

第五章 图形变换与裁剪 (三)

5.5 二维线段裁剪

苏小红

计算机科学与技术学院

哈尔滨工业大学

5.5 二维线段裁剪

1 直线段裁剪

- ∞ 直接求交算法
- ∞ Cohen-Sutherland算法
- ∞ 中点分割裁剪算法
- ∞ 梁友栋-Barsky算法

2 多边形裁剪

- ∞ Sutherland_Hodgman算法
- ∞ Weiler-Atherton算法

1. 直线段裁剪(1/18)

❖ 裁剪（clipping）的目的

- ❧ 判断图形元素是否在裁剪窗口之内并找出其位于内部的部分

❖ 裁剪处理的基础

- ❧ 图元关于窗口内外关系的判别
- ❧ 图元与窗口的求交

❖ 裁剪与覆盖的区别

1. 直线段裁剪(2/18)

❖ 裁剪窗口

❧ 矩形、圆形、一般多边形

❖ 被裁剪对象

❧ 线段、多边形、曲线、字符

❖ 设计裁剪算法的核心问题

❧ 效率高，速度快

1. 直线段裁剪(3/18)

❖ 把直线当作点的集合，逐点裁剪

∞ 点 (x, y) 在窗口内的充分必要条件是：

$$x_{\min} \leq x \leq x_{\max}$$

$$y_{\min} \leq y \leq y_{\max}$$

问题：极其费时，精度不高。

1. 直线段裁剪(4/18)

❖ 把直线当作一个整体来裁剪

∞ 矩形裁剪窗口:

$$[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$$

∞ 待裁剪线段:

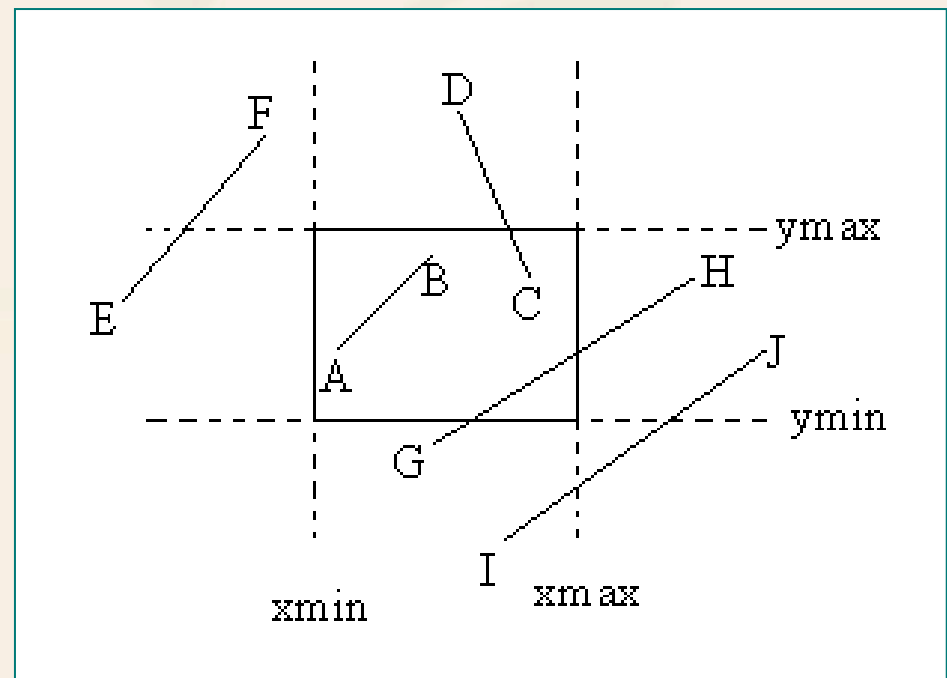
$$P_0(x_0, y_0)P_1(x_1, y_1)$$

❖ 前提: 任何平面线段在凸多边形窗口进行裁剪后, 落在窗口内的线段不会多于1条。

1. 直线段裁剪(5/18)

❖ 待裁剪线段和窗口的关系

1. 完全落在窗口内,
2. 完全落在窗口外,
3. 部分在内, 部分在外.



5.5 二维线段裁剪

1 直线段裁剪

∞ 直接求交算法

∞ Cohen-Sutherland算法

∞ 中点分割裁剪算法

∞ 梁友栋-Barsky算法

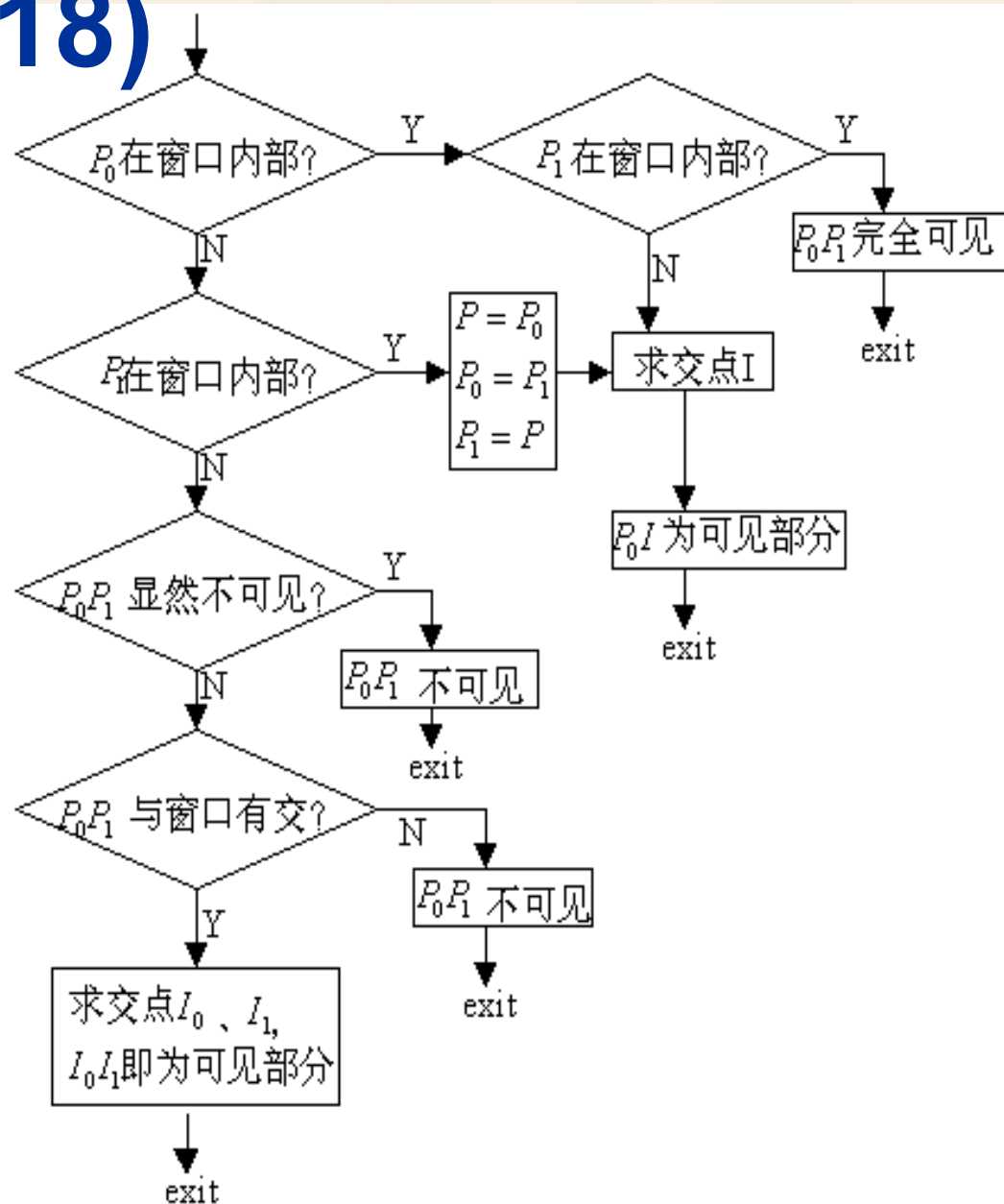
1. 直线段裁剪(6/18)

直接求交算法

直线与窗口边都

写成参数形式，

求参数值。



5.5 二维线段裁剪

1 直线段裁剪

∞ 直接求交算法

∞ **Cohen-Sutherland**算法

∞ 中点分割裁剪算法

∞ 梁友栋-Barsky算法

1. 直线段裁剪(7/18)

Cohen-Sutherland 算法

❖ 待裁剪线段和窗口的关系

1. 完全落在窗口内,
2. 完全落在窗口外,
3. 部分在内, 部分在外.

