# 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：

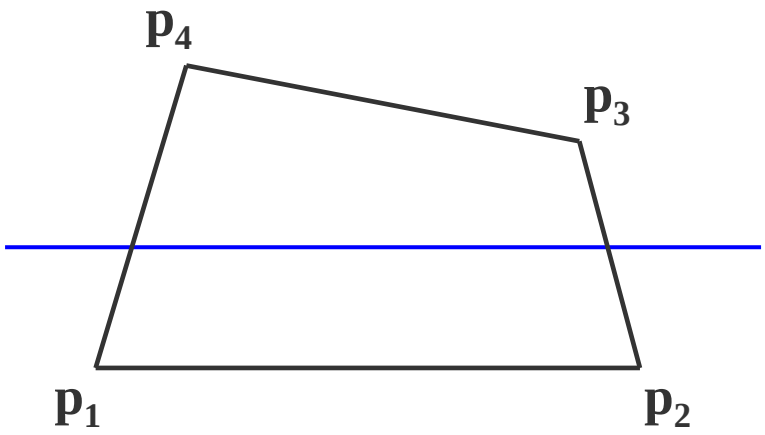
（1）扫描线：当处理图形图像时按一条条扫描线处理

（2）增量的思想

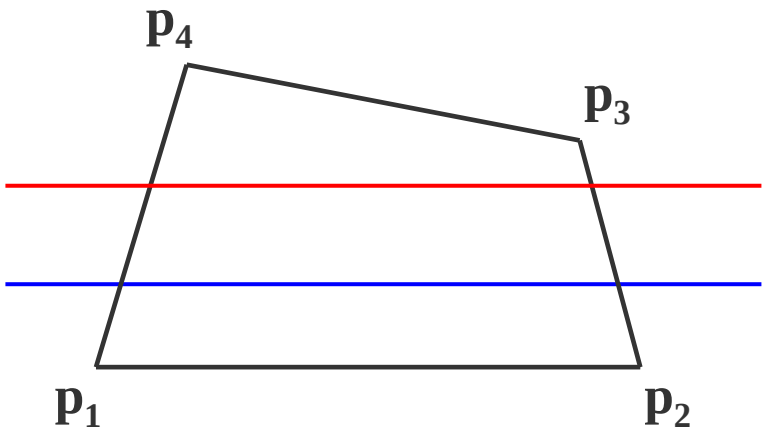
求交点的时候能不能也采取增量的方法？每条扫描线的y值都知道，关键是求x的值。x是什么？

