

第五章 二维图形的裁剪

本章学习内容

□ 直线段裁剪

- 直接求交算法;
- Cohen-Sutherland算法;
- 中点算法
- Nicholl-Lee-Nicholl算法

□ 多边形裁剪

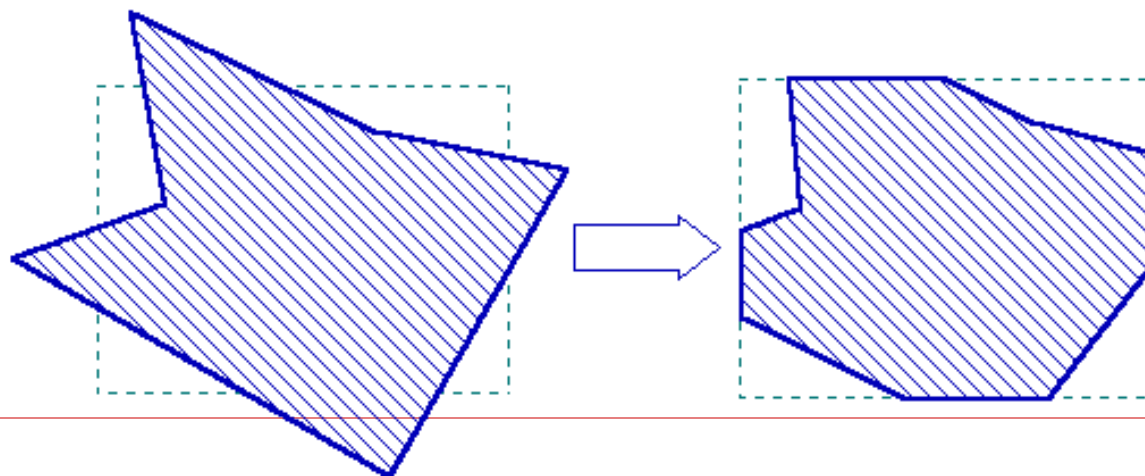
- Sutherland_Hodgman算法
- Weiler-Athenton算法

□ 字符裁剪

掌握要点

- ❑ 掌握什么是裁剪、裁剪窗口，裁剪算法的基本内容：图形关于窗口区域内外关系的判别、图形与窗口的求交计算；
- ❑ 掌握裁剪直线段的Cohen-Sutherland算法、中点分割算法；
- ❑ 掌握裁剪多边形的Sutherland-Hodgman算法(又称逐边裁剪算法)；
- ❑ 了解如何裁剪一个字符串，如何裁剪一个点阵表示(或矢量表示)的字符。

- 在使用计算机处理图形信息时，计算机内部存储的图形往往比较大，而屏幕显示的只是图的一部分。因此需要确定图形中哪些部分落在显示区之内，哪些落在显示区之外，以便只显示落在显示区内的那部分图形。这个选择过程称为裁剪。



直线段裁剪

□ 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

□ 假定条件

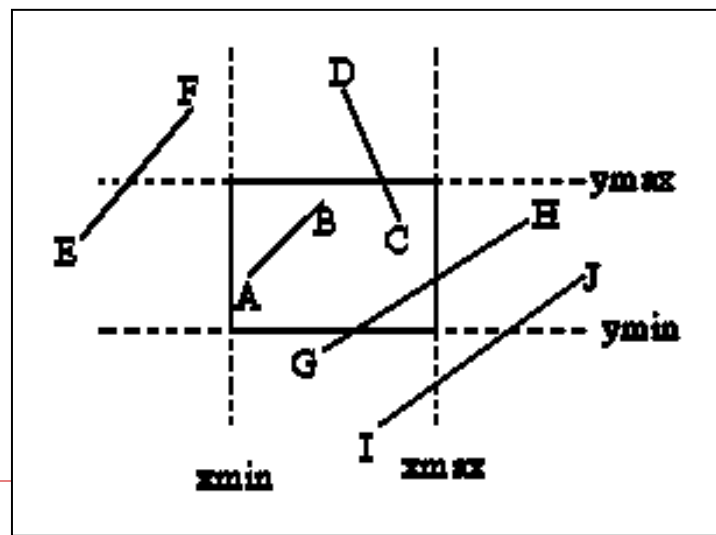
- 矩形裁剪窗口: $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$
- 待裁剪线段: $P_0(x_0, y_0) P_1(x_1, y_1)$

□ 待裁剪线段和窗口的关系

- 线段完全可见(P_0P_1 均在窗口中, 如AB)
- 显然不可见(P_0P_1 均在窗口某条边所在直线的外侧中, 如EF)
- 线段至少有一端点在窗口之外, 但非显然不可见(如CD,GH,IJ), 此时需要进一步求交来确定线段是否有可见部分并求出可见部分。

□ 为提高效率, 算法设计时考虑:

- 快速判断情形(1)(2);
- 设法减少情形(3)求交次数和每次求交时所需的计算量。



点裁剪

□ 点(x, y)在矩形窗口内的充分必要条件是：

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

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

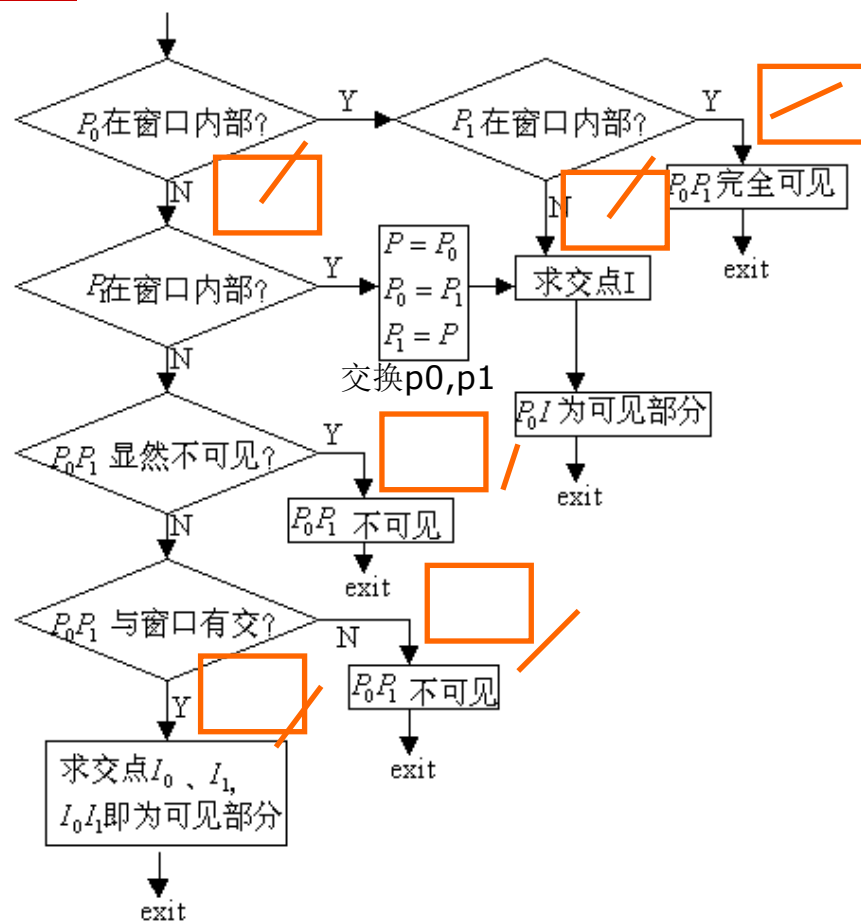
□ 问题：点在任意多边形窗口内或外如何判别？

■ 第四章中点在多边形区域内外的判别方法。

直接求交算法

直线与窗口边都写成参数形式，
求解参数值。

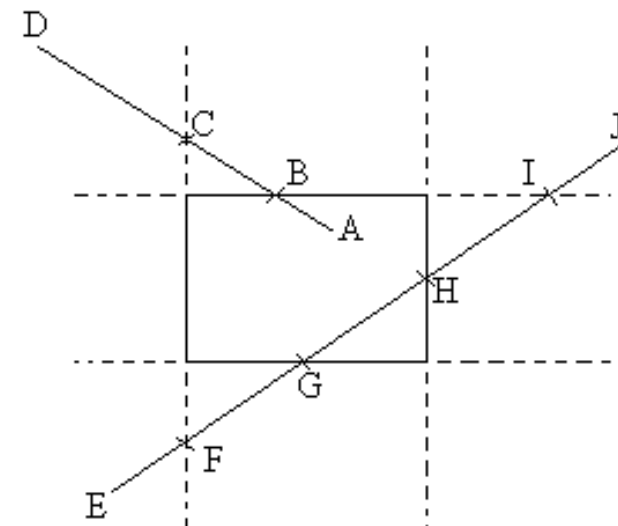
□ 判断属于何种情况计算复杂



Cohen-Sutherland 算法 (编码算法)