为提高效率, 该算法强调:

- 快速判断情形(1)(2);
- 减少情形(3)的求交次数和求交所需的计算量。

1. 直线段裁剪(8/18)

Cohen-Sutherland 算法

算法步骤:

1. 判别线段两端点是否都落在窗口内，如果是，则线段完全可见，转至第4步；
2. 判别线段是否为显然不可见，如果是，则裁剪结束，转至第4步；
3. 求线段与窗口边延长线的交点，这个交点将线段分为两段，其中一段显然不可见，丢弃。对余下的另一段重新进行第1步处理，
4. 结束

裁剪过程是递归的。

1. 直线段裁剪(9/18)

Cohen-Sutherland 算法

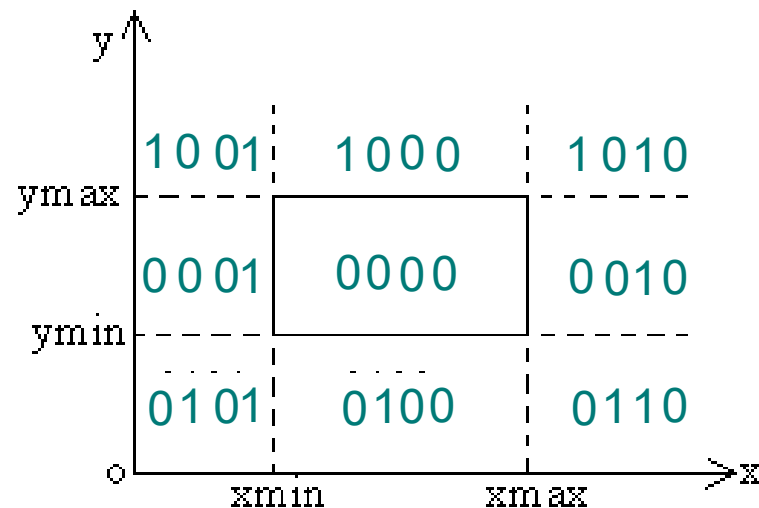
❖ 关键问题:

∞ 如何快速判别 完全可见 和 完全不可见 线段?

❖ 解决方法 —— 编码:

∞ 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, $C_t C_b C_r C_l$, 上下右左;

$$C_t = \begin{cases} 1 & , y > y_{\max} \\ 0 & , \text{else} \end{cases} \quad C_b = \begin{cases} 1 & , y < y_{\min} \\ 0 & , \text{else} \end{cases}$$
$$C_r = \begin{cases} 1 & , x > x_{\max} \\ 0 & , \text{else} \end{cases} \quad C_l = \begin{cases} 1 & , x < x_{\min} \\ 0 & , \text{else} \end{cases}$$



1. 直线段裁剪(10/18)

Cohen-Sutherland 算法

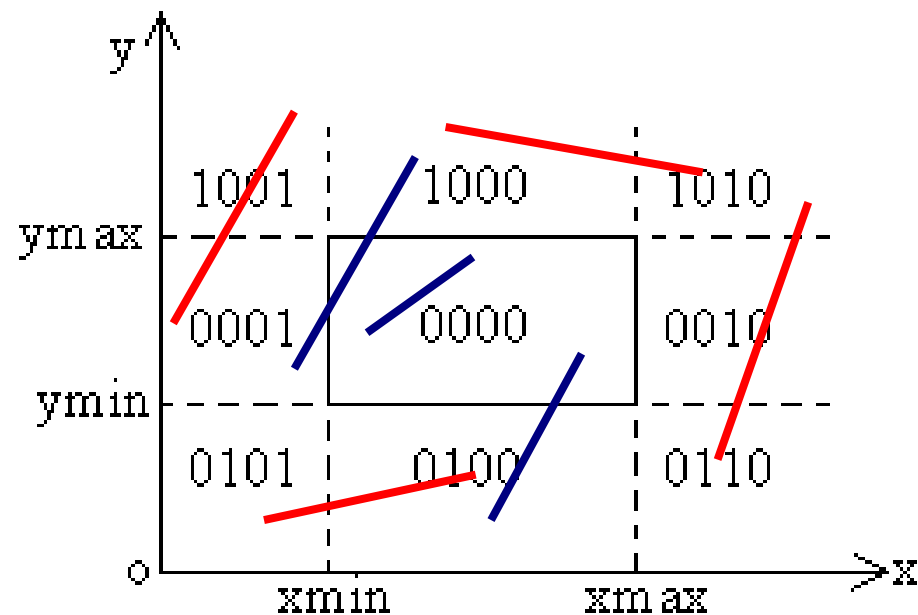
❖ 所以也称为编码裁剪算法

❖ 端点编码:

⌘ 定义为它所在区域的编码

❖ 快速判断“完全不可见”

⌘ 线段两端点编码的逻辑“与”运算结果非零，则完全不可见。



1. 直线段裁剪(11/18)

Cohen-Sutherland 算法

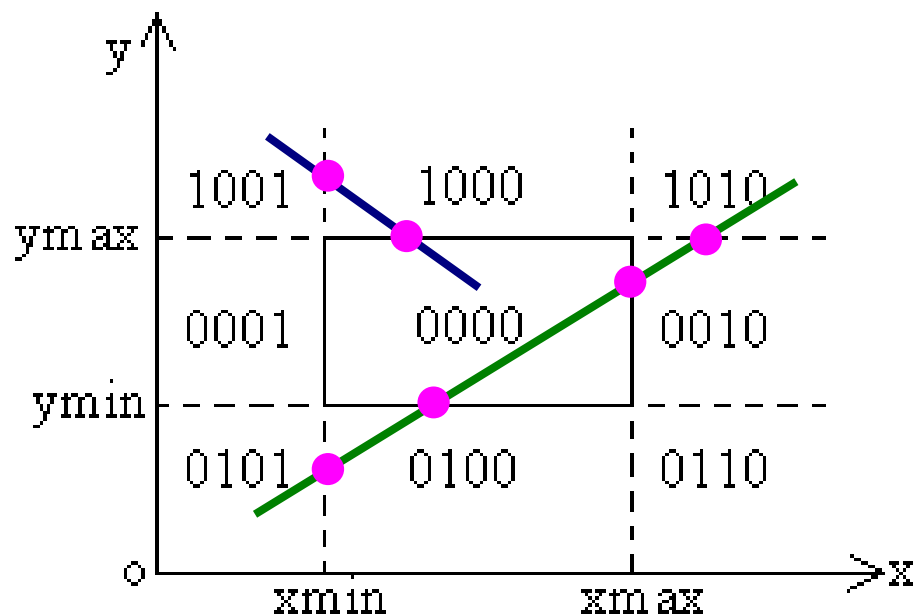
对于那些部分可见又部分不可见的线段，需要求交，求交前先测试与窗口哪条边所在直线有交？

