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

- 二维线段裁剪
- 二维多边形裁剪
- 字符串裁剪

裁剪在GIS应用

- **MapGIS中裁剪**

short _dsGetMapDbsfRcClip (DbsAI dbsAi,MAPPROJECT PrjHand , D_RECT *frc) **//区域提取输出图库文件**

short _dsRectAskDbsClsDat(DbsAI dbsAi,short clsNo,D_RECT *frc) **//由给定的矩形范围提取当前指定点、线或区**

- **ArcGIS中裁剪**

//arcgisscripting包

Import arcgisscripting

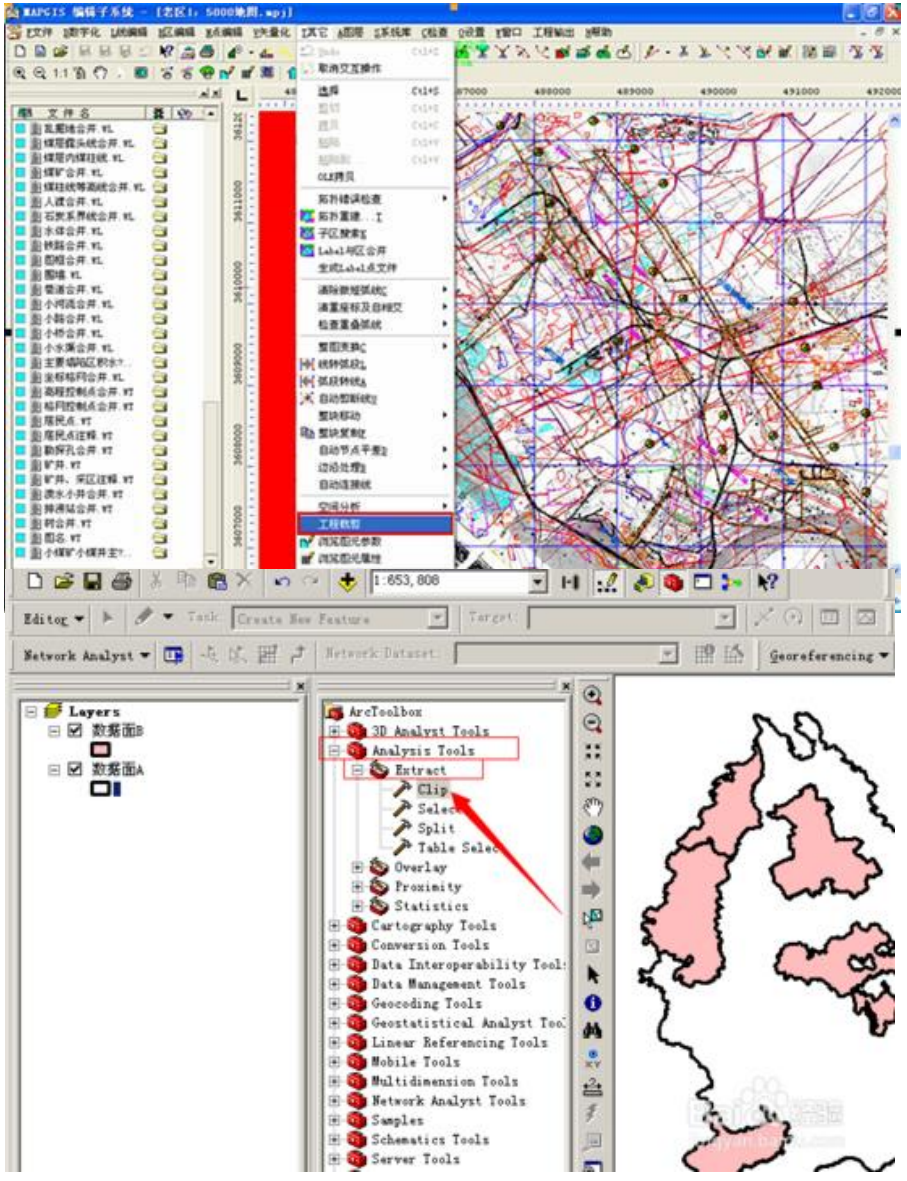
gp = arcgisscripting.create();

gp.Clip_management(“c:/image.tif” ,…)

// arcpy包

Imort arcpy

Arcpy.Clip_management(“c:/image.tif” ,…)



基本概念

- **裁剪的目的**
  - 判断图形元素是否落在裁剪窗口之内并找出其位于内部的部分
- **裁剪的处理的基础**
  - 图元关于窗口内外关系的判别
  - 图元与窗口的求交
- **裁剪对象**
  - 裁剪窗口：多边形或矩形
  - 裁剪图元：点、线段、多边形