□ 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**；否则进入第二步；
2. 判别线段是否为**显然不可见**，如果是，则裁剪结束；否则进行第三步；
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一步，第二步判断，直至结束。



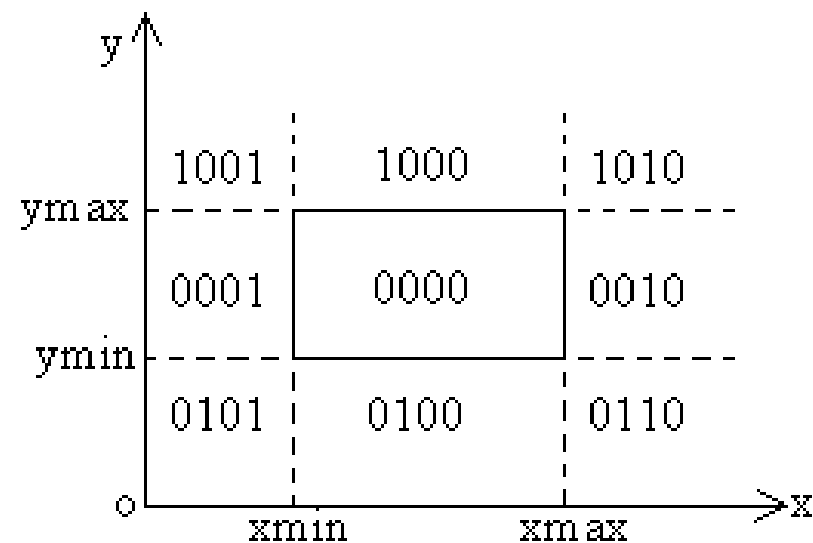
- 特点：对显然不可见线段的快速判别
- 编码方法：由窗口四条边所在直线把二维平面分成9个区域，每个区域赋予一个四位编码，CtCbCrCl，上下右左；

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

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

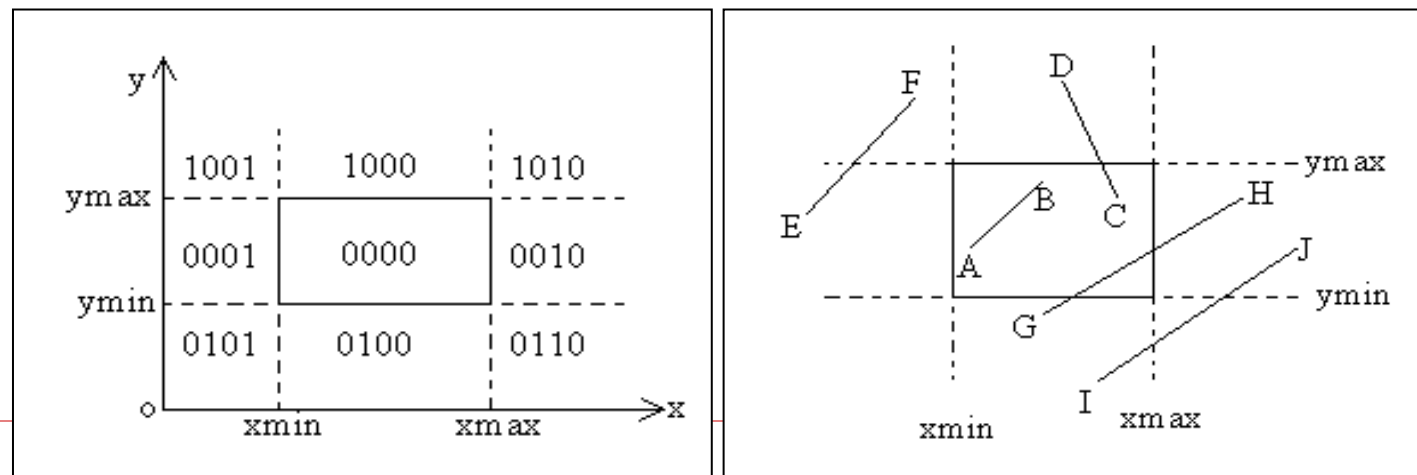
$$C_r = \begin{cases} 1 & \text{当 } x > x_{\max} \\ 0 & \text{else} \end{cases}$$

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



□ 端点编码：定义为它所在区域的编码，

- 完全可见的判别方法：两个端点的编码是否都为0000
- 显然不可见的判别方法：线段的两个端点的编码的逻辑“与”非零，如EF(0001,1001)；而GH(0100,0010),IJ(0100,0010)不是显然不可见。

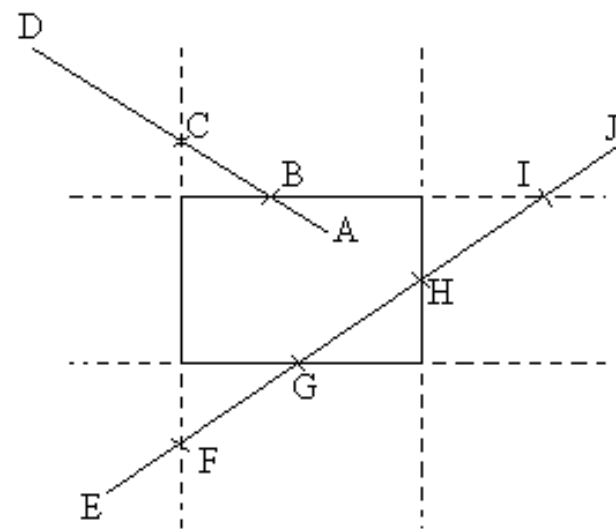


- 对于那些非完全可见、又非显然不可见的线段，需要求交(如线段AD)，求交前需要先测试与窗口哪条边所在直线有交，这只要判断两个端点编码中各位的值。

- 如：AD(0000,1001)，说明AD与上，左线有交

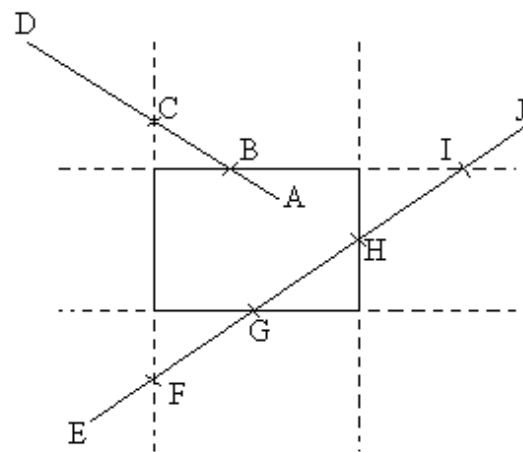
讨论：EJ的情况？

- 程序中固定求交顺序，一般采用左，上，右，下顺序。



Cohen-Sutherland算法(编码算法)

- 特点：用编码方法可快速判断线段--完全可见和显然不可见。
- 特别适用二种场合：
 - 大窗口场合(大多数对象都在窗口中);
 - 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。)
- 最坏情形，线段求交四次，如EJ。



Cohen-Sutherland算法程序(1)

```
#define LEFT 1
#define RIGHT 2
#define BOTTOM 4
#define TOP 8

int encode(float x, float y)          // 计算点x, y的编码
{
    int c=0;
    if(x<XL) c|=LEFT;                // 置位CL
    if(x>XR) c|=RIGHT;               // 置位CR
    if(x<YB) c|=BOTTOM;              // 置位CB
    if(x<YT) c|=TOP;                 // 置位CT
    return c;
}
```