➤ 逐个端点判断其编码

$C_l C_t C_r C_b$ 中各位是否为“1”，若是，则需求交。

➤ 最坏情形：

线段求交四次。



1. 直线段裁剪(12/18)

Cohen-Sutherland 算法

1) 特点:

用编码方法可快速判断线段——

完全可见 或 完全不可见。

2) 特别适用两种场合:

① 大窗口场合;

② 窗口特别小的场合

(如: 光标拾取图形时, 光标看作小的裁剪窗口)

5.5 二维线段裁剪

1 直线段裁剪

∞ 直接求交算法

∞ Cohen-Sutherland算法

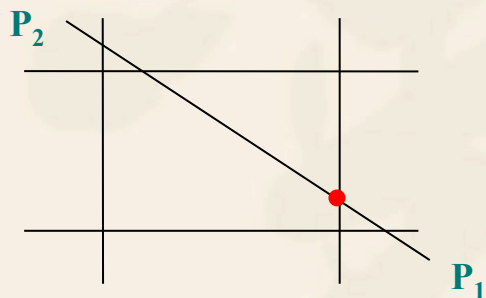
∞ 中点分割裁剪算法

∞ 梁友栋-Barsky算法

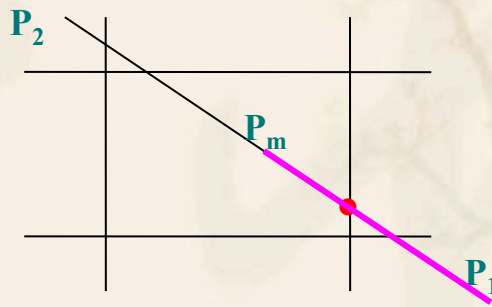
1. 直线段裁剪(13/18)

中点分割法

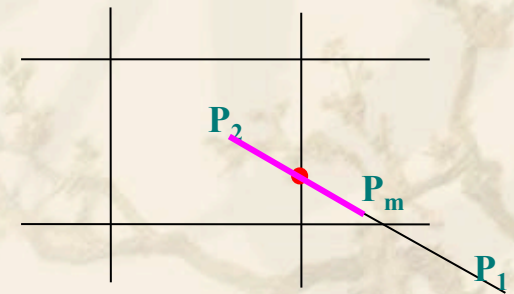
- ❖ 基本思想：利用对分搜索思想搜索交点
 - 从 P_1 点出发找出距 P_1 最近的可见点
 - 从 P_2 点出发找出距 P_2 最近的可见点
 - 不断地在中点处将线段一分为二，对每段线段重复 **Cohen-Sutherland** 裁剪算法的线段可见性测试方法，直至找到每段线段与窗口边界线的交点或分割子段的长度充分小可视为一点为止
- ❖ 取中点 $P_m = (P_1 + P_2) / 2$ 。



从 P_1 点出发找距 P_1 最近的可见点



用 P_1P_m 代替 P_1P_2



用 P_mP_2 代替 P_1P_2

1. 直线段裁剪(13/18)

中点分割法

❖ 优点:

- ❧ 算法原理和编码裁剪是一致的，不同之处在于用移位运算代替求交计算
- ❧ 适合硬件实现

5.5 二维线段裁剪

1 直线段裁剪

∞ 直接求交算法

∞ Cohen-Sutherland算法

∞ 中点分割裁剪算法

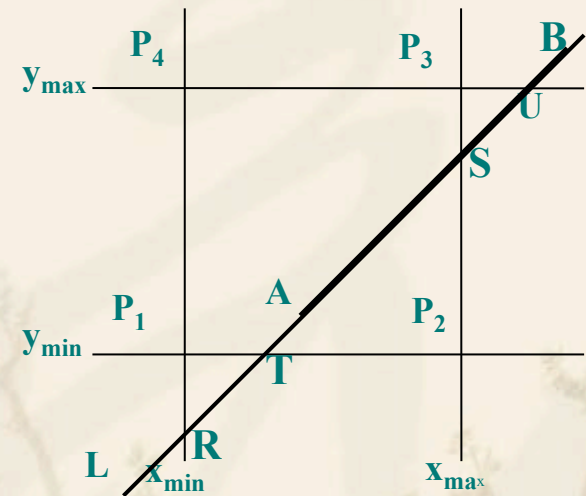
∞ 梁友栋-Barsky算法

1. 直线段裁剪(14/18)

Liang-Barsky裁剪算法

❖ 基本思想:

∞ 把二维裁剪化为一维裁剪
问题，并向 x （或 y ）方向
投影以决定可见线段

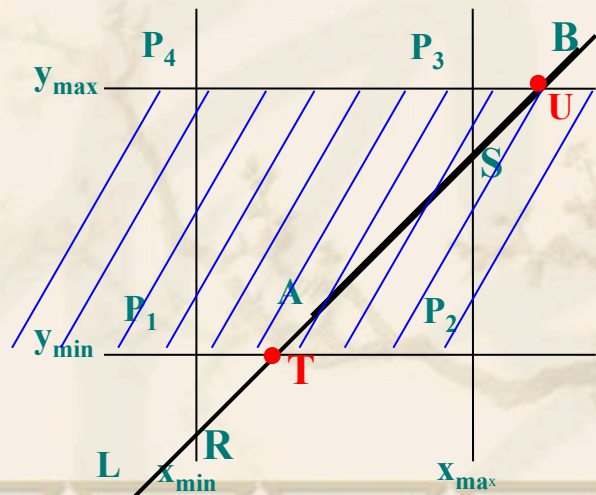
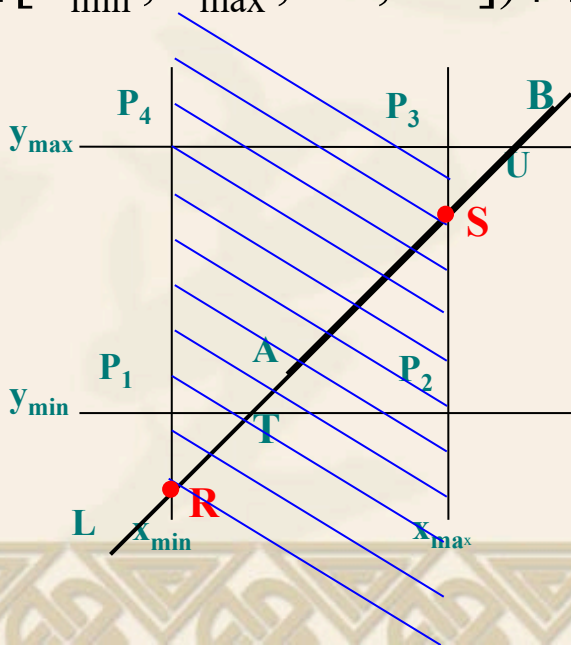


AS是一维窗口TS中的可见部分

1. 直线段裁剪(15/18)

- ❖ 当 Q 为空集时，线段 AB 不可能在窗口中有可见线段。
- ❖ 当 Q 不为空集时， Q 可看成是一个一维窗口
- ❖ 直线 L 与区域的交：

$$\begin{aligned} Q &= L \cap \square P_1 P_2 P_3 P_4 = L \cap [x_{\min}, x_{\max}; -\infty, +\infty] \cap [-\infty, +\infty; y_{\min}, y_{\max}] \\ &= (L \cap [x_{\min}, x_{\max}; -\infty, +\infty]) \cap (L \cap [-\infty, +\infty; y_{\min}, y_{\max}]) = RS \cap TU \end{aligned}$$