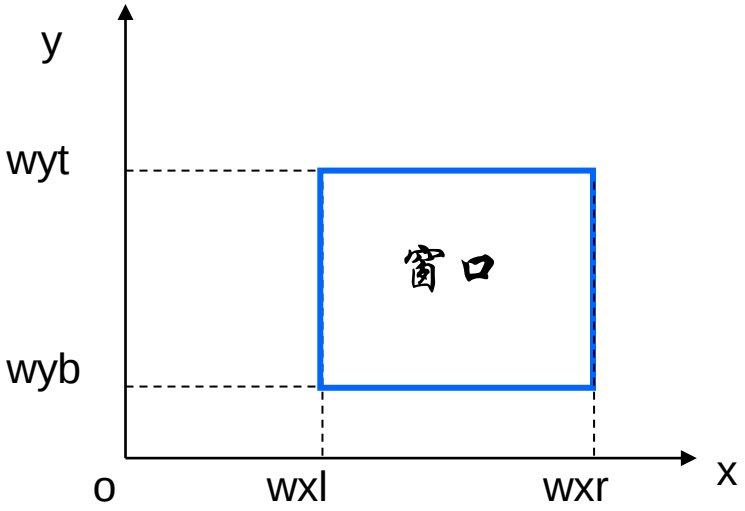
点裁剪

- **定义**

  点(x，y)在窗口内的充分必要条件是：

$$x\min \le x \le x\max$$

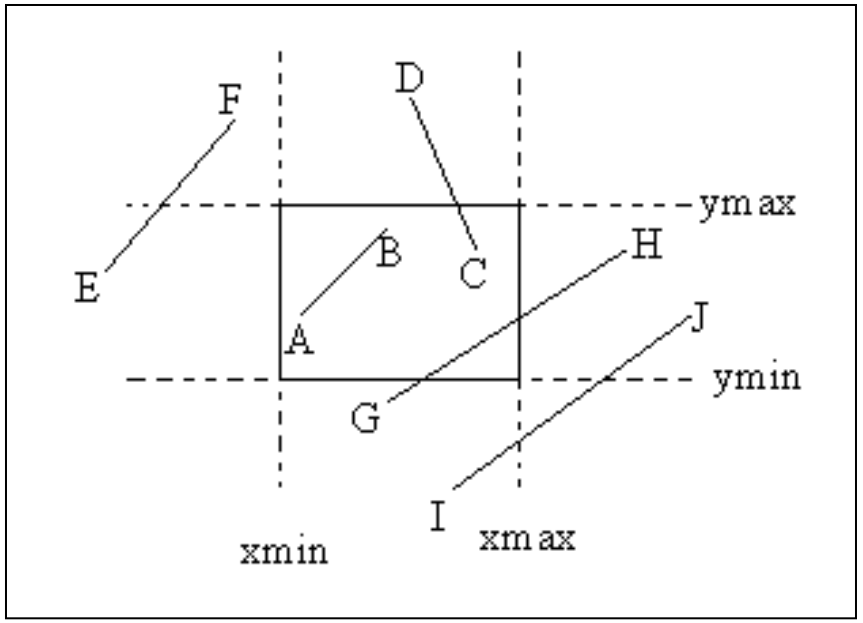
$$y\min \le y \le y\max$$



问题：对于任何多边形窗口，如何判别？

Cohn-Sutherland线段裁剪

- **待裁剪线段和窗口的关系**
  - 线段完全可见
  - 显然不可见
  - 线段至少有一端点在窗口之外，但非显然不可见



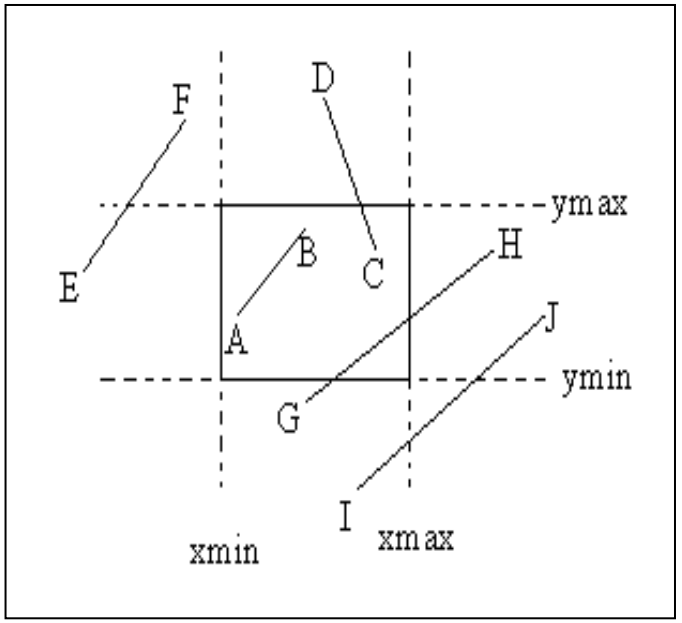
为提高效率应考虑：

1、快速判断情形(1)(2)；

2、设法减少情形(3)求交次数和每次求交时所需的计算量。

Cohn-Sutherland线段裁剪

- **算法思想**
  - 若p1p2完全在窗口内，则该线段可见
  - 若p1p2在窗口某一侧（左、右、上、下）则不可见
  - 若不满足上两条件，则求交点。交点将线段分为两段，舍弃在窗口外部分；另一部分重复上述处理



核心问题：

1、如何快速判断p1p2完全在窗口内、窗口某一侧？

2、如何求交点？

Cohn-Sutherland线段裁剪

- **快速判别方法：编码方法**

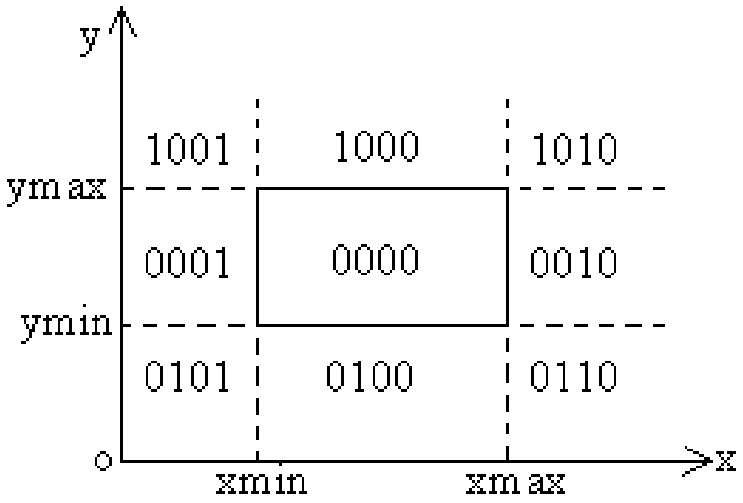
由窗口四条边所在直线把二维平面分成9个区域，每个区域赋予一个四位编码，$C_tC_bC_rC_l$，上下右左。

$$C_t=\begin{cases}1 & \text{当} y>y\max\\ 0 & else\end{cases}$$

$$C_b=\begin{cases}1 & \text{当} y<y\min\\ 0 & else\end{cases}$$

$$C_r=\begin{cases}1 & \text{当} x>x\max\\ 0 & else\end{cases}$$

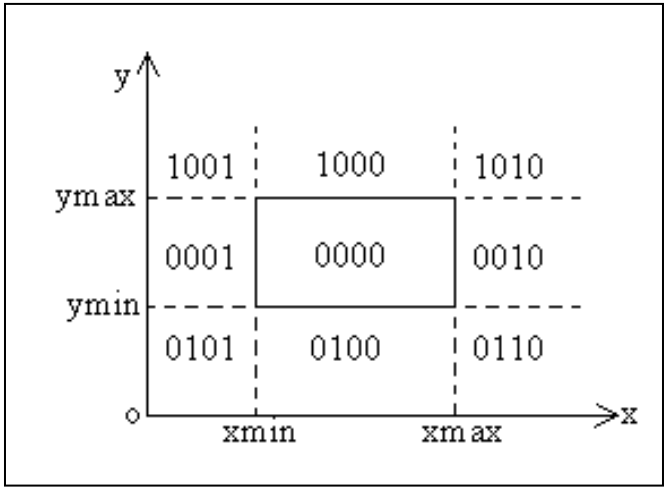
$$C_l=\begin{cases}1 & \text{当} x<x\min\\ 0 & else\end{cases}$$



Cohn-Sutherland线段裁剪

具体方法：

裁剪一条线段时，先求出端点p_1和p_2的编码code1和code2，然后：



(1)若code1|code2=0，对直线段应简取之。

(2)若code1&code2≠0，对直线段可简弃之。

(3)若上述两条件均不成立。则需求出直线段与窗口边界的交点。在交点处把线段一分为二，其中必有一段完全在窗口外，可以弃之。再对另一段重复进行上述处理，直到该线段完全被舍弃或者找到位于窗口内的一段线段为止。

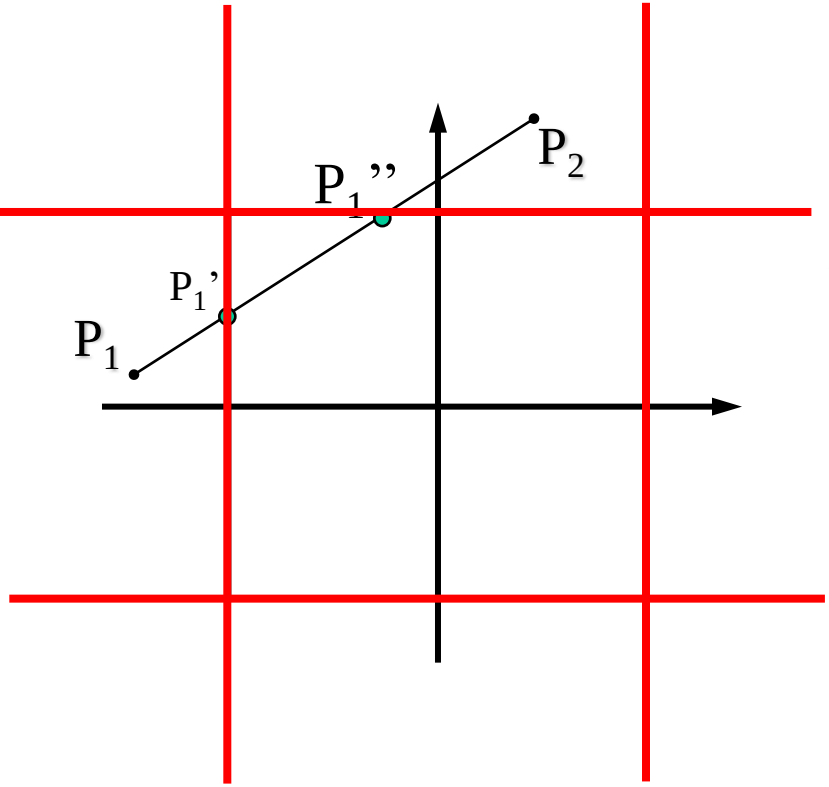
Cohn-Sutherland线段裁剪

- **线段P1(x1,y1)P2(x2,y2)与窗口边界的交点计算**

```
if(LEFT&code !=0)
{     x=XL; y=y1+(y2-y1)*(XL-x1)/(x2-x1);}
else if(RIGHT&code !=0)
{     x=XR; y=y1+(y2-y1)*(XR-x1)/(x2-x1);}
else if(BOTTOM&code !=0)
{     y=YB; x=x1+(x2-x1)*(YB-y1)/(y2-y1);}
else if(TOP & code !=0)
{     y=YT; x=x1+(x2-x1)*(YT-y1)/(y2-y1);}
```

Cohn-Sutherland线段裁剪

示例：



(1) 求P_1 P_2的编码，非完全可见
P_1：(- 3/2, 1/6)；编码 (0001)
P_2：(1/2, 3/2)；　编码 (1000)

(2) 与左边界判断，求交点并舍弃窗口外部分，得P'_1P_2;
P'_1:(-1,1/2)，编码(0000)

(3) 与下边界判断，得 P'_1P_2;

(4) 与右边界判断，得 P'_1P_2;