Cohen-Sutherland算法程序(2)

```
void CS_LineClip(x1,y1,x2,y2,XL,XR,YB,YT)
float x1,y1,x2,y2,XL,XR,YB,YT;
    //(x1,y1)(x2,y2)为线段的端点坐标, 其他四个参数定义窗口的边界
{
    int code1,code2,code;
    code1=encode(x1,y1);
    code2=encode(x2,y2);    // 端点坐标编码
    while(code1!=0 || code2!=0) { // 直到“ 完全可见”
        if(code1&code2 !=0) return; // 排除“ 显然不可见” 情况
        code = code1;
        if(code1==0) code=code2; // 求得在窗口外的点
        //按顺序检测到端点的编码不为0, 才把线段与对应的窗口边界求交。
        if(LEFT&code !=0) { // 线段与窗口左边延长线相交
            x=XL;
            y=y1+(y2-y1)*(XL-x1)/(x2-x1);
        }
    }
}
```

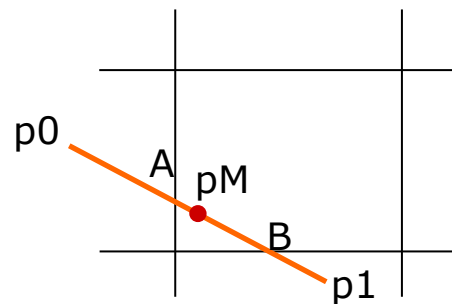
Cohen-Sutherland算法程序(3)

```
    else if(RIGHT & code != 0) { // 线段与窗口右边延长线相交
        x = XR;
        y = y1 + (y2 - y1) * (XR - x1) / (x2 - x1);
    }
    else if(BOTTOM & code != 0) { // 线段与窗口下边延长线相交
        y = YB;
        x = x1 + (x2 - x1) * (YB - y1) / (y2 - y1);
    }
    else if(TOP & code != 0) { // 线段与窗口上边延长线相交
        y = YT;
        x = x1 + (x2 - x1) * (YT - y1) / (y2 - y1);
    }
    if(code == code1) { x1 = x; y1 = y; code1 = encode(x, y); } // 裁去P1到交点
    else { x2 = x; y2 = y; code2 = encode(x, y); } // 裁去P2到交点
}
displayline(x1, y1, x2, y2);
}
```


中点分割法

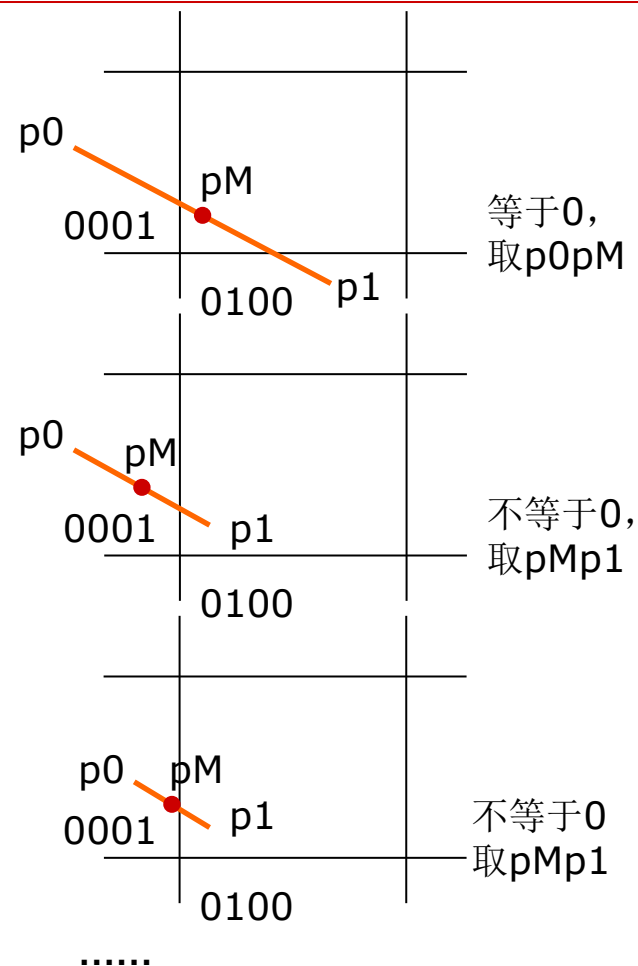
□ 中点分割算法的大意：

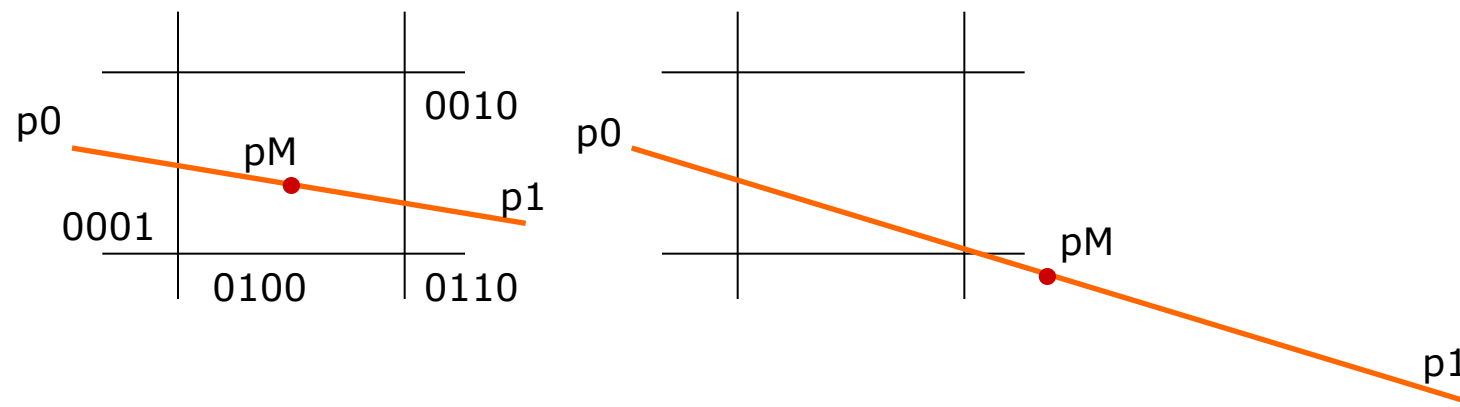
- 与Cohen-Sutherland算法一样，首先对线段端点进行编码，并把线段与窗口的关系分为三种情况：全在、完全不在和线段和窗口有交。
- 对前两种情况，进行一样的处理。对于第三种情况，用中点分割的方法求出线段与窗口的交点。
- 即：从 p_0 点出发找出距 p_0 最近的可见点A
从 p_1 点出发找出距 p_1 最近的可见点B
两个可见点之间的连线即为线段 p_0p_1 的可见部分。



□ 从 p_0 出发找最近可见点采用中点分割方法:

1. 计算出 P_0P_1 的中点 P_m ;
2. 计算 P_0 和 P_m 的区域码的位与, 若结果等于0, 说明最近可见点在 P_0P_m 上, 取 P_0P_m 代替 P_0P_1 ; 否则取 P_mP_1 代替 P_0P_1 ;
3. 如果 P_mP_1 的长度小于给定的容差, 即 $|P_1P_m| < \varepsilon$, 转步4; 否则转步1;
4. 结果输出: P_m 就是 P_0 的最近可见点, 算法结束。

