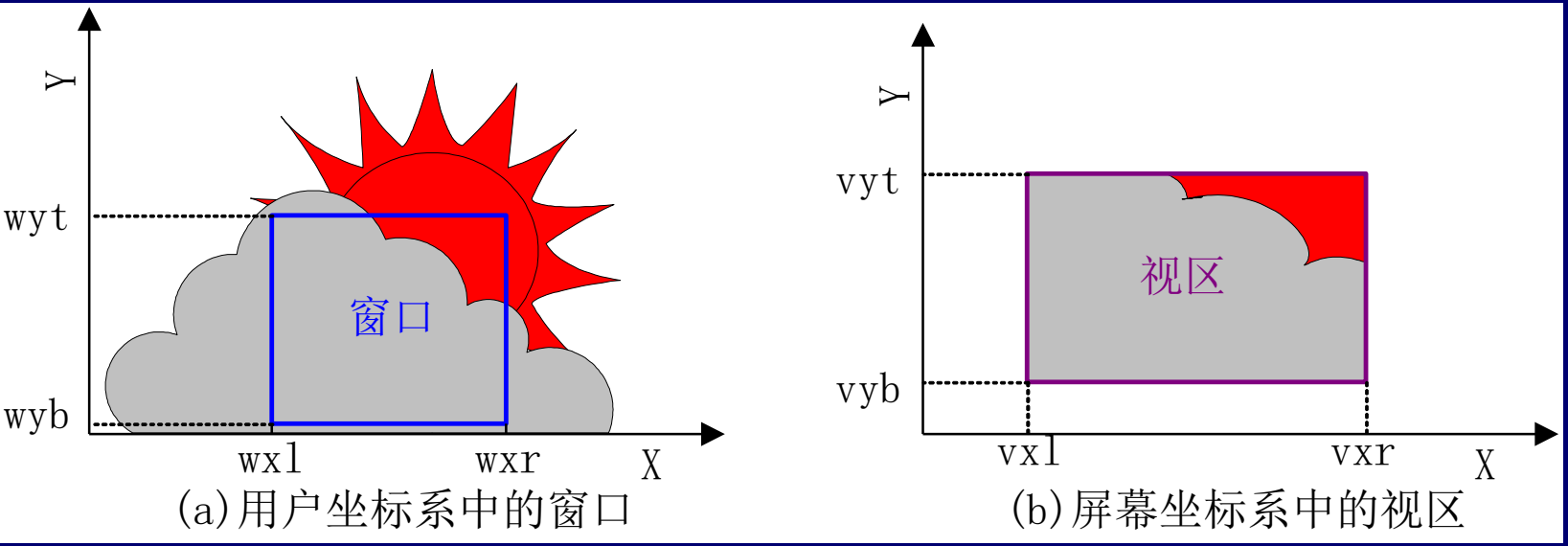
第七章

图形裁剪算法

一、窗口视图变换

基本概念

- 在计算机图形学中，将在用户坐标系中需要进行观察和处理的一个坐标区域称为**窗口**（Window）
- 将窗口映射到显示设备上的坐标区域称为**视区**（Viewport）



要将窗口内的图形在视区中显示出来，必须经过将窗口到视区的变换（Window-Viewport Transformation）处理，这种变换就是**观察变换（Viewing Transformation）**。