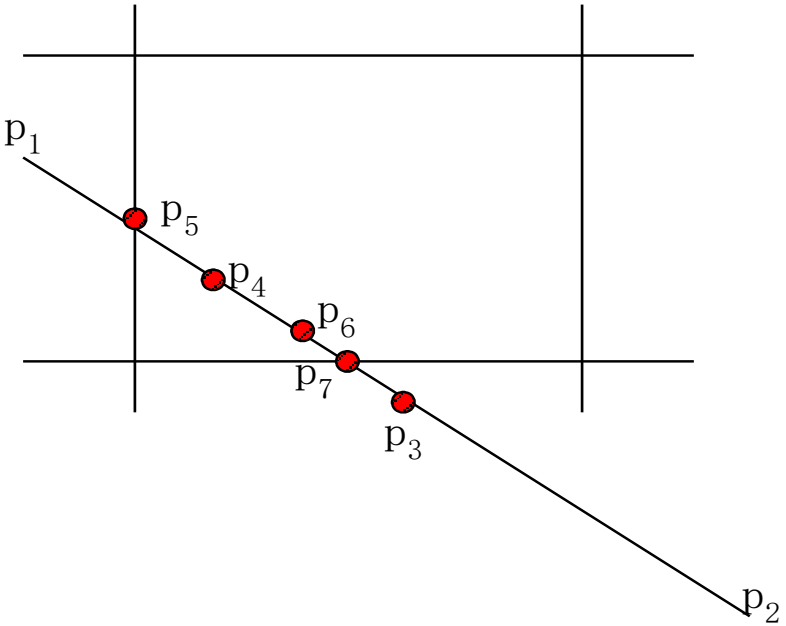
(5) 与上边界判断，求交点并舍弃窗口外部分，得 P''_1(-1/4, 1) P_2(-1, 1/2)；并编码，两端点编码全部为0，线段完全可见，程序结束。

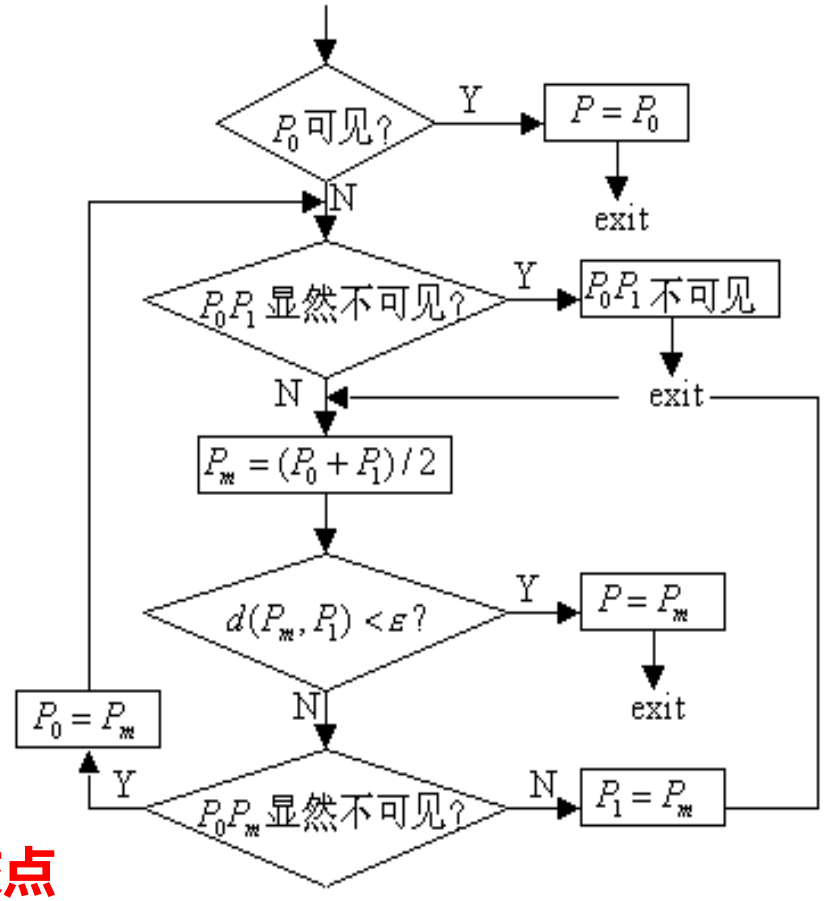
中点分割裁剪

- **算法思想：**从P0点出发找出距P0最近的可见点，从P1点出发找出距P1最近的可见点。

  中点Pm=(P1+P2)/2

- **算法见框图**





核心：二分逼近来确定线段与窗口的交点

中点分割裁剪

算法1——步骤：

(1) 输入直线段的两端点坐标：$p_1(x_1, y_1)$、$p_2(x_2, y_2)$，以及窗口的四条边界坐标：wyt、wyb、wxl和wxr。

(2) 对p_1、p_2进行编码：点p_1的编码为code1，点p_2的编码为code2。

(3) 若code1|code2=0，对直线段应简取之，保留当前直线段的端点坐标，转(5)；否则，若code1&code2≠0，对直线段可简弃之，转(5)；当上述两条均不满足时，进行步骤(4)。

(4) 求出直线段的中点M，将p_1M、p_2M入栈。

(5) 当栈不空时，从栈中弹出一条直线段，取为p_1p_2，转(2)进行处理。否则，继续(6)。

(6) 当栈为空时，合并保留的直线段端点，得到窗口内的直线段p_1p_2。用直线扫描转换算法画出当前的直线段p_1p_2，算法结束。

中点分割裁剪

算法2——步骤：

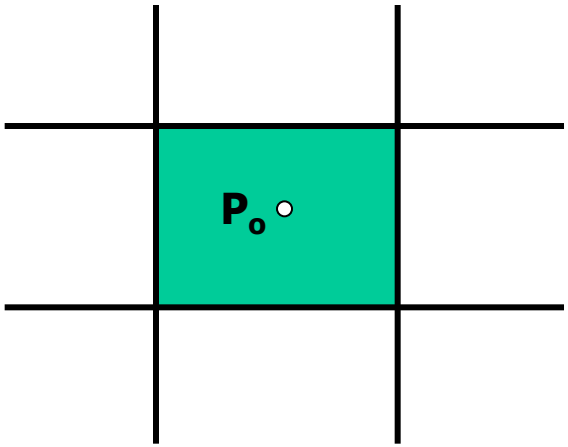
(1) 若 code1|code2=0 ，对直线段应简取之，结束；否则，若 code1&code2≠0，对直线段可简弃之，结束；当这两条均不满足时，进行步骤(2)。

(2)找出该直线段离窗口边界最远的点和该直线段的中点。判中点是否在窗口内：若中点不在窗口内，则把中点和离窗口边界最远点构成的线段丢掉，以线段上的另一点和该中点再构成线段求其中点；如中点在窗口内，则又以中点和最远点构成线段，并求其中点，直到中点与窗口边界的坐标值在规定的误差范围内相等，则该中点就是该线段落在窗口内的一个端点坐标。

(3)如另一点在窗口内，则经(2)即确定了该线段在窗口内的部分。如另一点不在窗口内，则该点和所求出的在窗口上的那一点构成一条线段，重复步骤(2)，即可求出落在窗口内的另一点。

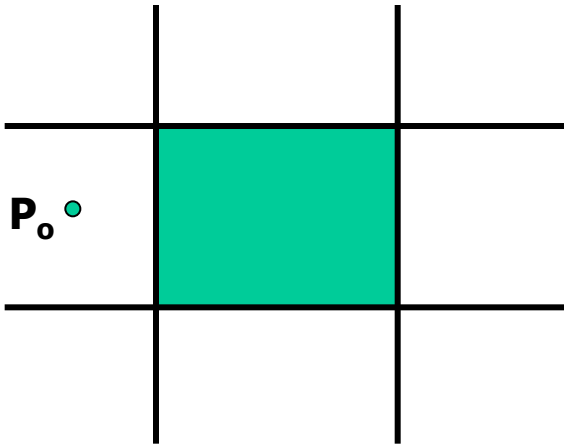
Nicholl-Lee-Nicholl直线裁剪

- **算法思想：**划分更多区域来减少求交运算
- **端点与裁剪窗口位置关系：**



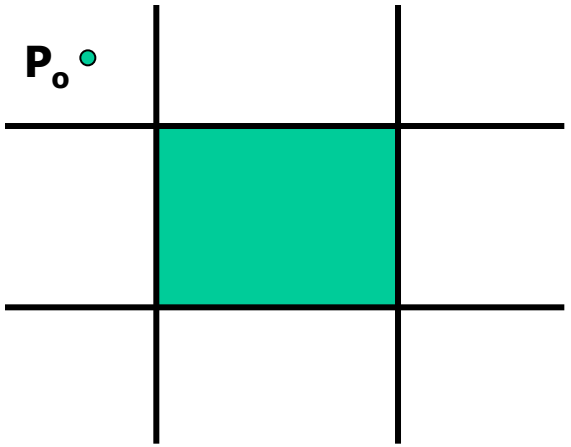
P_0 在裁剪窗口内

(a)