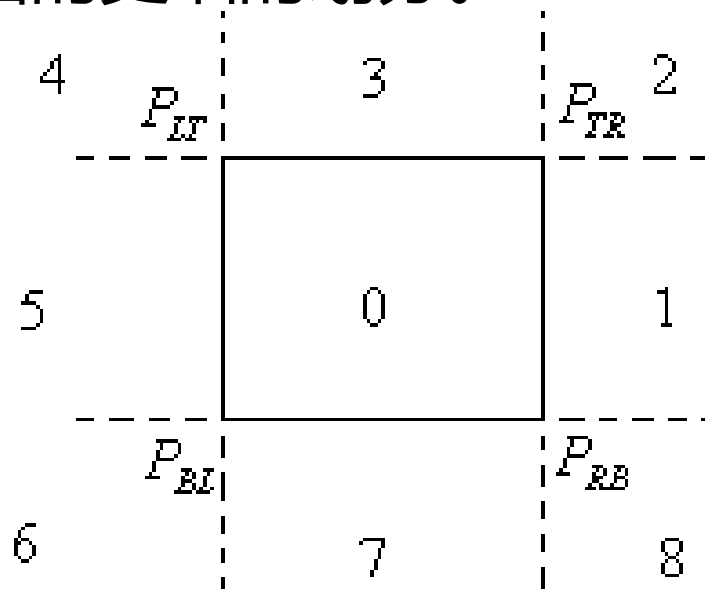




- 该算法的主要计算过程只用到加法和除2运算，所以特别适合硬件实现。

Nicholl-Lee-Nicholl算法

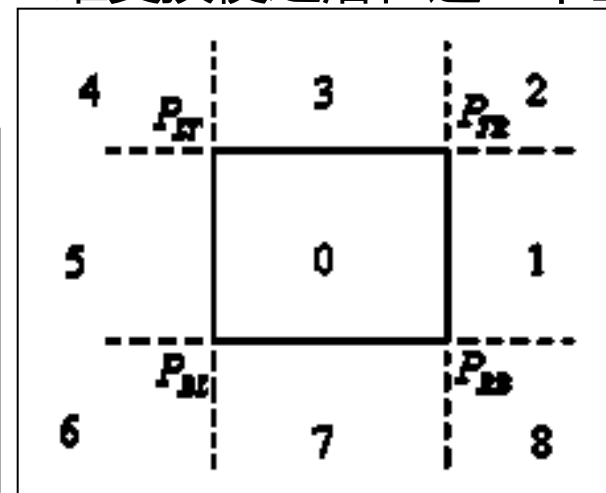
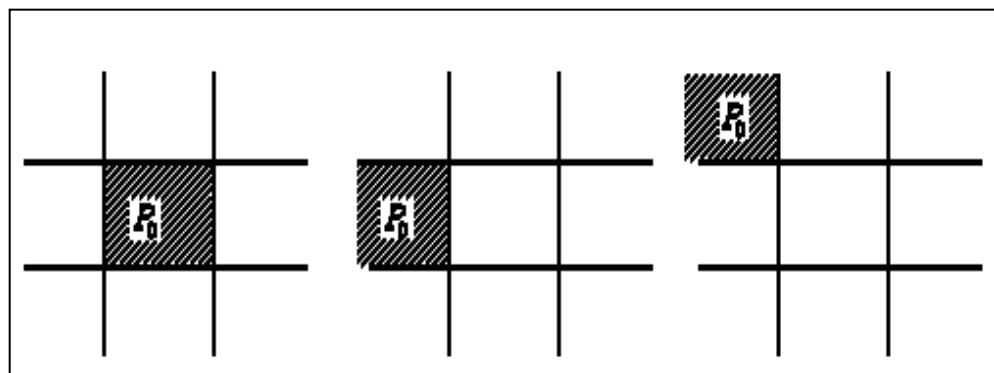
- ❑ 消除C-S算法中多次求交的情况。
- ❑ 基本想法：对2D平面的更细的划分。



□ 假定待裁剪线段 P_0P_1 为非完全可见且非显然不可见。

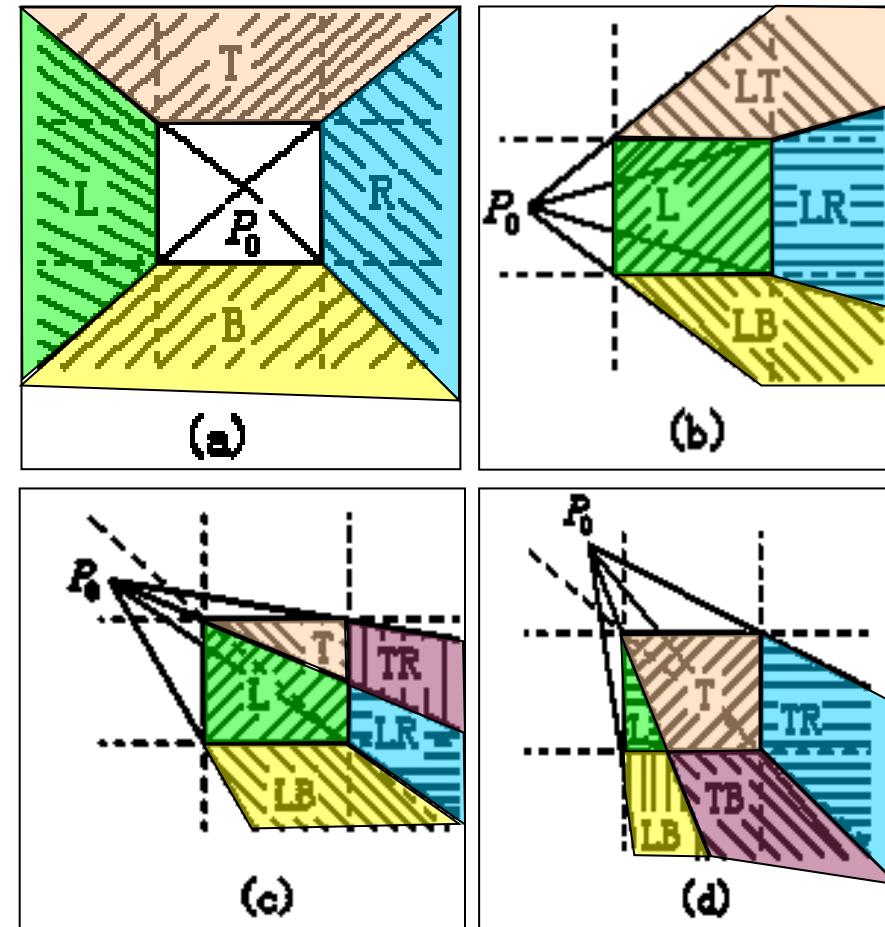
□ 步骤：

1. 窗口四边所在的直线将二维平面划分成9个区域，确定 P_0 所在区域。我们只考虑落在区域0、4、5的情况。其他情况总可以通过简单二维变换使之落在这三个区域里。



2. 从 P_0 点向窗口的四个角点发出射线，这四条射线和窗口的四条边所在的直线一起将二维平面划分为更多的小区。

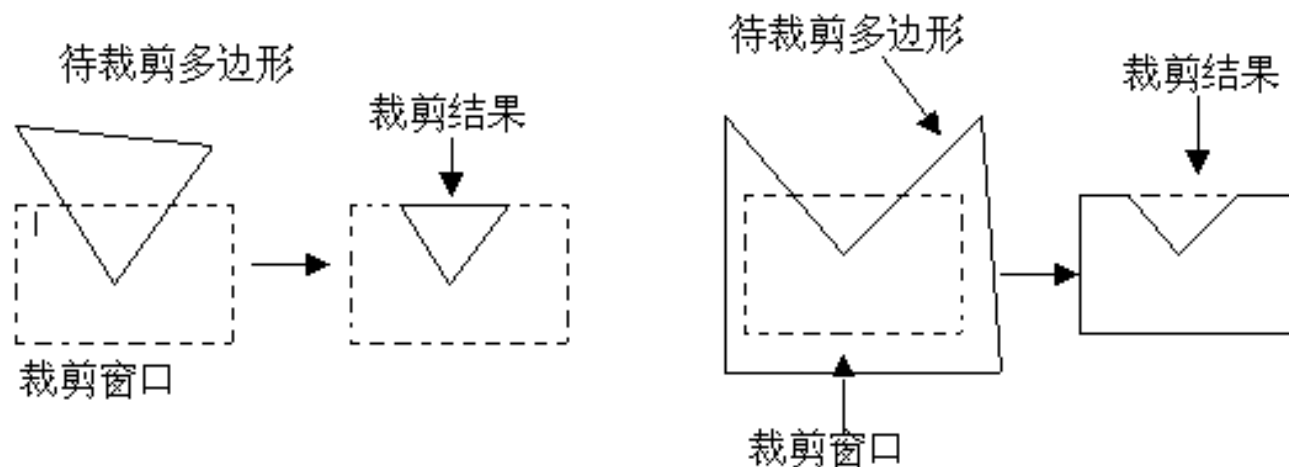
- 此时 P_1 的位置决定了 P_0P_1 和窗口边的相交关系。



-
- 第三步，确定P1所在的区域(判断P1所在区域位置，可判定P0、P1与窗口那条边求交)。根据窗口四边的坐标值及 P_0P_1 和各射线的斜率可确定P1所在的区域。
 - 第四步，求交点，确定 P_0P_1 的可见部分。
 - 评价：效率较高，但仅适合二维矩形窗口。