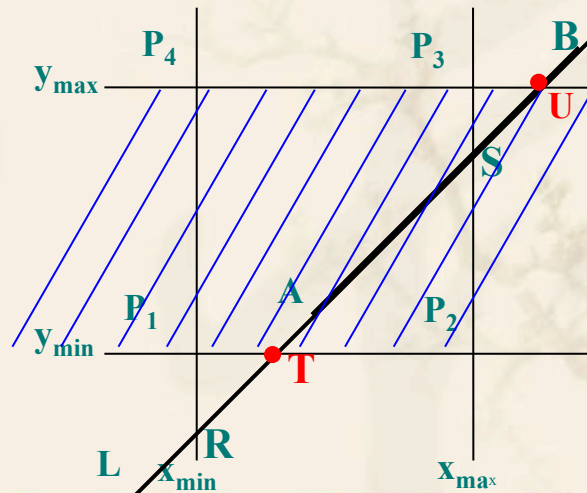
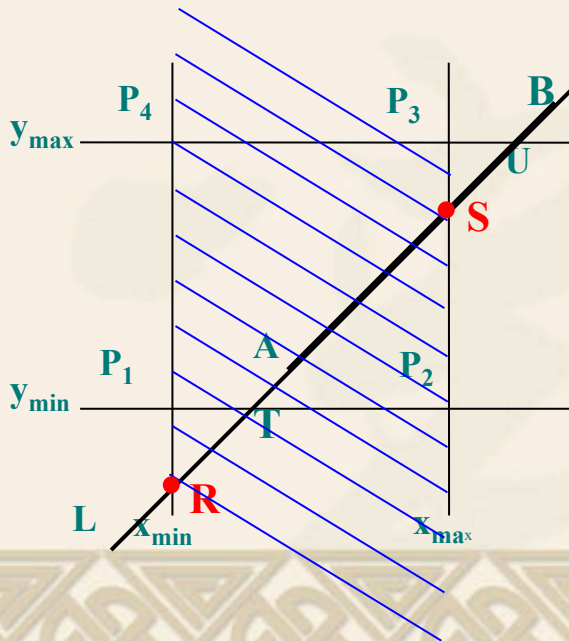


1. 直线段裁剪(16/18)

- ❖ 当 Q 为空集时，线段 AB 不可能在窗口中有可见线段。
- ❖ 当 Q 不为空集时， Q 可看成是一个一维窗口。

❖ 存在可见线段的充要条件

$\Leftrightarrow AB \cap Q$ 即 $AB \cap RS \cap TU$ 不为空集。

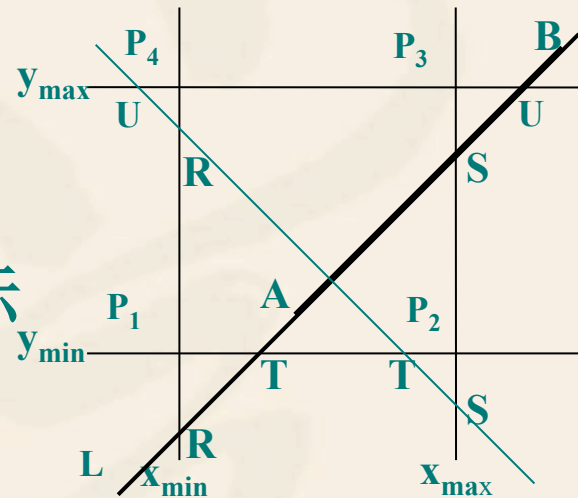


$$Q = RS \cap TU$$

1. 直线段裁剪(17/18)

Liang-Barsky裁剪算法

- ❖ RS, AB, TU 三条线段的交集的端点坐标
- ❖ 等价于求三条线段的左端点的最大值，右端点的最小值



AS是一维窗口TS中的可见部分

- ❖ 向x轴投影，得到可见线段端点的 x 坐标变化范围

$$\max[x_{\min}, \min(x_A, x_B), \min(x_T, x_U)] \leq x \leq \min[x_{\max}, \max(x_A, x_B), \max(x_T, x_U)]$$

左端点x坐标 $x_\alpha = \max[x_{\min}, \min(x_A, x_B), \min(x_T, x_U)]$

右端点x坐标 $x_\beta = \min[x_{\max}, \max(x_A, x_B), \max(x_T, x_U)]$

y坐标可由将x坐标代入直线方程计算得到

1. 直线段裁剪(18/18)

Liang-Barsky裁剪算法

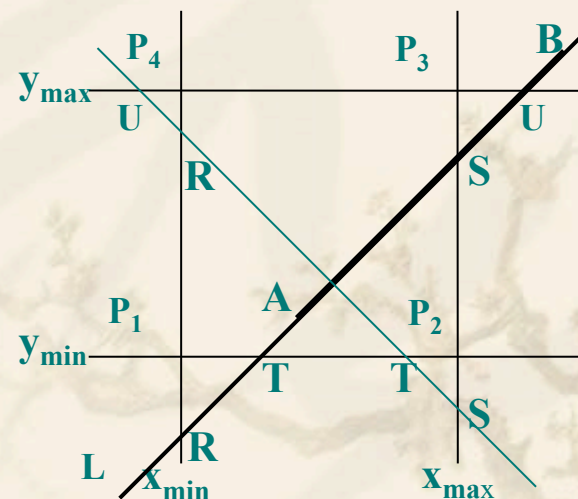
❖ **AB**有可见部分的充要条件也可表示为

$$\max[x_{\min}, \min(x_A, x_B), \min(x_T, x_U)] \leq \min[x_{\max}, \max(x_A, x_B), \max(x_T, x_U)]$$

$$\text{令 } L = \max[x_{\min}, \min(x_A, x_B)] \quad R = \min[x_{\max}, \max(x_A, x_B)]$$

$$\text{则有: } \max[L, \min(x_T, x_U)] \leq \min[R, \max(x_T, x_U)]$$

$$\text{等价于判断} \begin{cases} L \leq R \\ L \leq \max(x_T, x_U) \\ \min(x_T, x_U) \leq R \end{cases}$$



