一点根据连通性将新的颜色扩散到整个区域
  - 扫描转换多边形是从多边形的边界(顶点)信息出发，利用多种形式的连贯性进行填充的

扫描转换区域填充的核心是知道多边形的边界，要得到多边形内部的像素集，有多种方法。其中扫描线算法是利用一套特殊的数据结构，避免求交，然后一条条扫描线确定

区域填充条件更强一些，不但知道边界，而且还知道区域内的一点，可以利用四连通或八连通区域不断往外扩展

填充一个定义的区域的选择包括：

- 选择实区域颜色或图案填充方式
- 选择某种颜色和图案

这些填充选择可应用于多边形区域或用曲线边界定义的区域；此外，区域可用多种画笔、颜色和透明度参数来绘制