## 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

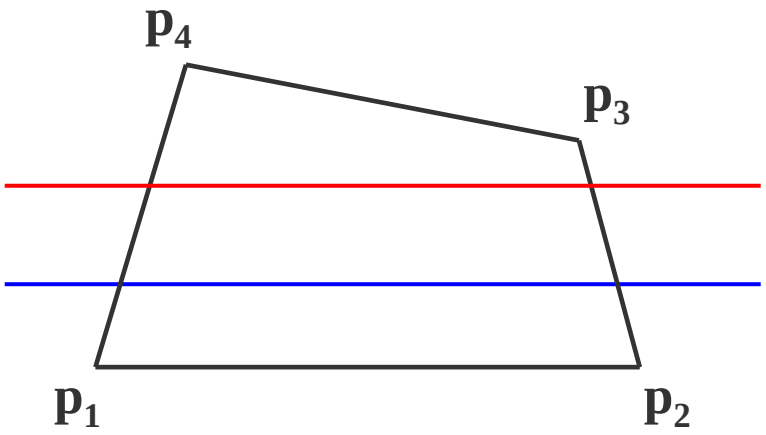


（2）考虑扫描线的连贯性




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

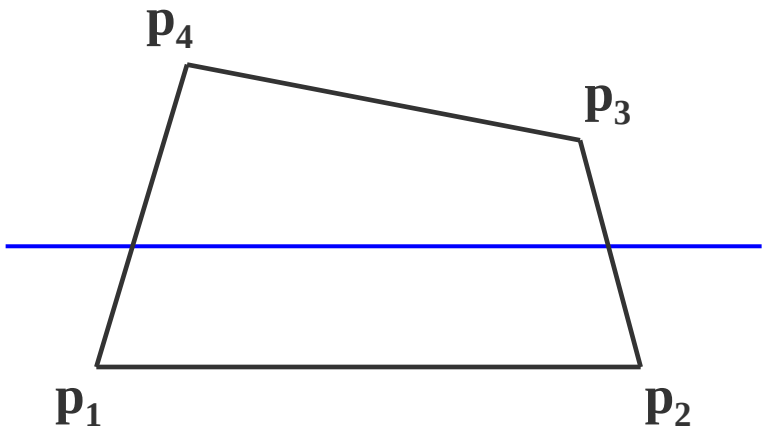
为了避免求交运算，需要引进一套特殊的数据结构

## 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）

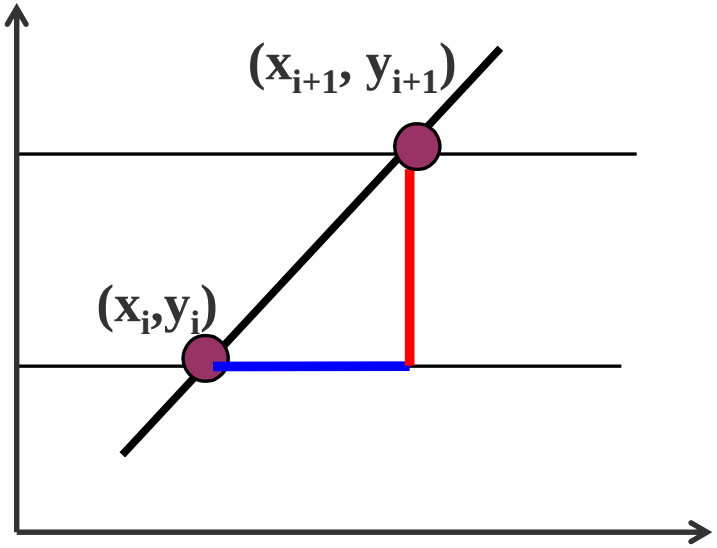
x：　当前扫描线与边的交点坐标

△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

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

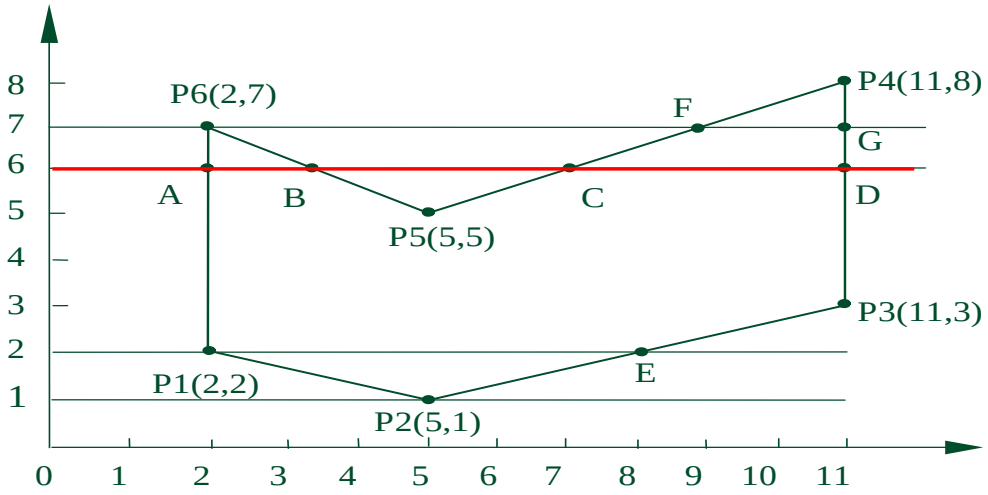
$$x_{i+1} - x_i = \frac{1}{k} \Rightarrow x_{i+1} = x_i + \frac{1}{k}$$

即： $\Delta x = \frac{1}{k}$

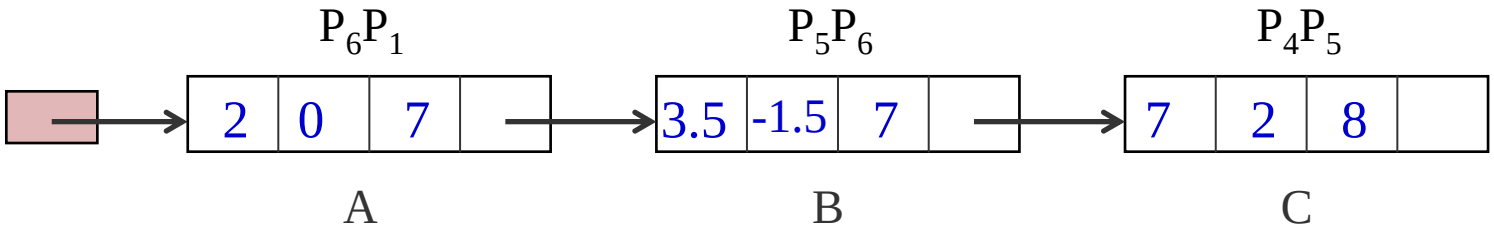
另外，需要知道一条边何时不再与下一条扫描线相交，以便及时把它从有效边表中删除出去，避免下一步进行无谓的计算

| 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)