5.5 二维线段裁剪

1 直线段裁剪

- ∞ 直接求交算法
- ∞ Cohen-Sutherland算法
- ∞ 中点分割裁剪算法
- ∞ 梁友栋-Barsky算法

2 多边形裁剪

- ∞ Sutherland_Hodgman算法
- ∞ Weiler-Atherton算法

2. 多边形裁剪 (Polygon clipping) -1/3

- ❖ 错觉：多边形裁剪是直线段裁剪的组合？
- ❖ 新的问题：

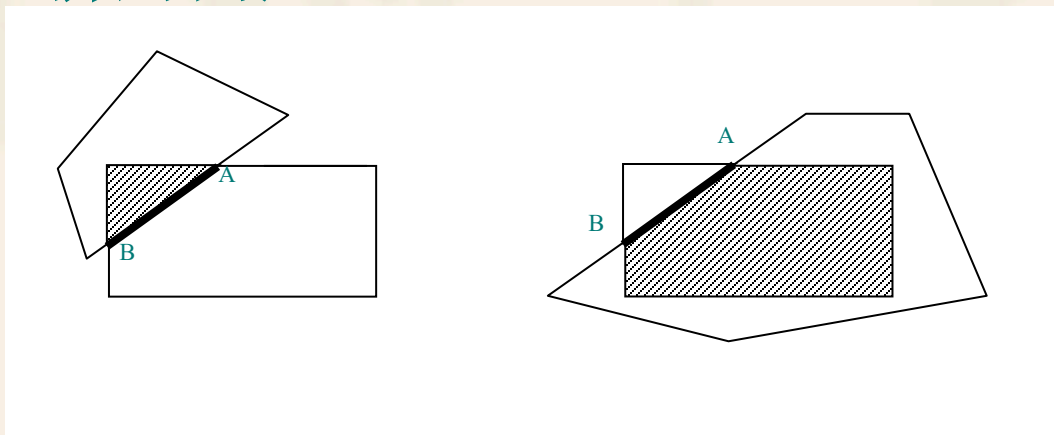


图1 因丢失顶点信息而去法确定裁剪区域

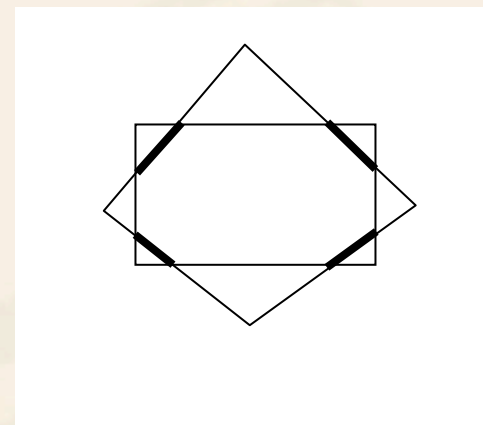
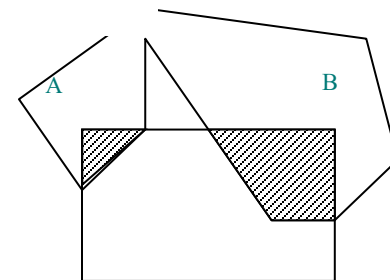
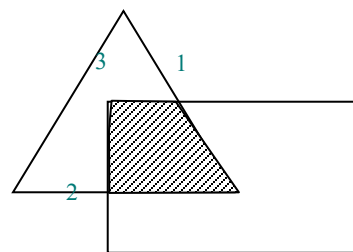
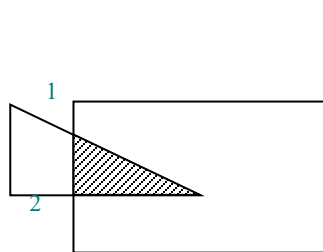
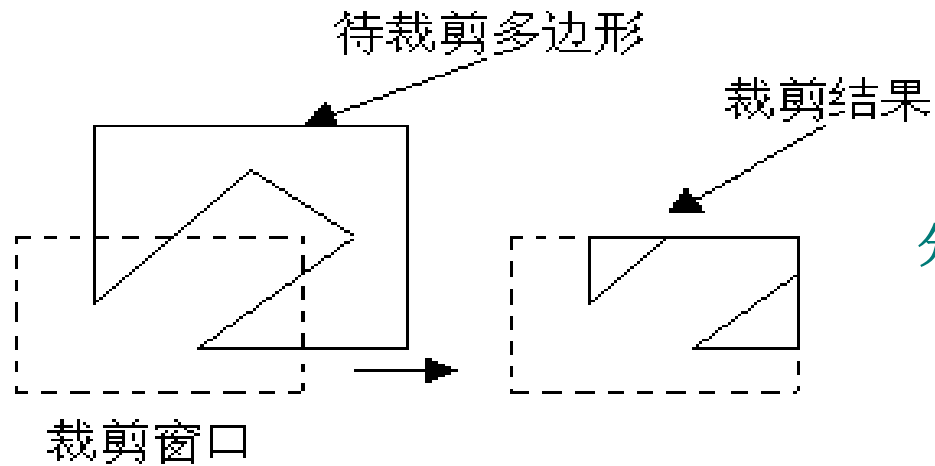


图2 原来封闭的多边形变成了孤立的线段

边界不再封闭，需要用窗口边界的恰当部分来封闭它

2. 多边形裁剪-2/3



裁剪后的多边形顶点形成的几种情况

2. 多边形裁剪-3/3

❖ 关键:

- ⌘ 不仅在于求出新的顶点，删去界外顶点
- ⌘ 还在于形成正确的顶点序列

❖ 常用算法

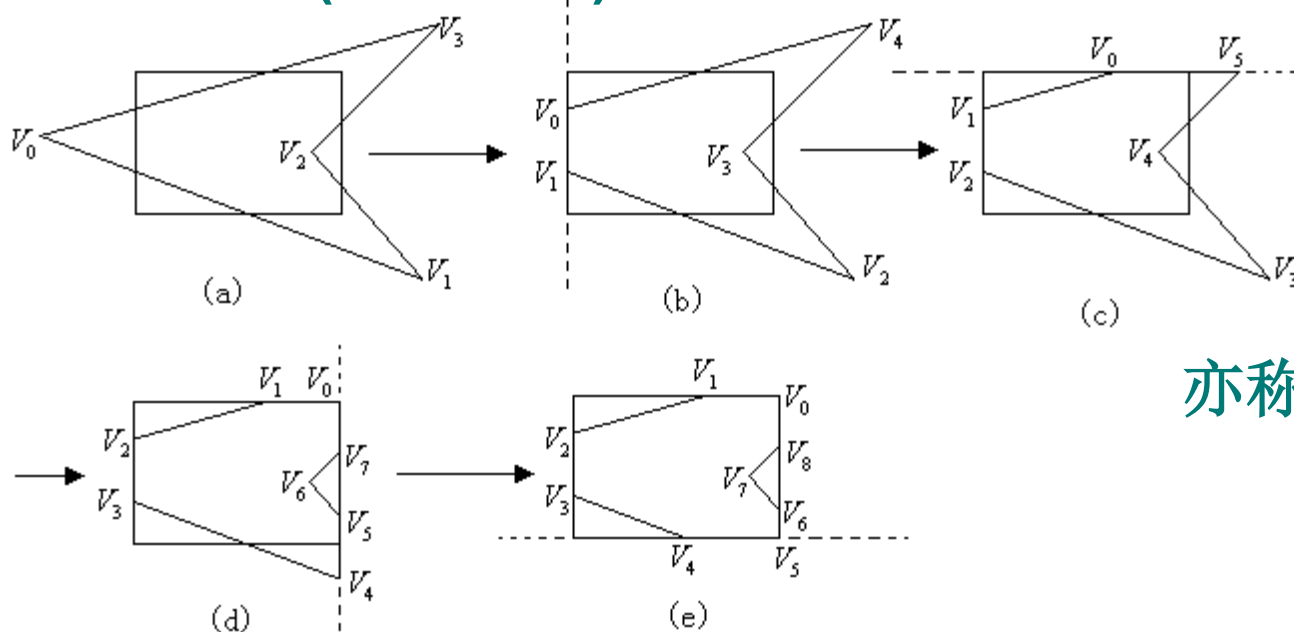
- ⌘ Sutherland_Hodgman算法
- ⌘ Weiler-Atherton算法

Sutherland-Hodgman算法-1/4

❖ 分割处理策略:

∞ 将多边形关于矩形窗口的裁剪分解为多边形关于窗口四边所在直线的裁剪。

❖ 流水线过程(左上右下): 左边的结果是右边的开始。

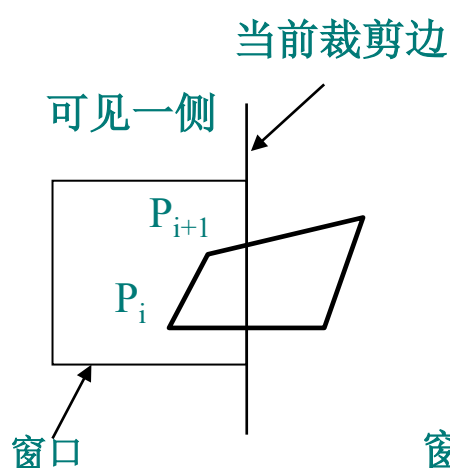
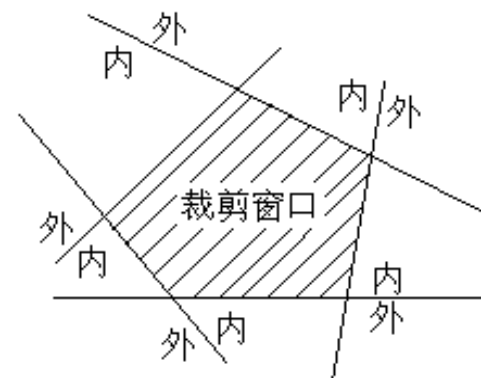