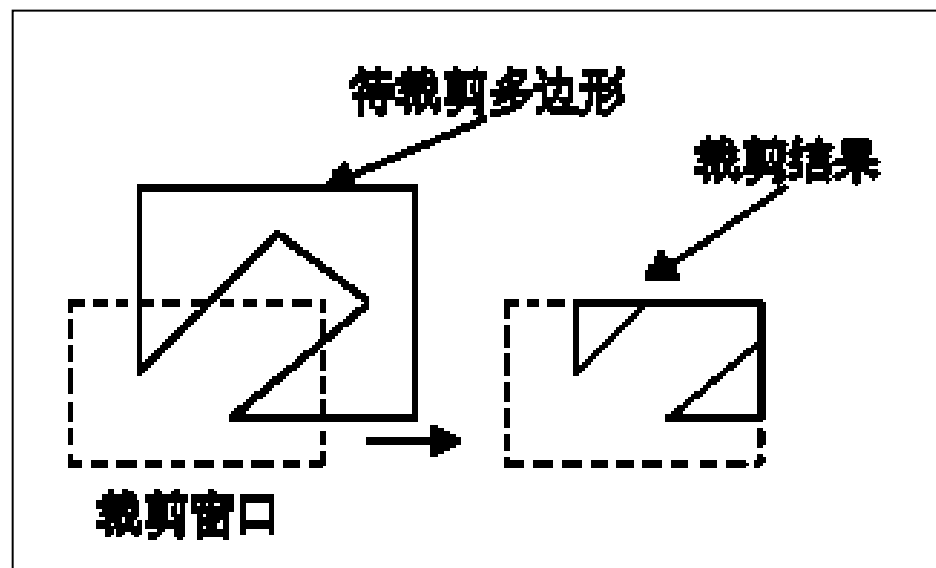
5.2 多边形裁剪

- 由于多边形由直线段组成，是否可以把它分解为边界的线段逐段进行裁剪？
 - 多边形是一个封闭区域，裁剪后仍应当是封闭的多边形。但是多边形的边被裁剪后一般不再封闭了，需要用窗口边的适当部分来封闭它。那么，如何选择这适当部分？



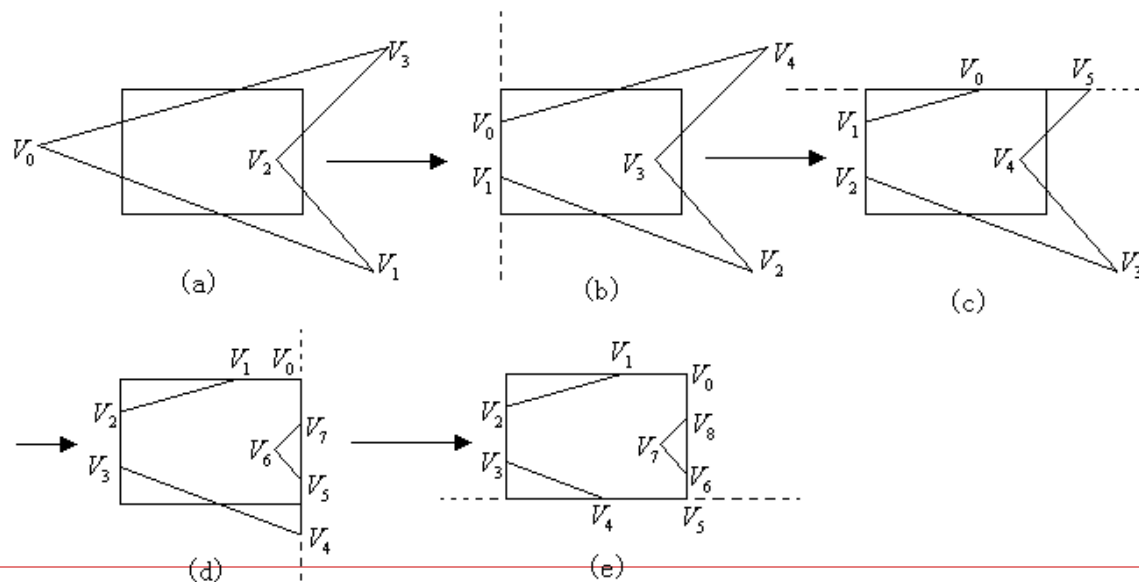
多边形裁剪

- 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？



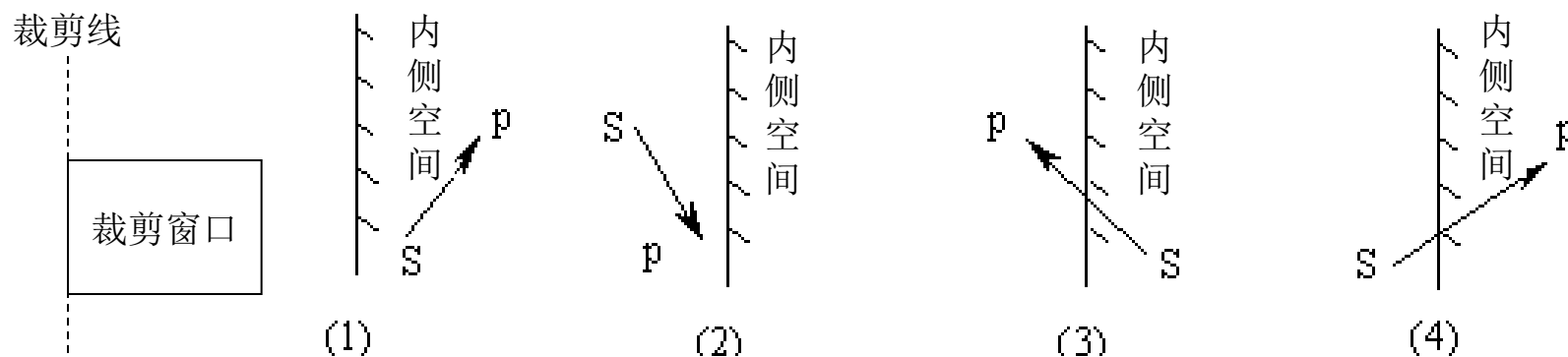
Sutherland-Hodgman算法(逐边裁剪算法)

- 采用分割处理策略：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(顺序：左上右下)：左边裁剪结果是上边裁剪的输入，同理...