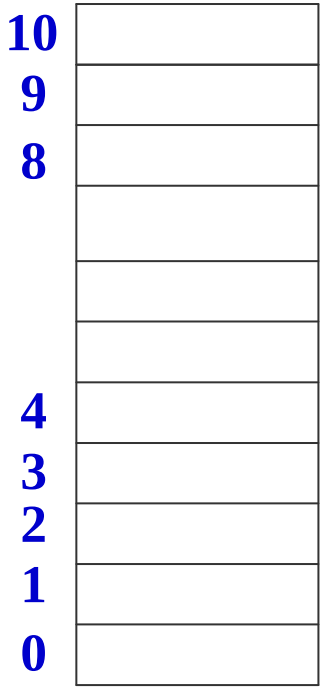
| x | △x | $y_{max}$ | next |
|---|---|---|---|

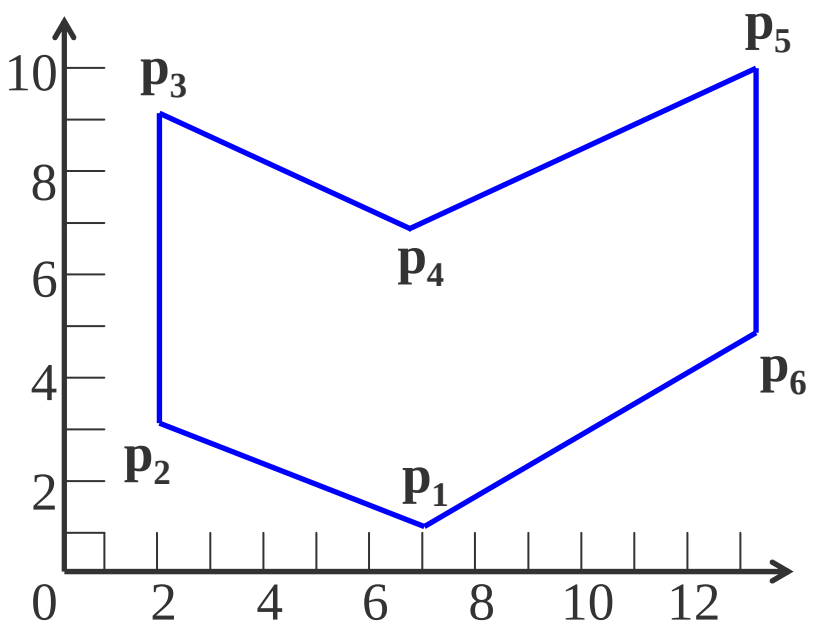

P6P1
P5P6
P4P5
2 0 7
3.5 -1.5 7
7 2 8
A
B
C

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：

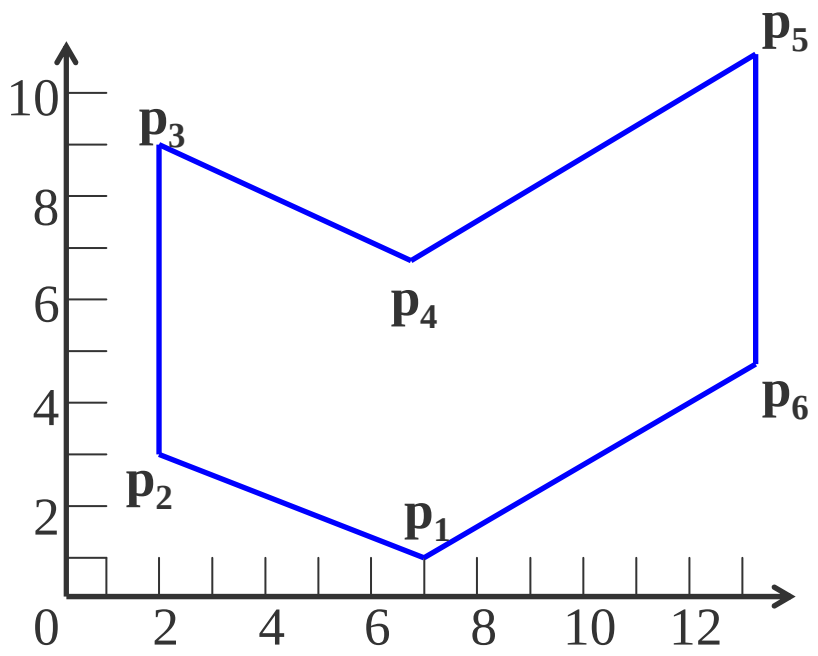
（1）首先构造一个纵向链表，链表的长度为多边形所占有的最大扫描线数，链表的每个结点，称为一个吊桶，对应多边形覆盖的每一条扫描线