亦称逐边裁剪算法

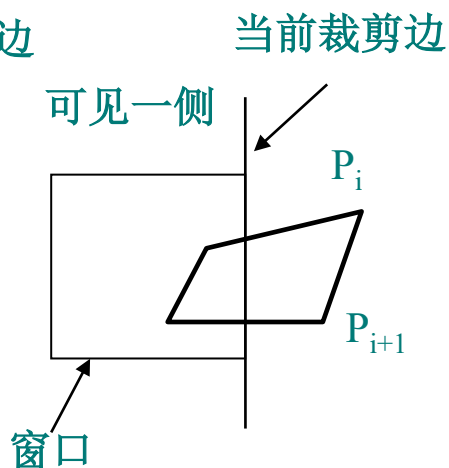
Sutherland-Hodgman算法-2/4

☞ 内侧空间与外侧空间

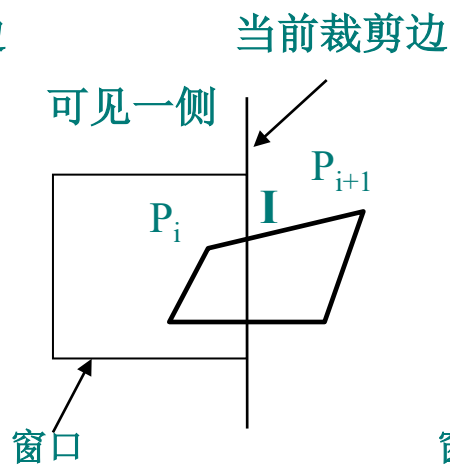
☞ 多边形的边与半空间的关系



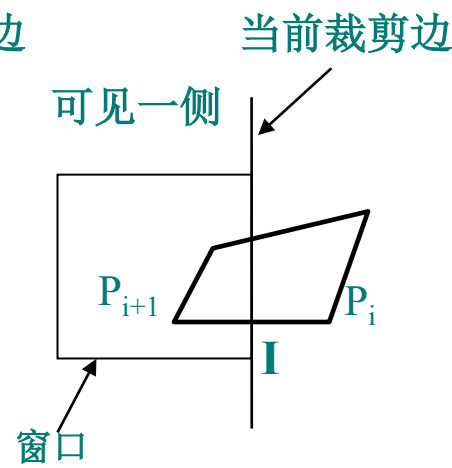
(a) 输出 P_{i+1}



(a) 无输出



(a) 输出I



(a) 输出I和 P_{i+1}

线段与当前裁剪边的位置关系

Sutherland-Hodgman算法-3/4

- ❖ 裁剪结果的顶点构成：
 - ∞ 裁剪边内侧的原顶点；
 - ∞ 多边形的边与裁剪边的交点。
- ❖ 顺序连接。

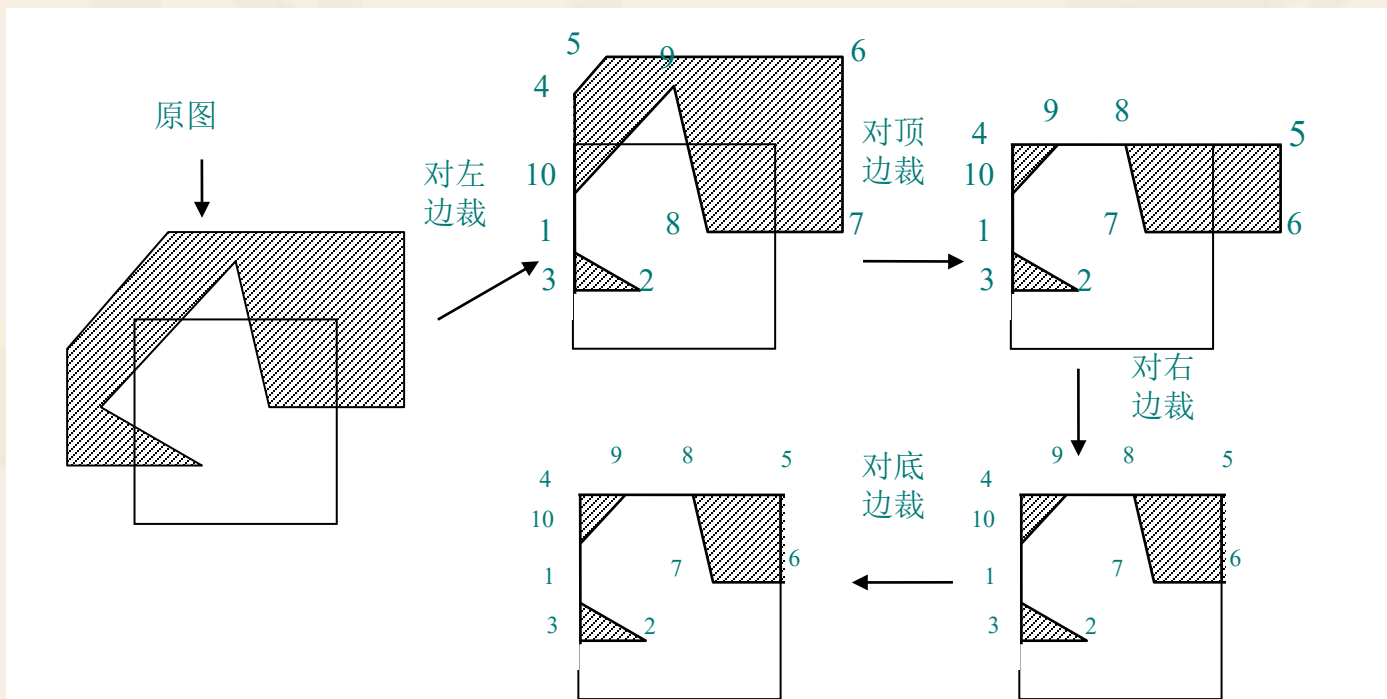
几点说明：

- 裁剪算法采用流水线方式，适合硬件实现。
- 可推广到任意凸多边形裁剪窗口

Sutherland-Hodgman算法-4/4

存在的问题

- ❖ 逐边裁剪要求裁剪窗口为凸多边形，那么凹多边形窗口怎么办？
- ❖ 逐边裁剪法对凹多边形裁剪时，裁剪后分裂为几个多边形，这几个多边形沿边框产生多余的线段？

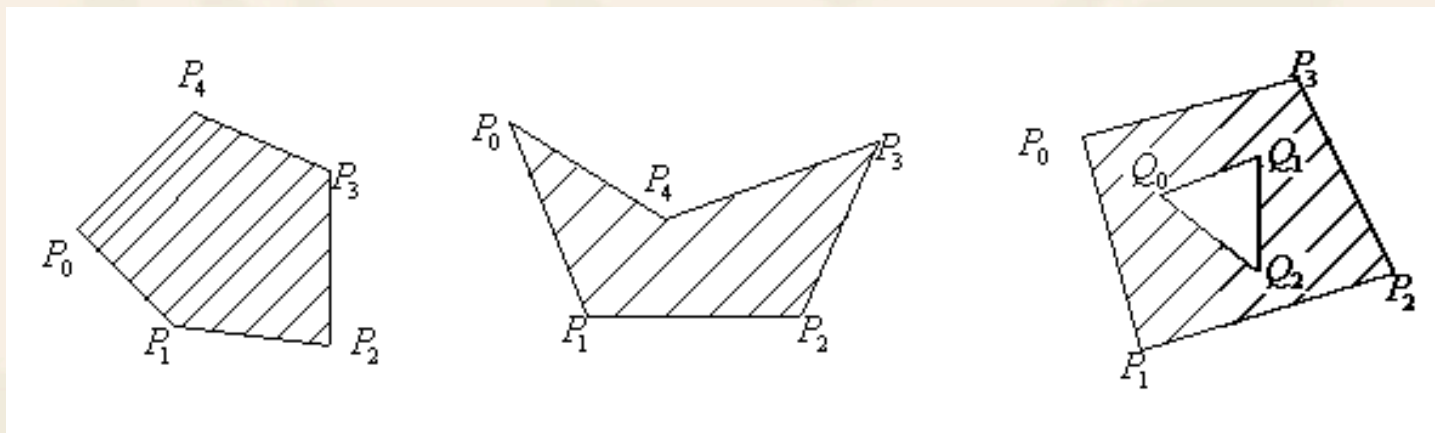


Demo

图6 逐边裁剪法对凹多边形裁剪时可能出现的问题

Weiler-Atherton算法-1/7

裁剪窗口为任意多边形（凸、凹、带内环）的情况：