P_0 在裁剪窗口的一边界外

(b)

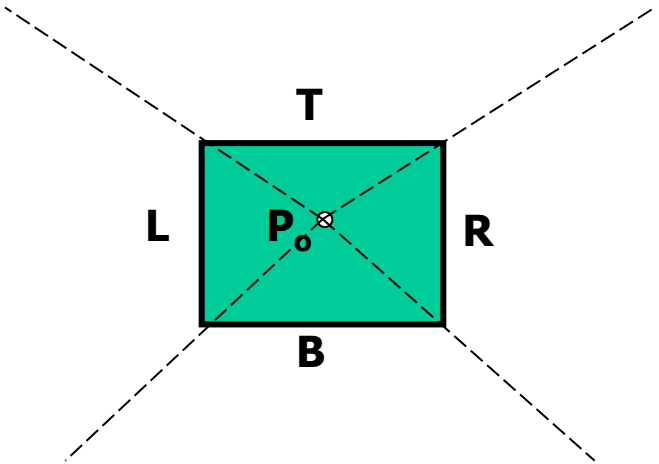


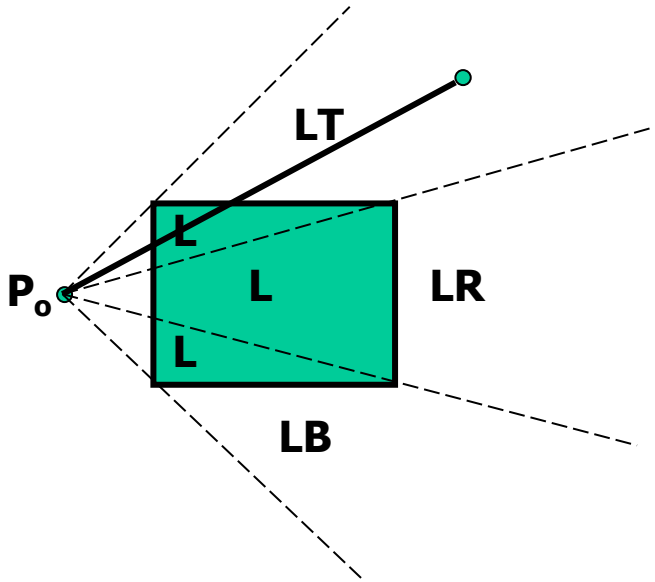
P_0 在裁剪窗口的一角

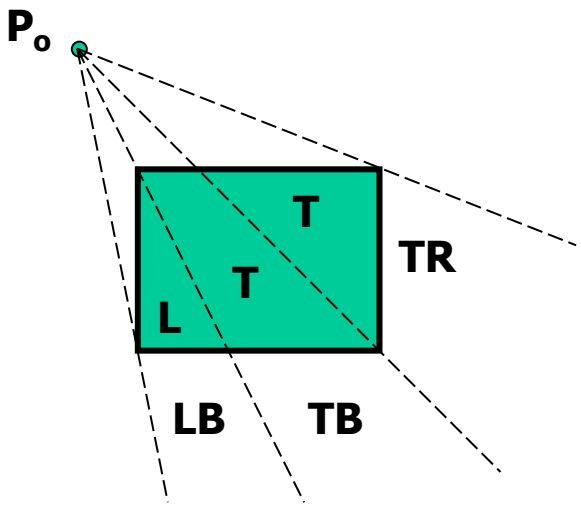
(c)

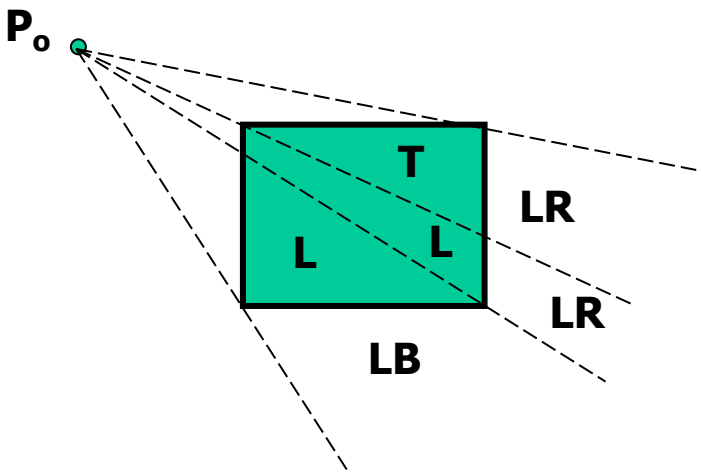
Nicholl-Lee-Nicholl直线裁剪

- 端点的裁剪分区关系：









Nicholl-Lee-Nicholl直线裁剪

- **算法步骤：**

(1) 将窗口所在直线划分二维平面为9个区域，判断P1在(a)(b)(c)哪个区域；

(2) 从P1点向窗口4个角点引射线，射线与窗口直线将二维平面划分更多小区域；

(3) 确定P2所在区域；

(4) 求交点，确定P1P2的可见部分。

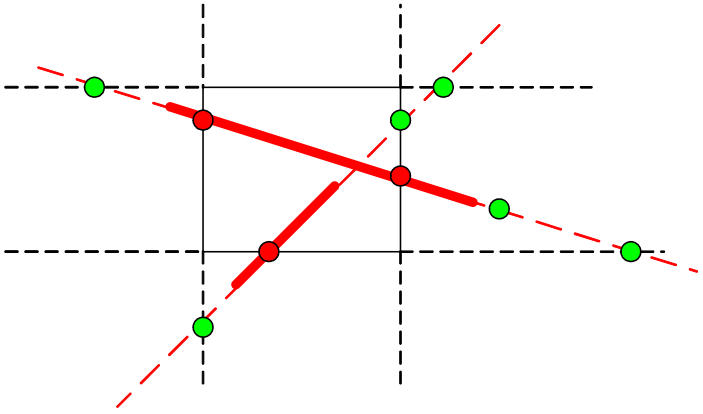
思考：如何判断P1P2在哪个小区域？

Liang-Barsky直线裁剪

- **直线参数方程：**

$$x = x_1 + u\Delta x \quad 0 \le u \le 1$$

$$y = y_1 + u\Delta y$$



- **参数化形式裁剪条件：**

$$XL \le x_1 + u\Delta x \le XR$$

$$YB \le y_1 + u\Delta y \le YT$$

可以统一表示为形式： $up_k \le q_k$

$$p_1 = -\Delta x \qquad q_1 = x_1 - XL \qquad p_2 = \Delta x \qquad q_2 = XR - x_1$$

$$p_3 = -\Delta y \qquad q_3 = y_1 - YB \qquad p_4 = \Delta y \qquad q_4 = YT - y_1$$

入边 **出边**

Liang-Barsky直线裁剪

- 当 p_k=0 （ $\Delta x = 0$或$\Delta y = 0$），
- q_k<0，则线段完全在边界外， $q_k \geq 0$，则该线段平行于裁剪边界并且在窗口内。

- 当 $p_k \neq 0$，
  - 当 q_k<0,线段从裁剪边界延长线的外部延伸到内部。
  - 当 q_k>0,线段从裁剪边界延长线的内部延伸到外部。

$$p_1 = -\Delta x \qquad q_1 = x_1 - XL \qquad p_2 = \Delta x \qquad q_2 = XR - x_1$$

$$p_3 = -\Delta y \qquad q_3 = y_1 - YB \qquad p_4 = \Delta y \qquad q_4 = YT - y_1$$