（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



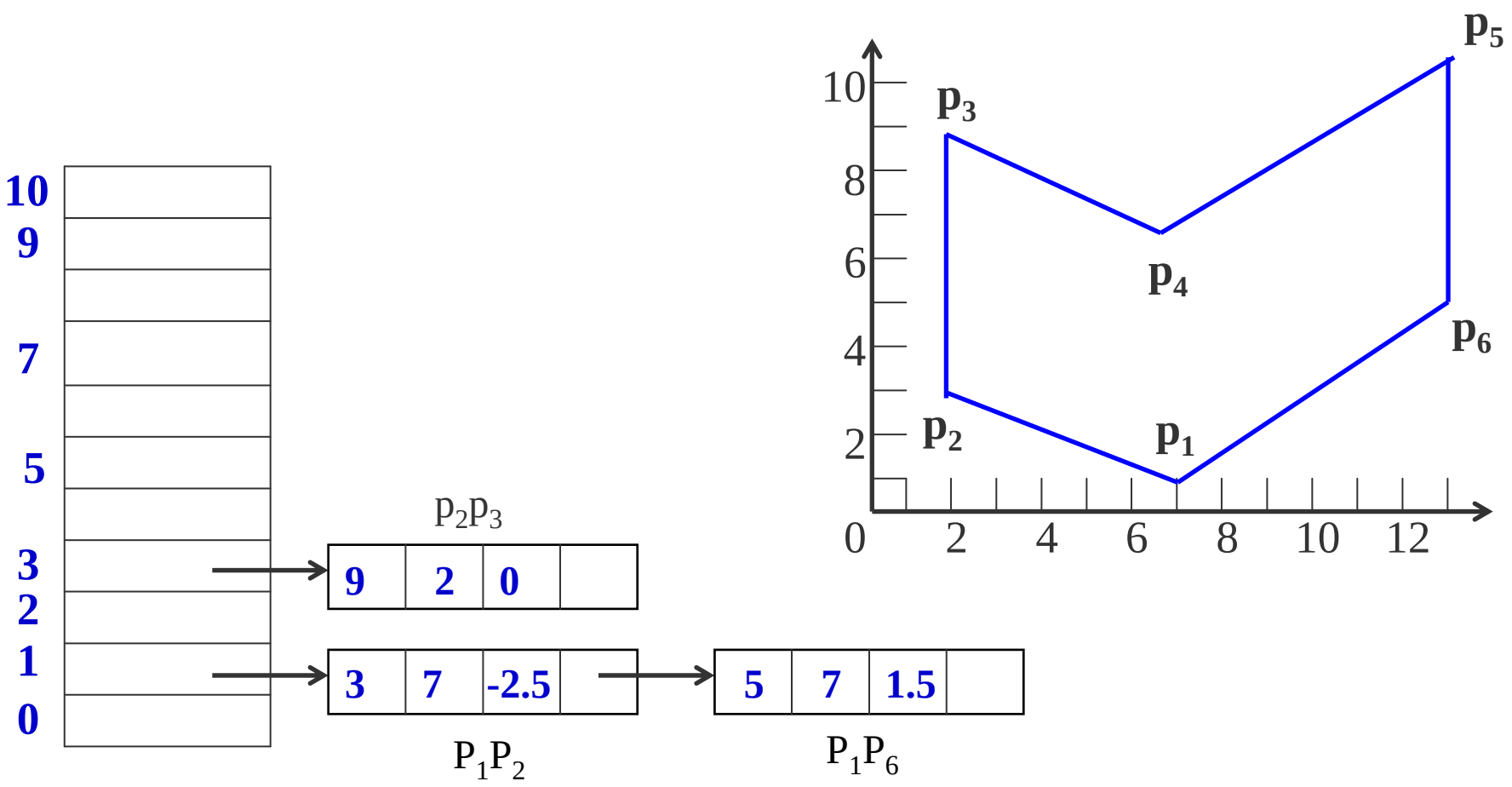



（2）NET挂在与该边低端y值相同的扫描线桶中。也就是说，存放在该扫描线第一次出现的边



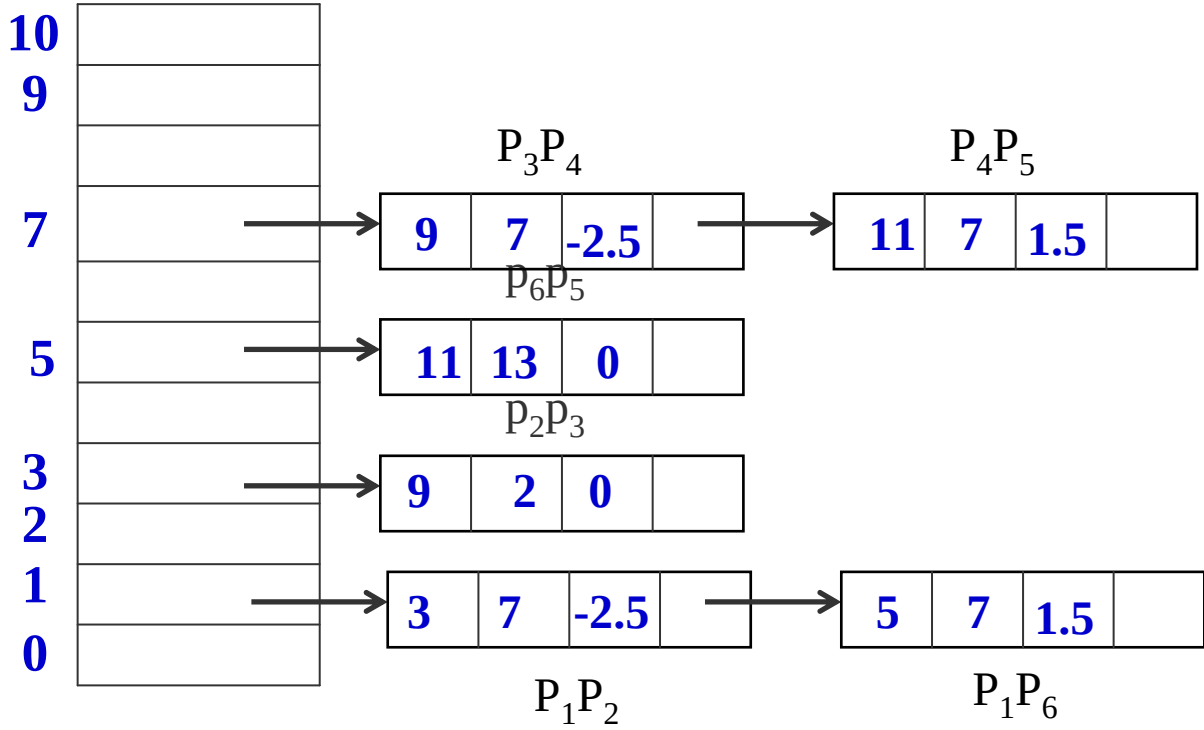

- 该边的$y_{max}$
- 该边较低点的x坐标值$x_{min}$
- 该边的斜率1/k
- 指向下一条具有相同较低端y坐标的边的指针

| $y_{max}$ | $x_{min}$ | 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