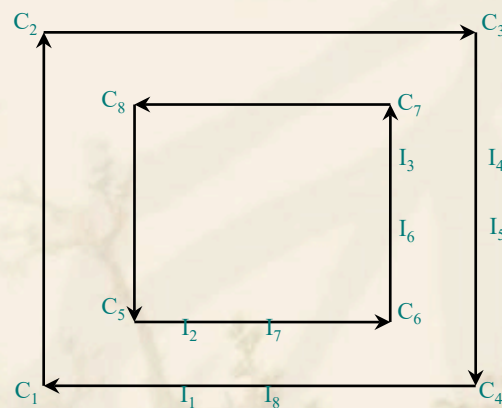
∞主多边形：被裁剪多边形，记为**SP**

∞裁剪多边形：裁剪窗口，记为**CP**

Weiler-Atherton算法-2/7

❖ 约定:

- ❧ **SP**与**CP**均用它们顶点的环形链表定义
- ❧ 外边界取顺时针方向
- ❧ 内边界取逆时针方向
- ❧ 使得沿多边形的边走动，其右边为多边形的内部。



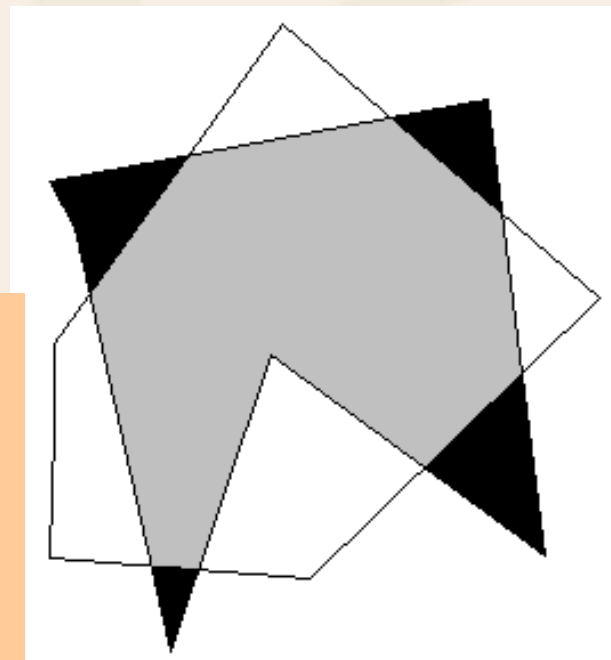
Weiler-Atherton算法-3/7

- ❖ **SP**和**CP**把二维平面分成两部分。
- ❖ 内裁剪: $SP \cap CP$
- ❖ 外裁剪: $SP - CP$

裁剪结果区域的边界由两部分构成:

1. **SP**的部分边界
2. **CP**的部分边界

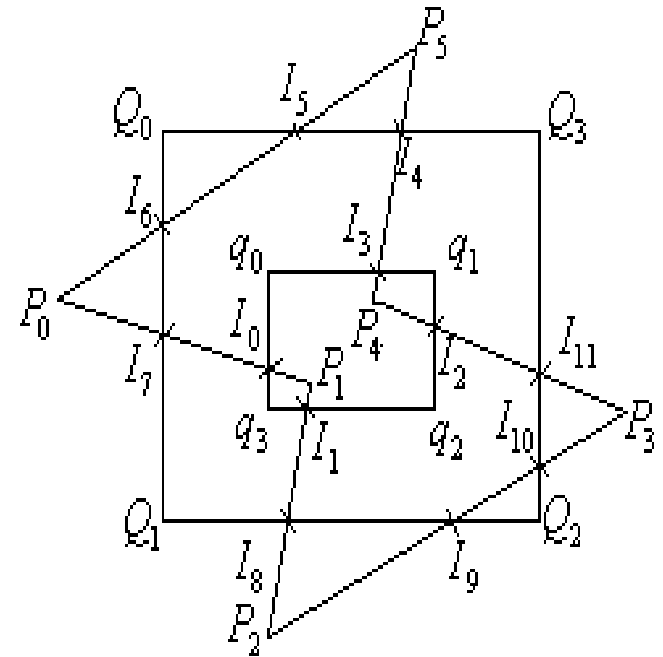
且在交点处, 边界发生交替
即由**SP**边界转至**CP**边界
或由**CP**边界转至**SP**边界



Weiler-Atherton算法-4/7

如果SP与CP有交点，则交点成对出现，
它们被分为如下两类：

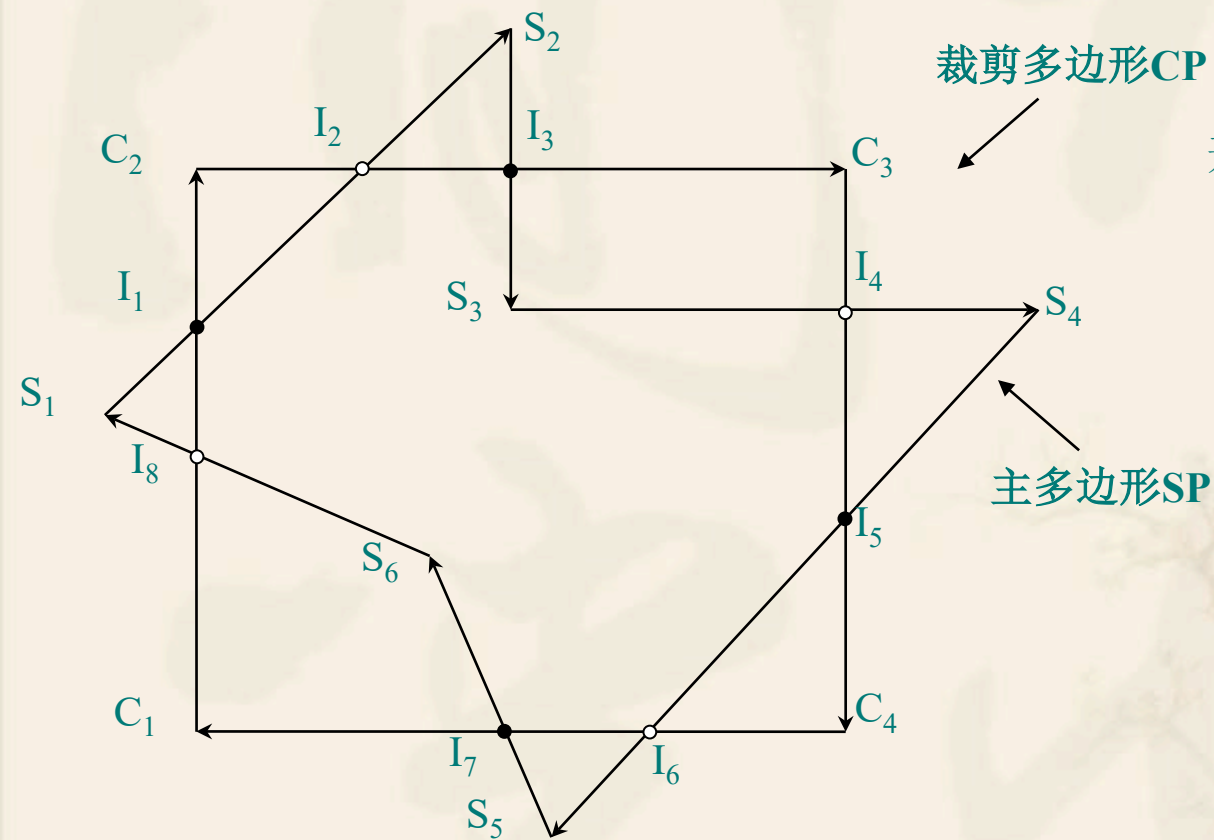
- ❖ 进点：SP边界由此进入CP
如， $I_1, I_3, I_5, I_7, I_9, I_{11}$
- ❖ 出点：SP边界由此离开CP
如， $I_0, I_2, I_4, I_6, I_8, I_{10}$