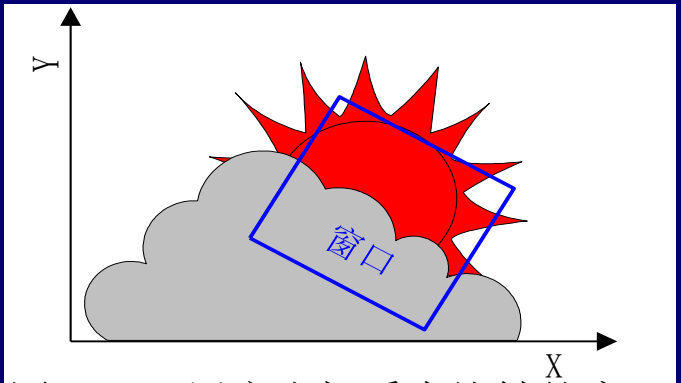
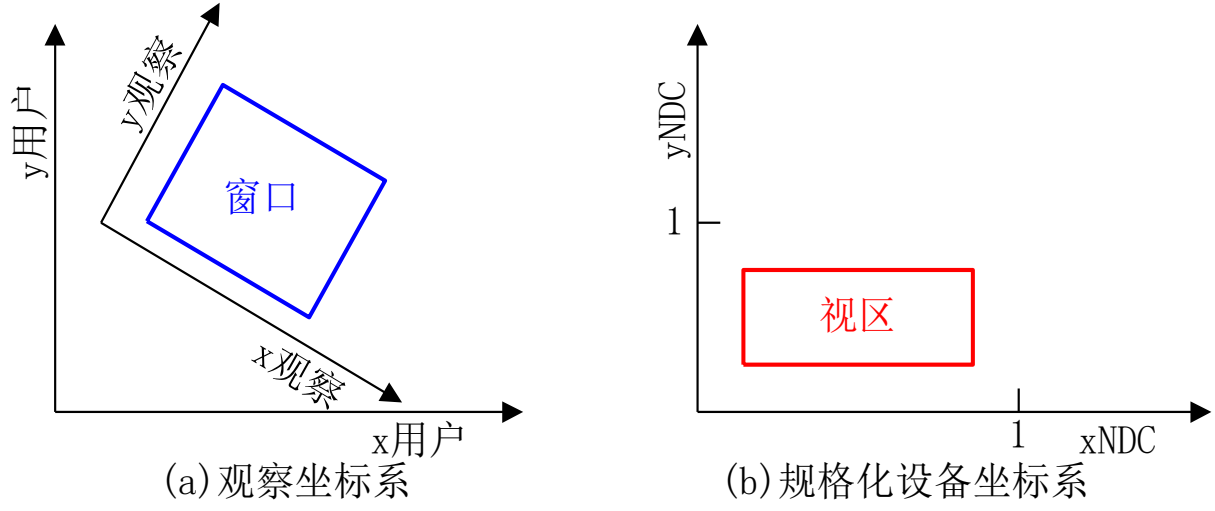