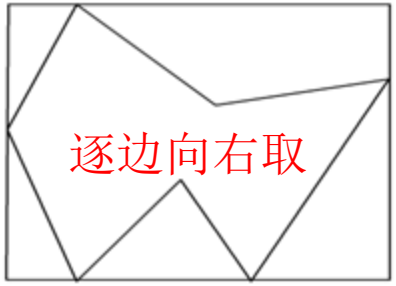
（2）连贯性思想

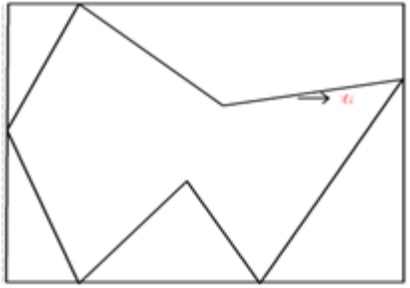
（3）构建了一套特殊的数据结构

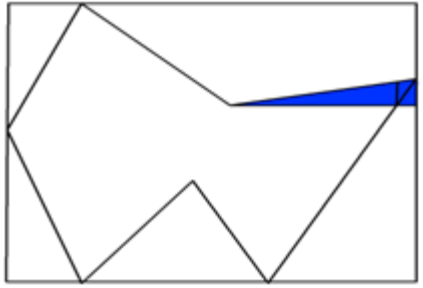
这里区间的端点通过计算扫描线与多边形边界的交点获得。所以待填充区域的边界线必须事先知道，因此它的缺点是无法实现对未知边界的区域填充

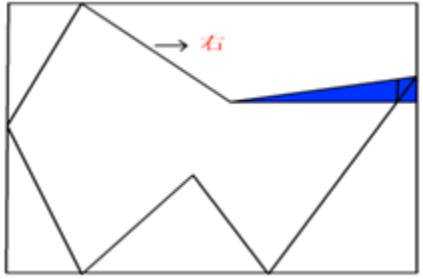
# 3、边缘填充算法

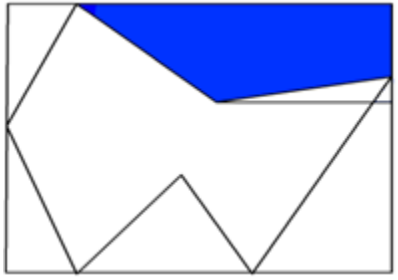
其基本思想是按任意顺序处理多边形的每条边。在处理每条边时，首先求出该边与扫描线的交点，然后将每一条扫描线上交点右方的所有像素取补。多边形的所有边处理完毕之后，填充即完成。