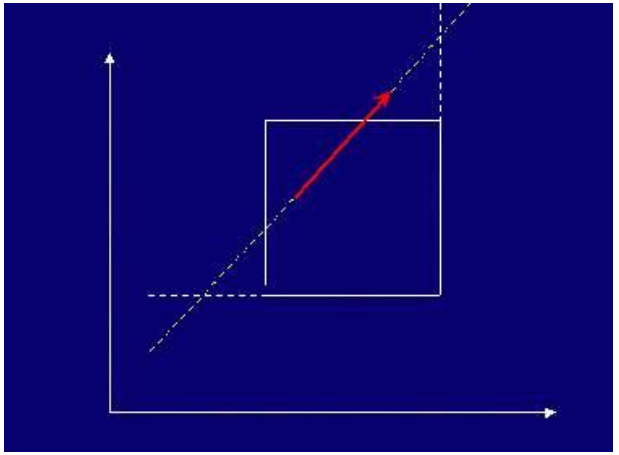
Liang-Barsky直线裁剪

- **算法思路：**

对每条直线计算出参数u1和u2，其定义了在裁剪矩形内的线段部分

- u1的值由线段从外到内遇到的矩形边界所决定（p<0）。对这些边界计算$r_k=q_k/p_k$。u1取0和各个r_k值之中的最大值。
- u2的值由线段从内到外遇到的矩形边界所决定（p>0）。对这些边界计算$r_k=q_k/p_k$。u2取1和各个r_k值之中的最小值。

- 如果u1>u2，则线段完全落在裁剪窗口之外，被舍弃。
- 否则裁剪线段由参数u的两个值u1,u2计算出来。



Liang-Barsky直线裁剪

算法步骤：

(1)输入直线段的两端点坐标：(x_1,y_1)和(x_2,y_2)，以及窗口的四条边界坐标：wyt、wyb、wxl和wxr。

(2)若$\Delta x=0$，则$p_1=p_2=0$。此时进一步判断是否满足$q_1<0$或$q_2<0$，若满足，则该直线段不在窗口内，算法转(7)。否则，满足$q_1>0$且$q_2>0$，则进一步**计算u_1和u_2**。算法转(5)。

(3)若$\Delta y=0$，则$p_3=p_4=0$。此时进一步判断是否满足$q_3<0$或$q_4<0$，若满足，则该直线段不在窗口内，算法转(7)。否则，满足$q_1>0$且$q_2>0$，则进一步**计算u_1和u_2**。算法转(5)。

(4)若上述两条均不满足，则有$p_k\neq0$（k=1,2,3,4）。此时**计算u_1和u_2**。

(5)求得u_1和u_2后，进行判断：若$u_1>u_2$，则直线段在窗口外，算法转(7)。若$u_1<u_2$，利用直线的参数方程求得直线段在窗口内的两端点坐标。

(6) 绘制在窗口内的直线段。

(7)算法结束。

Liang-Barsky直线裁剪

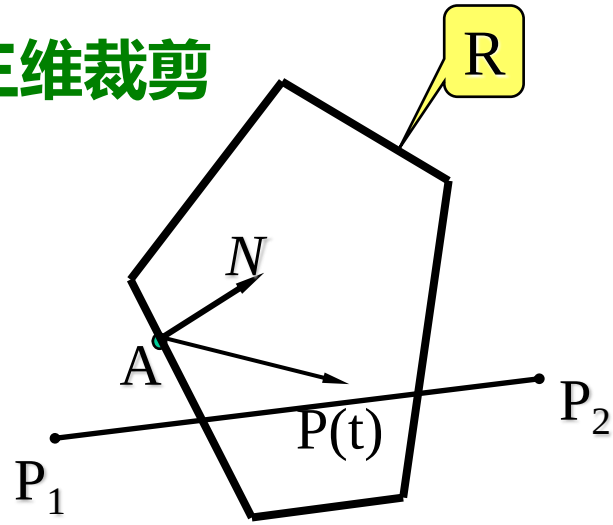
算法代码：

```
void
    LB_LineClip(x1,y1,x2,y2,XL,XR,YB,Y
    T)
float x1,y1,x2,y2,XL,XR,YB,YT;
{   float dx,dy,u1,u2;
    u1=0;u2=1 ;     dx =x2-x1;dy =y2-y1;
    if(ClipT(-dx,x1-XL,&u1,&u2)
      if(ClipT(dx,XR-x1, &u1,&u2)
          if(ClipT(-dy,y1-YB, &u1,&u2)
           if(ClipT(dy,YT-y1, &u1,&u2)
             {
                    displayline(x1+u1*dx
             ,y1+u1*dy,
             x1+u2*dx,y1+u2*dy)
                    return;
              }
 }
```

```
bool ClipT(p,q,u1,u2)
float p,q,*u1,*u2;
{  float r;
    if(p<0)
    {     r=q/p;
          if(r>*u2)  return FALSE;
          else if(r>*u1)
          { *u1=r; return TRUE; }
    }
    else if(p>0)
    {     r=p/q;
            if(r<*u1)return FALSE;
            else if(r<*u2)
              {     *u2=r;return TRUE;}
          }
    else if(q<0) return FALSE;
   return TRUE;
}
```

Cyrus-Beck直线裁剪

特点：可对任意凸多边形窗口实现二维和三维裁剪



- 考虑一个凸多边形 R 和一个线段 P_1P_2，
  P_1P_2 与 R 最多只有两个交点
- 设 A 是 R 边界上一点，N 是该区域边界
  在 A 点的内法向量
- 将 P_1P_2 用参数方程表示：$P(t) = (P_2 - P_1)\,t + P_1$
  则线段上任一点 $P(t)$, 与 N 的点积有三种可能

(1) P(t) 在多边形外侧：$N \cdot (P(t) - A) < 0$

(2) P(t) 在多边形的边及其延长线上：$N \cdot (P(t) - A) = 0$

(3) P(t) 在多边形内侧：$N \cdot (P(t) - A) > 0$

Cyrus-Beck直线裁剪

P(t)在凸多边形内的充要条件是：

对凸多边形边界上任意一点A和该处内法向量*N*，有：

$N \bullet (P(t) - A) > 0$

- 设多边形有 *M* 条边， 在每条边上取一个点 A_i 和该点的法向量 N_i，则可见线段的参数区间为下列不等式的解

$$N_i \bullet (P(t) - A_i) \geq 0 \qquad (i = 1，2，\ldots\ldots M)$$

$$0 \leq t \leq 1$$

- 上述可求得满足条件的一系列 *t* 值，在实际中只需最大和最小值，它们对应可见线段区间的端点。

Cyrus-Beck直线裁剪

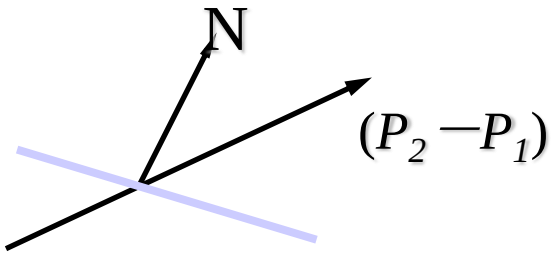
- **具体求解 t，将线段方程代入判别式：**

$$\begin{cases} N_i \bullet (P_1 + (P_2 - P_1)t - A_i) \geq 0 \\ 0 \leq t \leq 1 \end{cases} \Rightarrow \begin{cases} N_i \bullet (P_1 - A_i) + N_i \bullet (P_2 - P_1)t \geq 0 \\ 0 \leq t \leq 1 \end{cases}$$

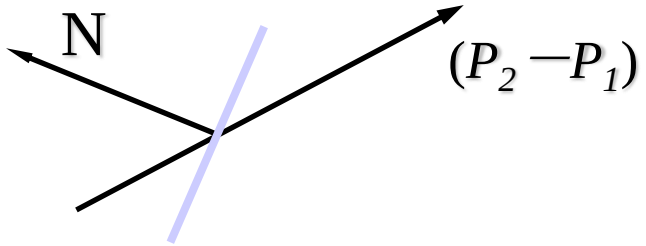
(1) 当 $N_i \cdot (P_2 - P_1) = 0$ 时， N_i 垂直于 $(P_2 - P_1)$ ，第 i 条边与 P_1P_2 平行，无交点