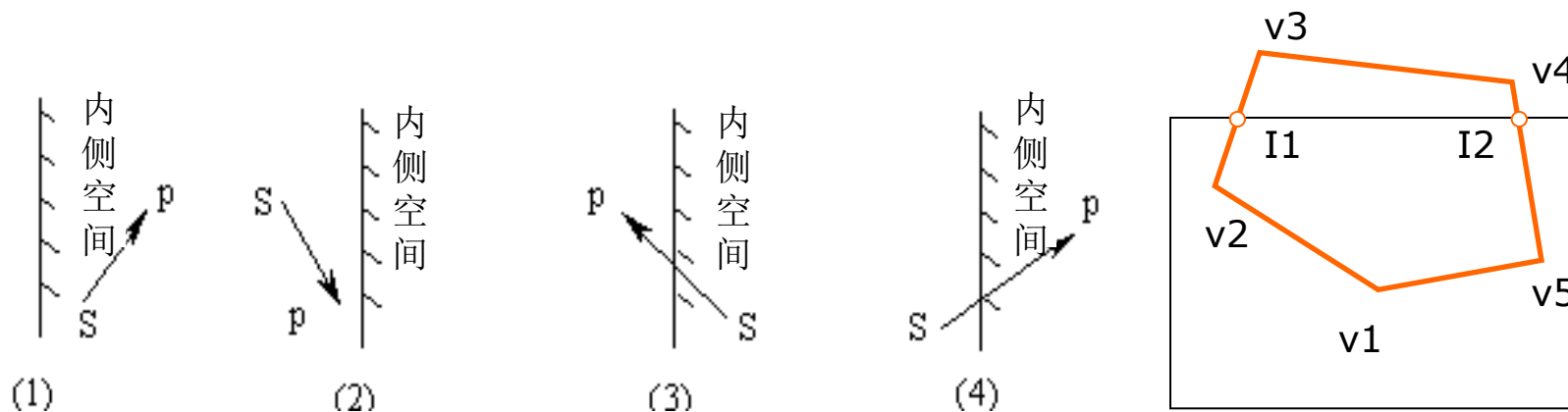
Sutherland-Hodgman算法

- 考虑窗口的边以及延长线构成的裁剪线，该线把平面分成两个部分：一部分包含窗口，称为内侧空间；另一部分称为外侧空间。
- 依序考虑多边形的各条边的两端点S,P：它们与裁剪线的位置关系只有四种。
(1)S,P均在内侧 (2)S,P均在外侧 (3)S在内侧,P在外侧 (4)S在外侧,P在内侧。



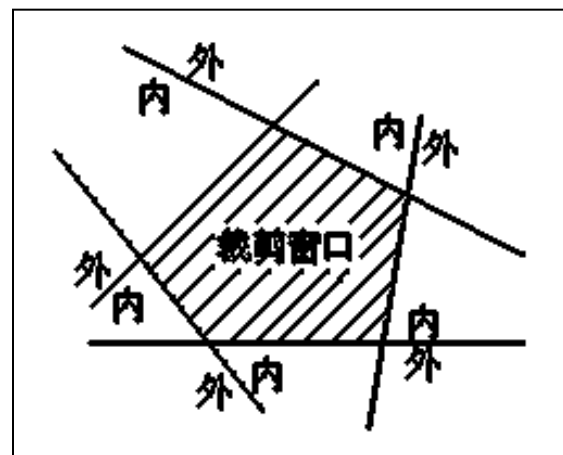
Sutherland-Hodgman算法

- 假设现在处理的多边形边是SP，端点S在上一轮处理过了。线段端点S、P与裁剪线比较之后，可输出0至2个顶点。
- 对于情况(1)仅输出顶点P；情况(2)输出0个顶点；情况(3)输出线段SP与裁剪线的交点I；情况(4)输出线段SP与裁剪线的交点I和终点P



Sutherland-Hodgman算法

- 裁剪结果的顶点集由两部分构成：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

□ 适用范围：

- 适用于裁剪凸多边形。
- 对于凹多边形，如果裁剪后的多边形只有一个，结果仍然是正确的；但是如果裁剪后的多边形有分离部分出现，即结果多边形多于一个，这时会出现多余的线。
- 解决这个问题的办法：将凹多边形分割成2个或多个凸多边形分别进行处理。

Sutherland-Hodgman算法程序

```
typedef struct
    { float x; float y; }Vertex;           //顶点
typedef Edge Vertex[2];                    //边
typedef VertexArray Vertex[MAX];          //多边形

SutherlandHodgmanClip( VertexArray InVertexArray, VertexArray OutVertexArray,
    edge ClipBoundary, int &Inlength, int &Outlength) // 后二者为入，出多边形边数
{ Vertex s, p, ip;
    int j;
    Outlength = 0;                          // 裁剪结果多边形清空
    S = InVertexArray [InLength -1]; // 获得第一个S点
```

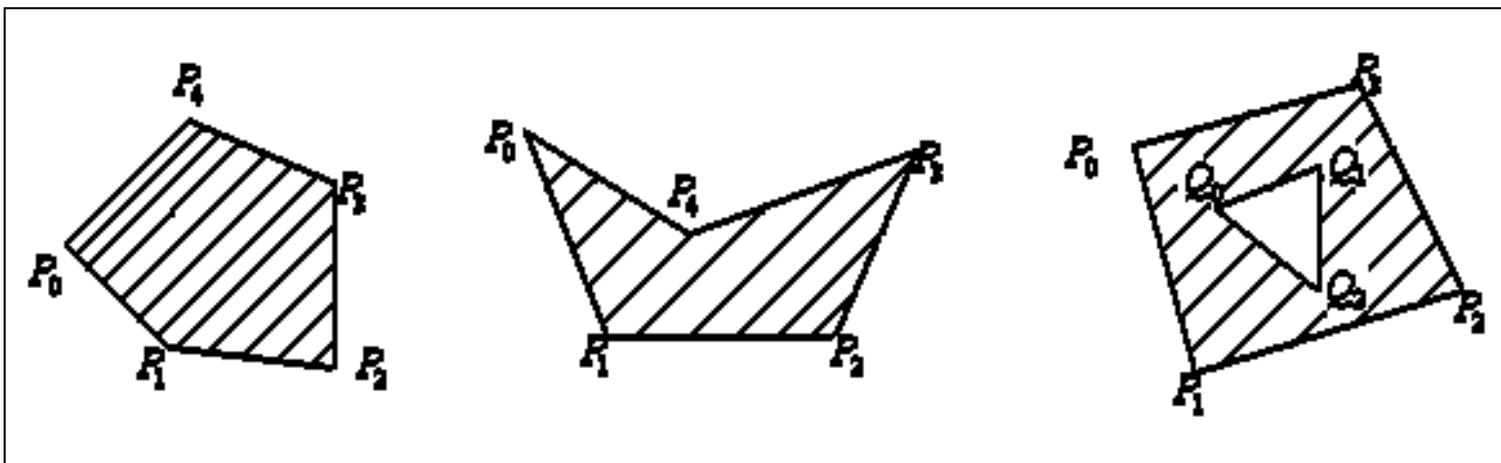
```
For (j = 0; j < Inlength; j++)    // 按序处理被裁多边形各边
{
    P = InVertexArray [j];        // 获得P点
    if(Inside (P, ClipBoundary))
    {
        if(Inside (S, ClipBoundary))    //SP在窗口内, 情况1
            Output(p, OutLength, OutVertex Array);
        else
        {                                //S在窗口外, 情况4
            Intersect (S, P, ClipBoundary, &ip);
            Output (ip, OutLength, OutVertexArray);
            Output (P, OutLength, OutVertexArray);
        }
    }
}
```

```
else if (Inside (S, WindowsBoundary))
{
    //S在窗口内, P在窗口外, 情况3
    Intersect (S, P, ClipBoundary, &ip);
    Output (ip, OutLength, OutVertexArray);
}
//情况2没有输出, 不予编程
// P点成为下一个S点
S = P;
} //endwhile
} //endmain
```