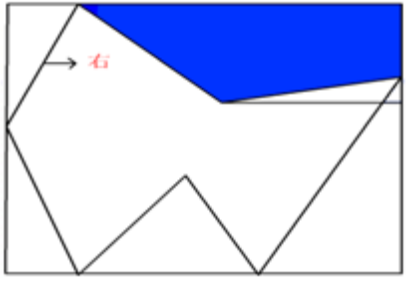

逐边向右取

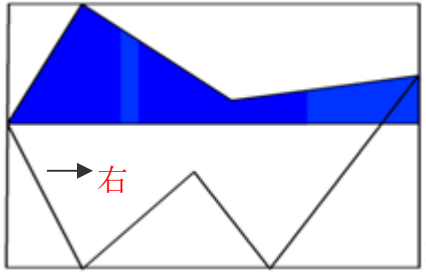

右

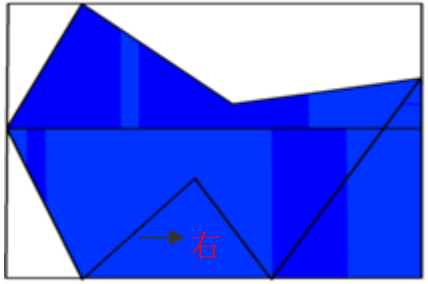


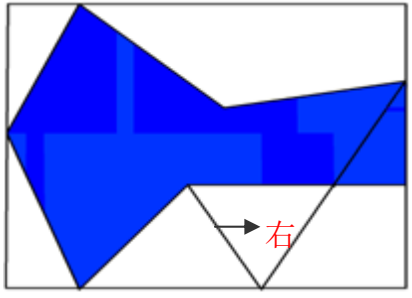

右

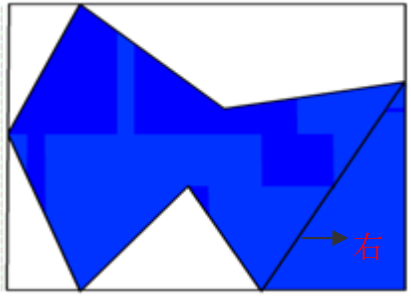


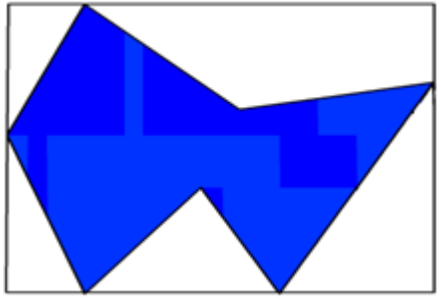

右


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。