(2) 当 $N_i \cdot (P_2 - P_1) > 0$ 时， P_1 在该边外侧 ，可求出交点 t_i ，并将其归入下限组

(3) 当 $N_i \cdot (P_2 - P_1) < 0$ 时， P_2 在该边外侧 ，可求出交点 t_i ，并将其归入上限组



$N_i (P_2 - P_1) > 0$ 情况



$N_i (P_2 - P_1) < 0$ 情况

Cyrus-Beck直线裁剪

(4) 前述判别式 = 0 对应线段与边的交点，因此当 $N_i(P_2 - P_1) \neq 0$ 时可求出 t 为:

$$t_i = -\frac{N_i \bullet (P_1 - A_i)}{N_i \bullet (P_2 - P_1)}$$

$$N_i \bullet (P_1 - A_i) + N_i \bullet (P_2 - P_1)t = 0$$

$$0 \leq t \leq 1$$

(5) **可见线段**为下限组中的最大值至上限组中的最小值之间的线段

即：

$$t_l = \max\{0, \max\{t_i : N_i \bullet (P_2 - P_1) > 0\}\}$$

$$t_h = \min\{1, \min\{t_i : N_i \bullet (P_2 - P_1) < 0\}\}$$

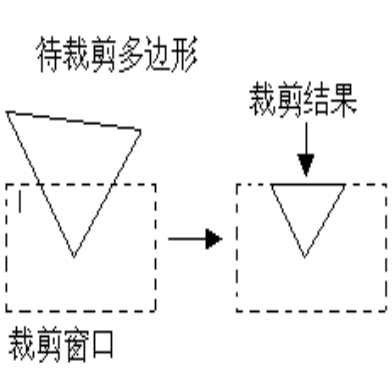
(6) 当 $t_l < t_h$，则 t_l 和 t_h 是可见线段的端点参数，否则线段在区域之外

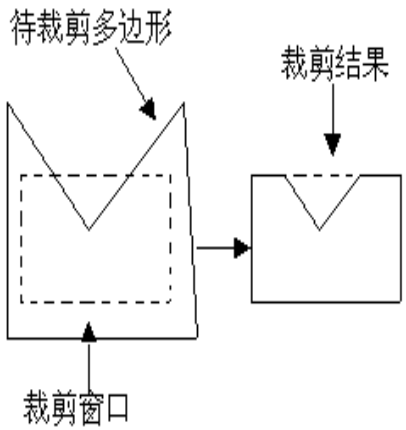
该方法可以求出任意凸多边形对线段的裁剪

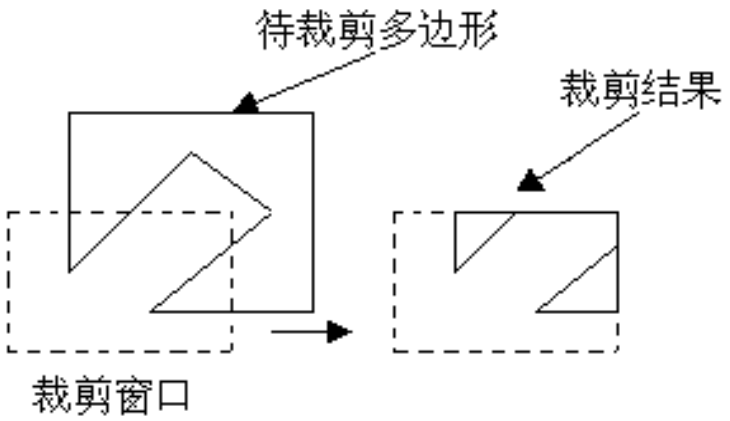
多边形裁剪

- **错觉**：直线段裁剪的组合？
- **问题**：1）边界不再封闭，需要用窗口边界的恰当部分来封闭它，如何确定其边界？

  2）一个凹多边形可能被裁剪成几个小的多边形，如何确定这些小多边形的边界？

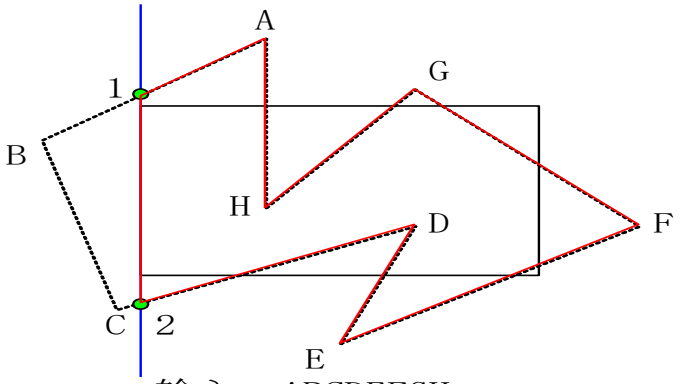




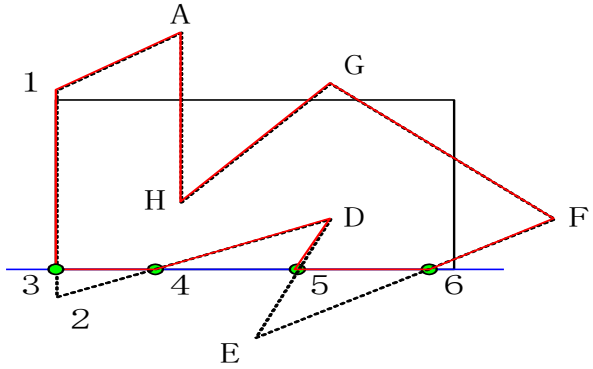


Sutherland-Hodgeman多边形裁剪

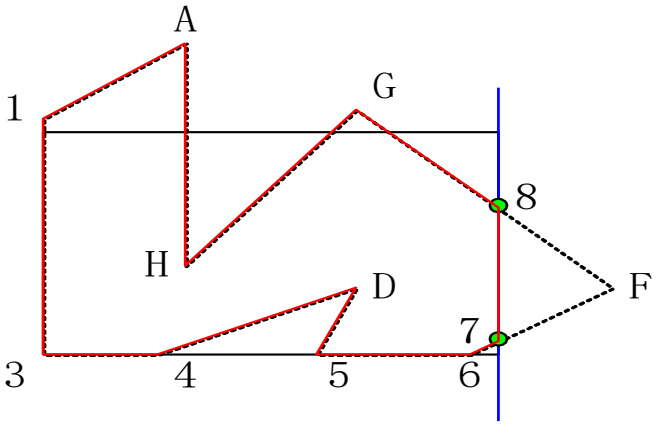
算法思想：



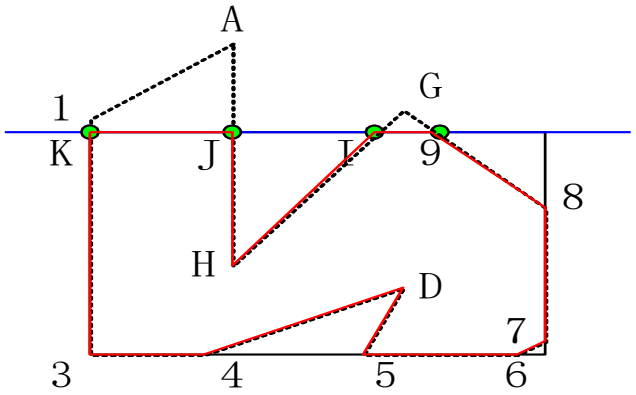
输入： ABCDEFGH
输出： A12DEFGH
(a)用左边界裁剪



输入： A12DEFGH
输出： A134D56FGH
(b)用下边界裁剪



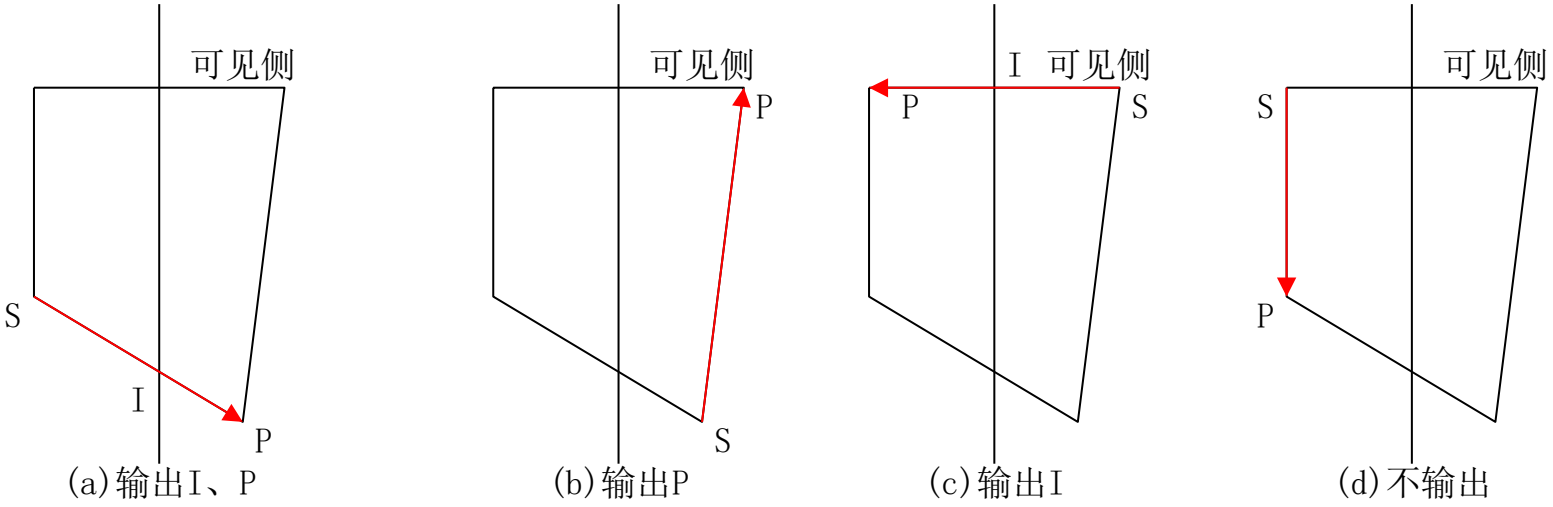
输入： A134D56FGH
输出： A134D5678GH
(c)用右边界裁剪