➤ 注意这里只处理了流水线的一个环节, 请同学们自行完成整个流水过程。

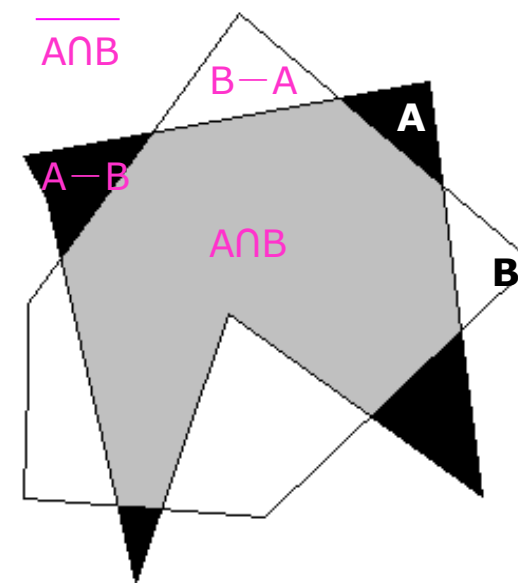
Weiler-Athenton算法

- 上面所述的算法，裁剪窗口都是矩形。
- 有时需考虑裁剪窗口为任意多边形(凸、凹、带内环)的情况：



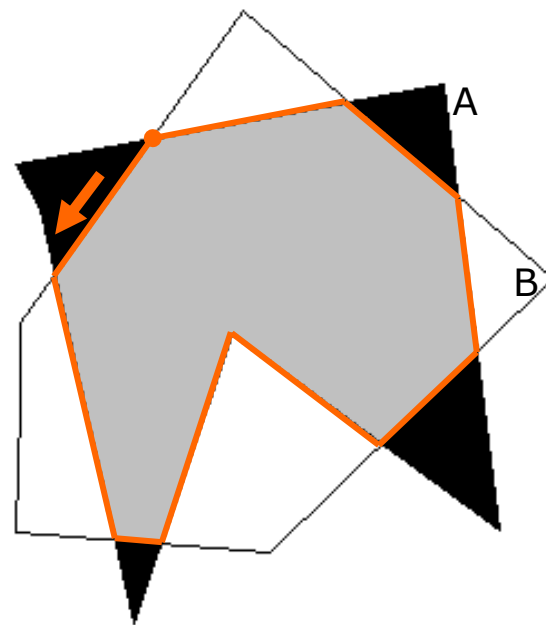
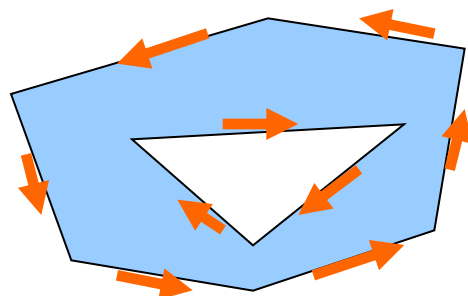
Weiler-Athenton算法

- 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
- 主多边形和裁剪多边形把二维平面分成四部分。
 - 内裁剪(图元在窗口内的部分): $A \cap B$
 - 外裁剪(图元在窗口外的部分): $A - B$



思路

- 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。
- 约定
 - 外环 (由多边形的外部边界构成)顶点编号取逆时针方向
 - 内环 (由多边形的内部边界构成)顶点编号取顺时针方向



□ 由于多边形的封闭性，如果主多边形与裁剪多边形有交点，则交点成对出现，它们被分为如下两类：

■ 进点：主多边形边界由此进入裁剪多边形内

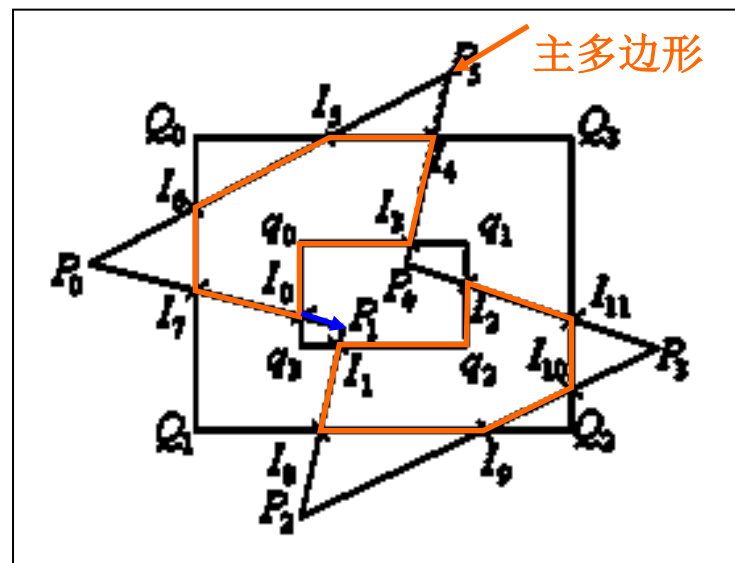
如：l1, l3, l5, l7, l9, l11

■ 出点：主多边形边界由此离开裁剪多边形区域

如：l0, l2, l4, l6, l8, l10

□ 那么：

1. 建顶点表；
2. 求交点，按序插入顶点表；
3. 裁剪...



步骤

1. 建立主多边形和裁剪多边形的顶点表.
2. 求主多边形和裁剪多边形的交点, 并将这些交点按顺序插入两多边形的顶点表中。在两多边形顶点表中的相同交点间建立双向指针。
3. 裁剪:

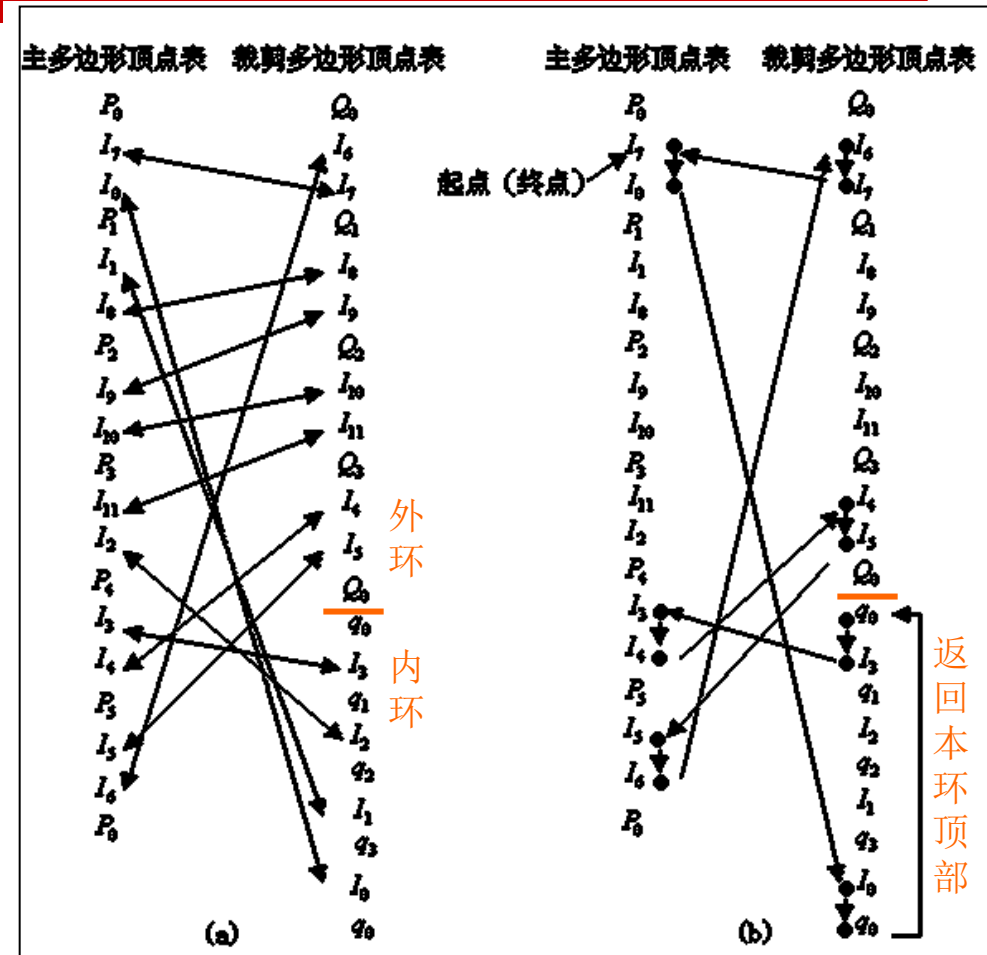
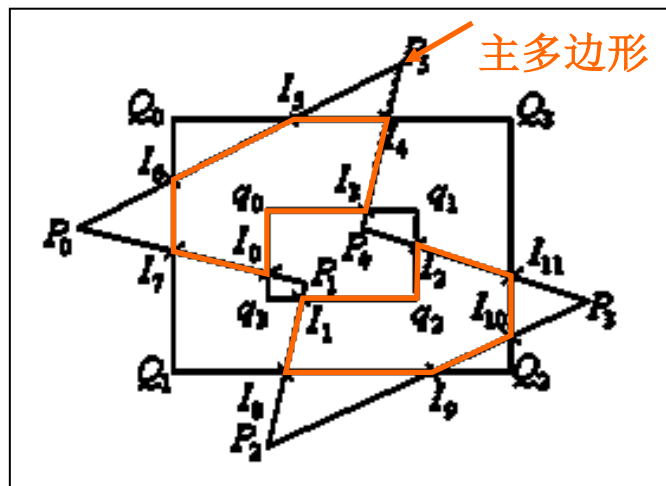
如果存在没有被跟踪过的交点, 执行以下步骤:

- ① 建立空的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, 将其输出到结果多边形顶点表中.