Weiler-Atherton算法-5/7

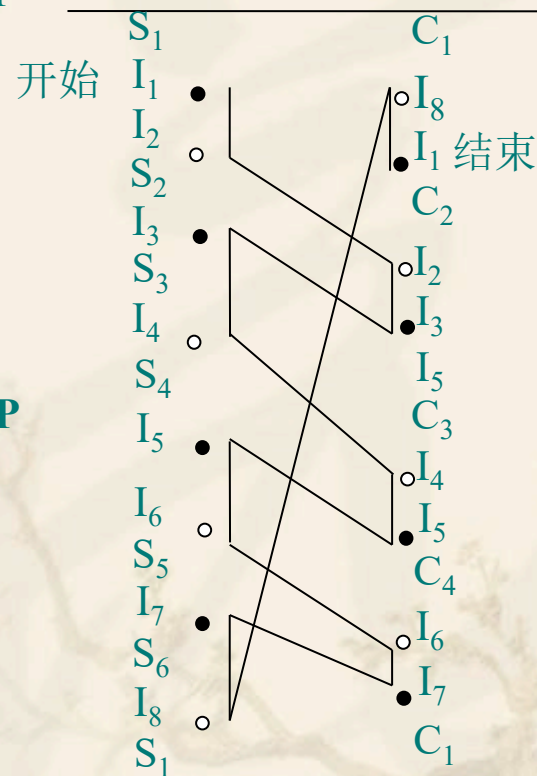
- ❖ 由任一个进点出发，沿**SP**的边，跟踪检测其与**CP**的交点（前交点），并判断该交点是进点还是出点。
- ❖ 若是进点：则沿SP边所示方向收集顶点序列。
- ❖ 若是出点：则从此点开始，检测CP的边所示方向收集顶点序列。
- ❖ 如此交替沿两个多边形的边行进。直至回到跟踪的起始点为止。

Weiler-Atherton算法-6/7

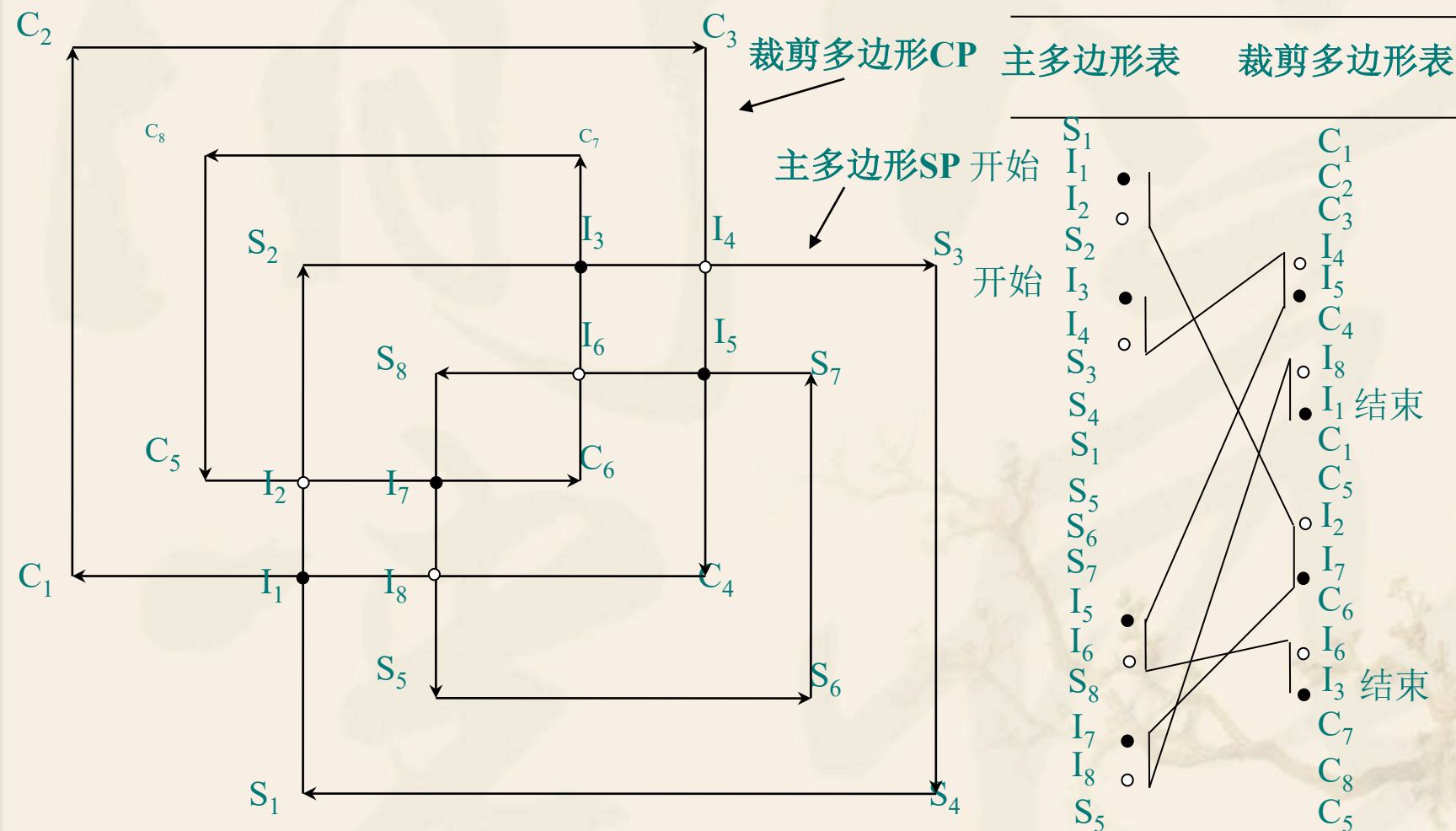


算法裁剪后所生成的多边形为 $I_1 I_2 I_3 S_3 I_4 I_5 I_6 I_7 S_6 I_8 I_1$

主多边形表 裁剪多边形表



Weiler-Atherton算法-7/7



算法裁剪后所生成的多边形为 $I_1I_2I_7I_8I_1$ 和 $I_3I_4I_5I_6I_3$