输入： A134D5678GH
输出： K34D56789IHJ
(d)用上边界裁剪

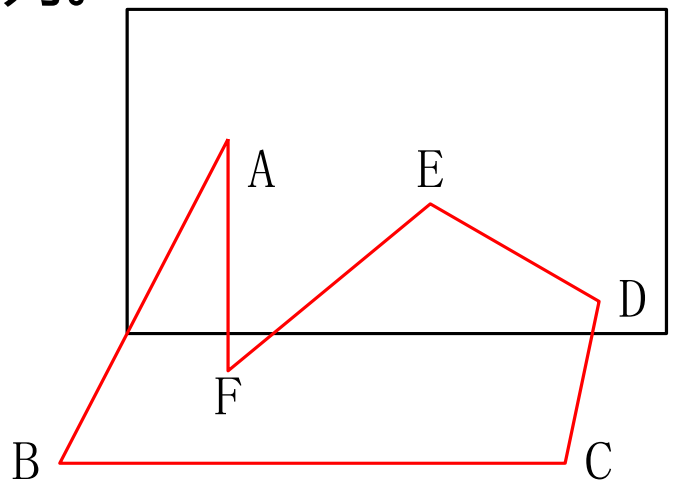
Sutherland-Hodgeman多边形裁剪

- **沿着多边形依次处理顶点情况：**
  - (1) 由不可见到可见：输出与裁剪线的交点及终点
  - (2) 位于可见一侧：输出终点，作为新多边形顶点
  - (3) 由可见到不可见：输出与裁剪线的交点
  - (4) 位于不可见一侧：不输出

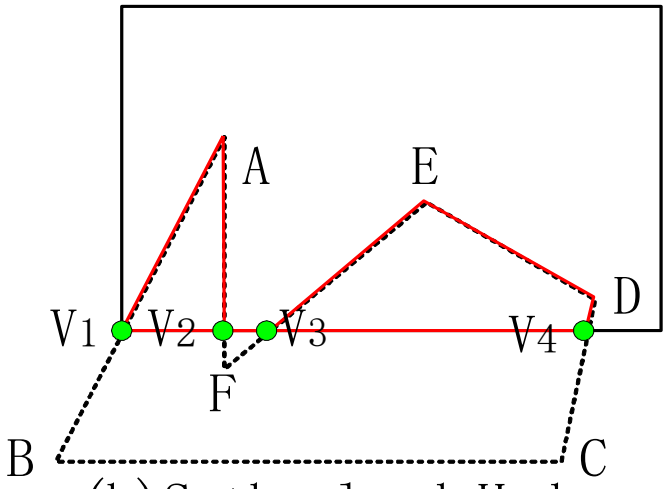


Sutherland-Hodgeman多边形裁剪

- **算法思路：**
  - 每条裁剪边在裁剪处理完所有顶点后，其输出顶点表将用窗口的下一条边界继续裁剪。
  - 窗口的一条边以及延长线构成的裁剪线把平面分为两个区域（可见侧，不可见侧），利用上述方法更新顶点序列。



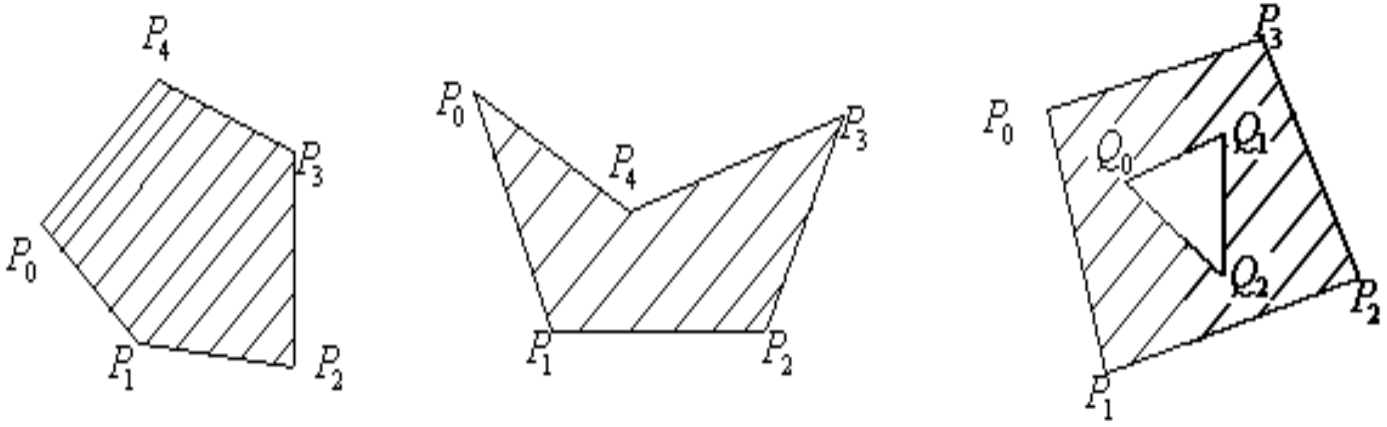
(a)裁剪前



(b)Sutherland-Hodgeman算法的裁剪结果

Weler-Athenton多边形裁剪

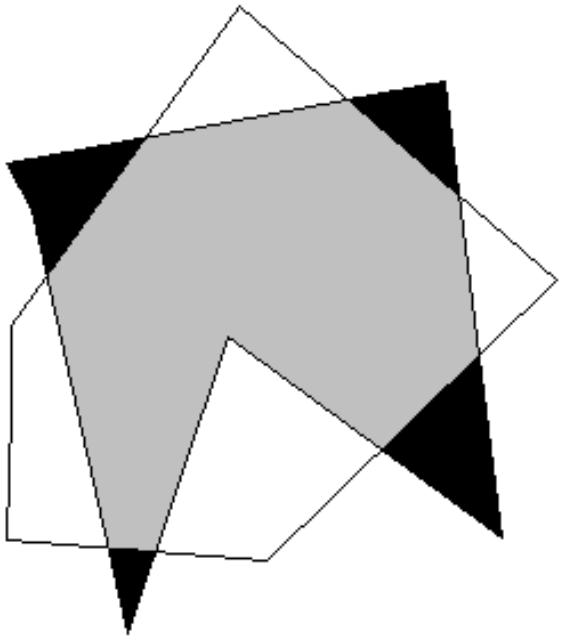
- **任意多边形裁剪：**裁剪窗口为任意多边形（凸、凹、带内环）的情况：
  - 主多边形：被裁剪多边形，记为A
  - 裁剪多边形：裁剪窗口，记为B