-
- ③ 如果该交点为进点，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，将其输出到结果多边形顶点表中，直至遇到新的交点。
 - ⑤ 将该交点输出到结果多边形顶点表中，并通过连接该交点的双向指针改变跟踪方向(如果上一步跟踪的是主多边形边界，现在改为跟踪裁剪多边形边界；如果上一步跟踪裁剪多边形边界，现在改为跟踪主多边形边界)。
 - ⑥ 重复(4)、(5)直至回到起点

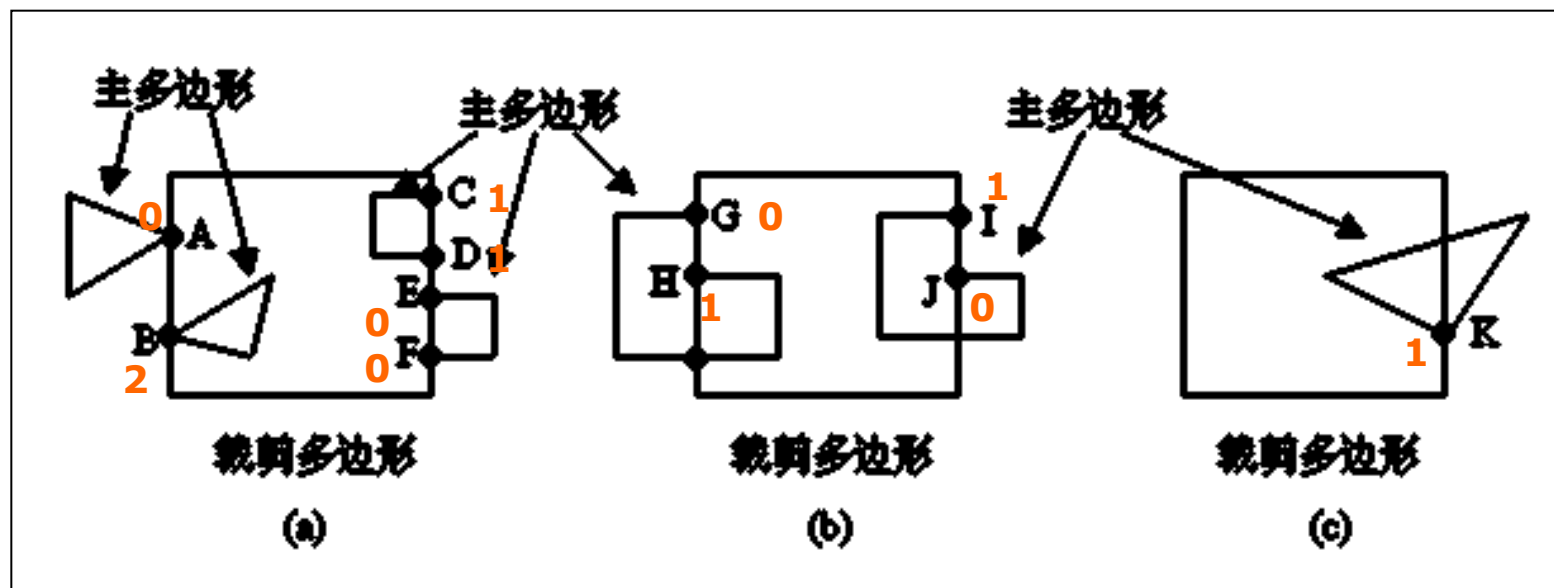
Weiler-Athenton算法

- ① 建立结点表并在相同交点间建立双向指针。
- ② 交替跟踪主多边形和裁剪多边形的边界即可获得结果。
- ③ 如一次跟踪结束还有未跟踪过的结点，任选一点再作跟踪。



交点的奇异情况处理

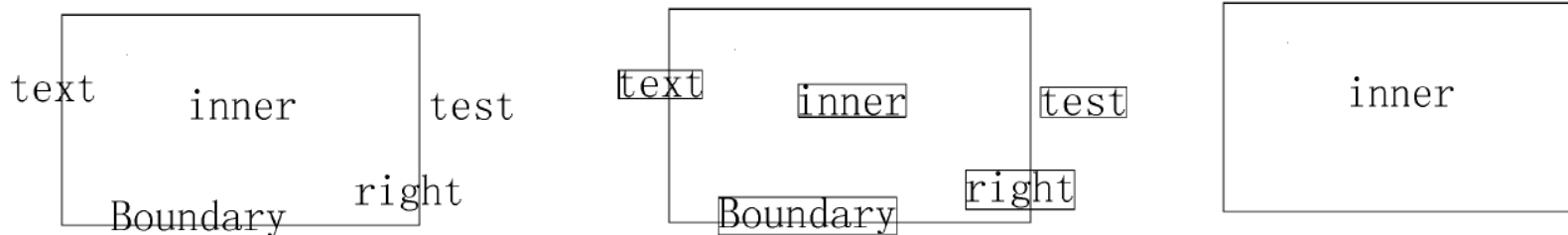
- 与裁剪多边形边重合的主多边形的边不参与求交点；
- 顶点处理：对于顶点落在裁剪多边形的边上的主多边形的边，如果落在该裁剪边的内侧，将该顶点算作交点；而如果这条边落在该裁剪边的外侧，将该顶点不看作交点。



5.3 字符裁剪

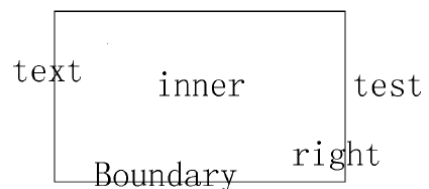
□ 当字符和文本部分在窗口内，部分在窗口外时，就提出了字符裁剪问题。策略选择有：

■ 基于字符串

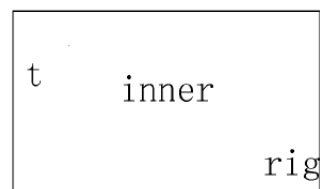


5.3 字符裁剪

■ 基于字符



A rectangular frame containing the text "text" at the top left, "inner" in the center, "test" at the top right, "Boundary" at the bottom left, and "right" at the bottom right.

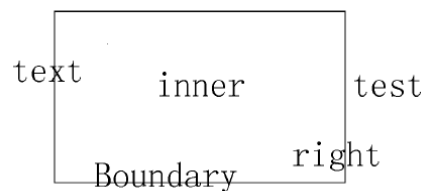


A rectangular frame containing the text "t" at the top left, "inner" in the center, and "rig" at the bottom right. The "test" and "Boundary" labels are absent.

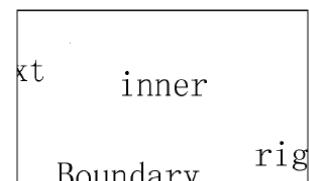
■ 基于构成字符的最小元素

□ 点阵字符：点裁剪

□ 矢量字符：线裁剪



A rectangular frame containing the text "text" at the top left, "inner" in the center, "test" at the top right, "Boundary" at the bottom left, and "right" at the bottom right.



A rectangular frame containing the text "xt" at the top left, "inner" in the center, and "rig" at the bottom right. The "test" and "Boundary" labels are absent.

END