Weler-Athenton多边形裁剪

- **裁剪形式**
  - 内裁剪：A∩B
  - 外裁剪：A-B
- **特点：**

  裁剪结果区域的边界由A的部分边界和B的部分边界两部分构成，并且在交点处边界发生交替。即**由A的边界转至B的边界，或由B的边界转至A的边界**。

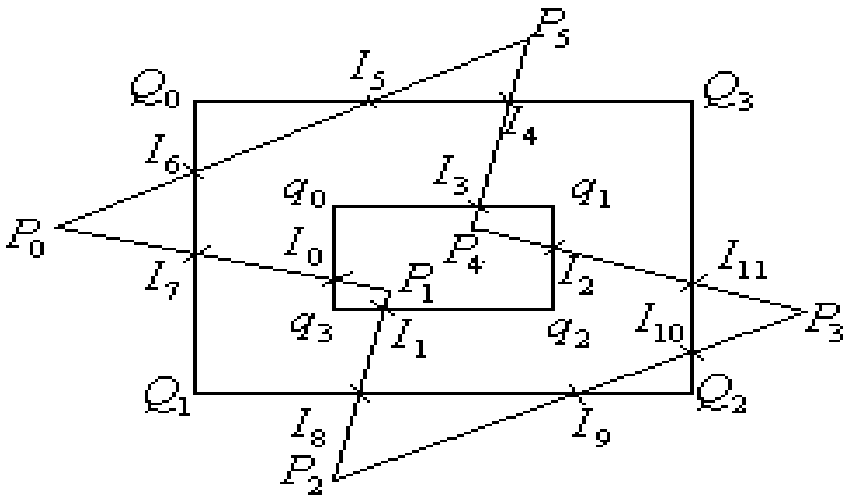


Weler-Athenton多边形裁剪

- **分析：**

主多边形与裁剪多边形有交点，则交点成对出现，分为两类：

- **进点**：主多边形边界由此进入裁剪多边形内
  如，I1,I3, I5, I7, I9, I11
- **出点**：主多边形边界由此离开裁剪多边形区域.
  如， I0,I2, I4, I6, I8, I10



Weler-Athenton多边形裁剪

- **算法（内裁剪）步骤：**

1、建立主多边形和裁剪多边的顶点表。

2、求主多边形和裁剪多边形的交点，并按顺序插入两多边形的顶点表中。

3、裁剪。如果存在没有被跟踪过的交点，执行以下步骤：

(1)建立裁剪结果多边形的顶点表。

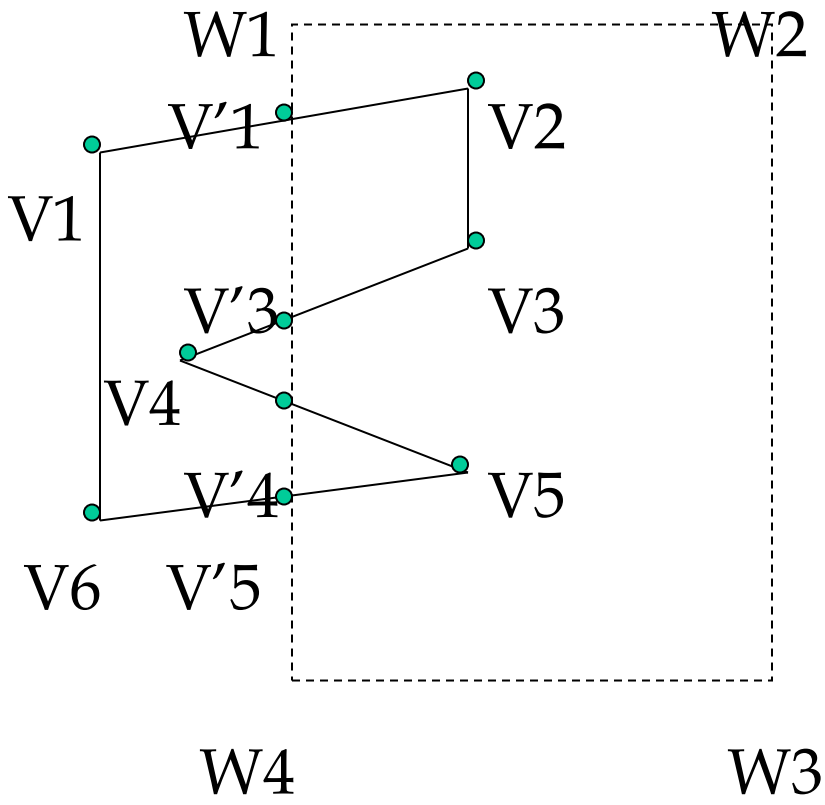
(2)选取任一没有被跟踪过的交点为始点，存入结果多边形顶点表中。

(3)若该交点为进点，跟踪主多边形边边界；否则跟踪裁剪多边形边界。

(4) 跟踪多边形边界，每遇到多边形顶点存入结果多边形顶点表中，直至遇到新的交点。

Weler-Athenton多边形裁剪

- 示例：



Ⅰ：V1,V2,V3,V4,V5,V6,V1

Ⅱ：W1,W2,W3,W4,W1

Ⅲ：V1,V'1,V2,V3,V'3,V4,
V'4,V5,V'5,V6,V1

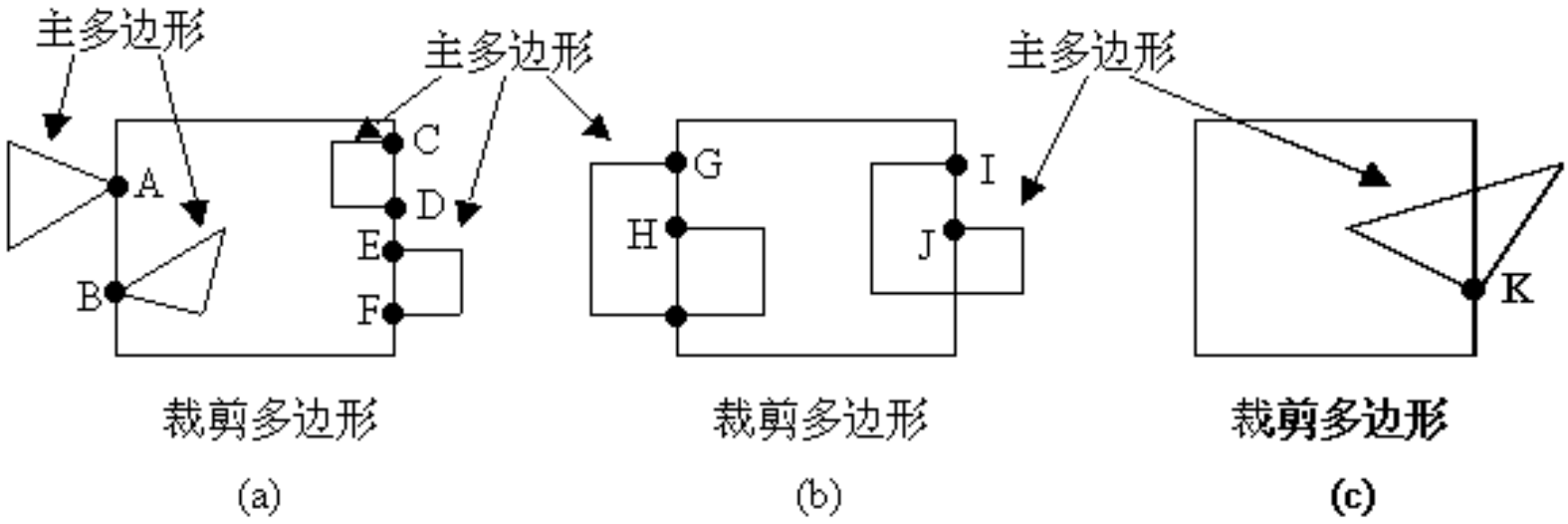
Ⅳ：W1,W2,W3,W4,V'5,V'4,
V'3,V'1,W1

Q1：V'1,V2,V3,V'3,V'1

Q2：V'4,V5,V'5,V'4

Weler-Athenton多边形裁剪

- **交点的奇异情况处理**
  - **与裁剪多边形边重合**的主多边形的边不参与求交点；
  - 对于**顶点落在裁剪多边形的边上**的主多边的边，如果落在该裁剪边的内侧，将该顶点算作交点；而如果这条边落在该裁剪边的外侧，将该顶点不看作交点。



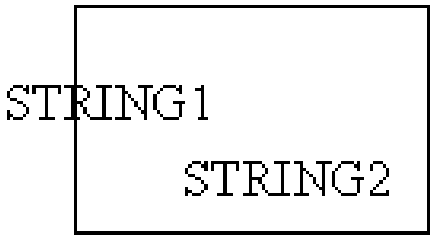
字符裁剪

1)串精度裁剪

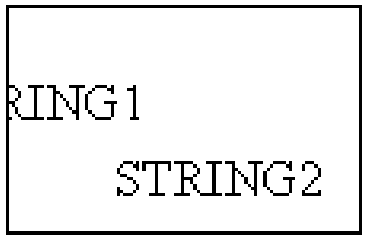
2)字符精度裁剪

字符裁剪

- **3)笔划、象素精度裁剪**
  - 点阵字符：点裁剪
  - 矢量字符：线裁剪



(a)



(b)

思考

1、请描述线段对裁剪窗口是任意多边形的算法。

谢　谢！