图6-17　用户坐标系中旋转的窗口



(a) 观察坐标系　　(b) 规格化设备坐标系

- **窗口到视区的变换**

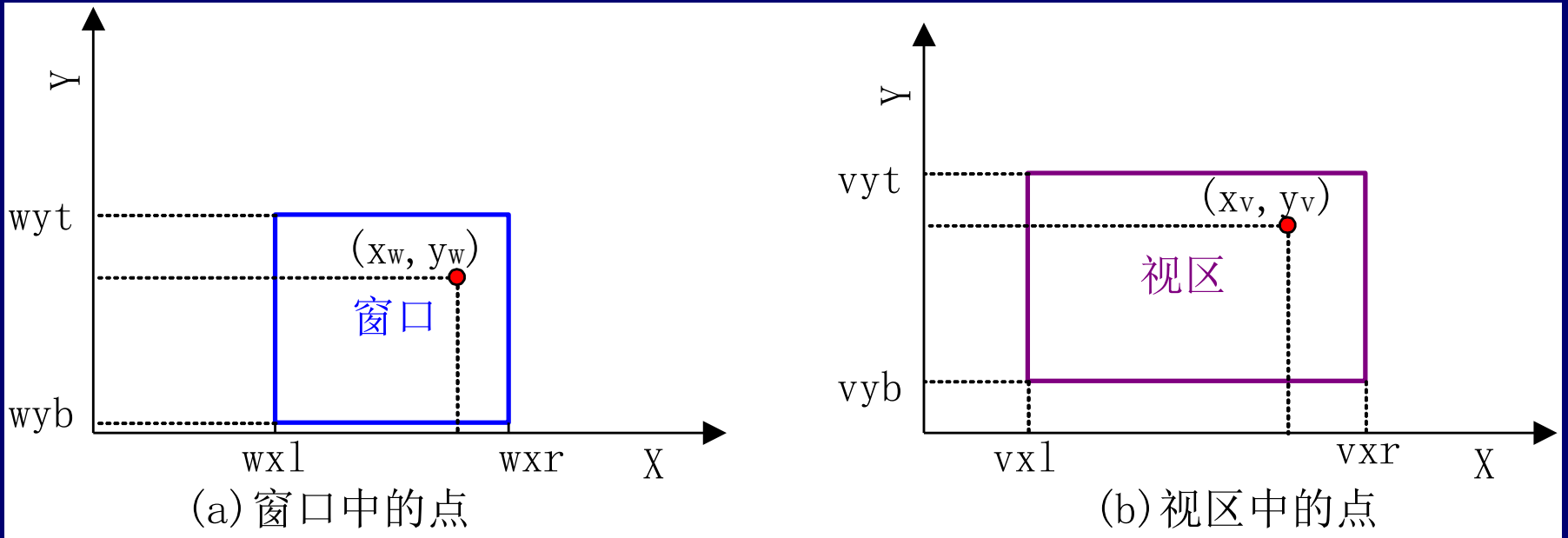


图6-23　窗口到视区的变换

要将窗口内的点（x_w,y_w）映射到相对应的视区内的点（x_v,y_v）需进行以下步骤：

(1) 将窗口左下角点移至用户系统系的坐标原点

(2) 针对原点进行比例变换

(3) 进行反平移

二、二维裁剪

- 在二维观察中，需要在观察坐标系下对窗口进行**裁剪**，即只保留窗口内的那部分图形，去掉窗口外的图形。
- **假设**窗口是标准矩形，即边与坐标轴平行的矩形，由上（y=wyt）、下（y=wyb）、左（x=wxl）、右（x=wxr）四条边描述。
- **5.2.1 点的裁剪**

1 点的裁剪

$$wxl \le x \le wxr,$$

且 $$wyb \le y \le wyt$$

1、直线段的裁剪

假定直线段用$p_1(x_1,y_1)p_2(x_2,y_2)$表示。

- 直线段和剪裁窗口的可能关系：
  - 完全落在窗口内
  - 完全落在窗口外
  - 与窗口边界相交

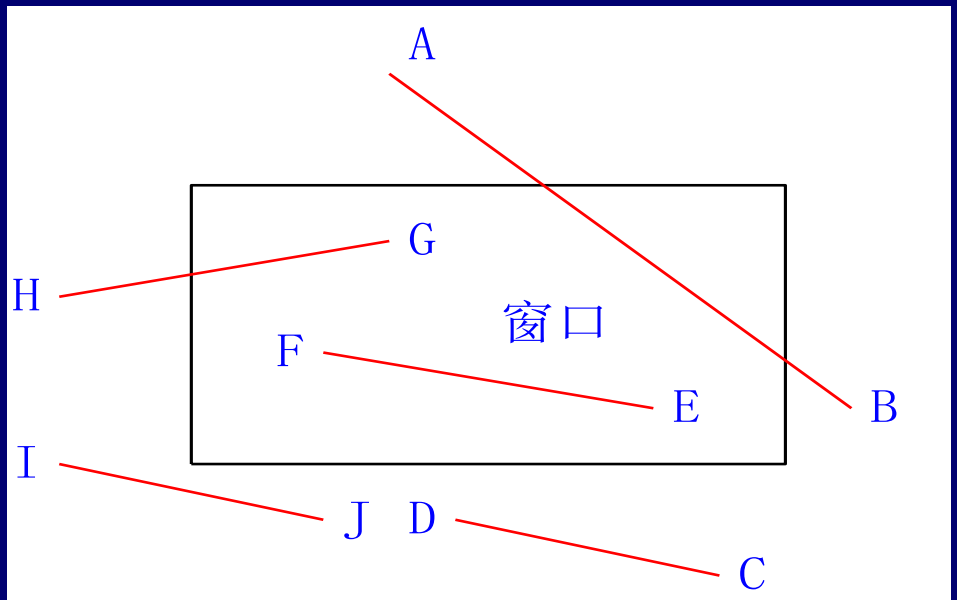


图6-24　直线段与窗口的关系

- **实交点**是直线段与窗口矩形边界的交点。
- **虚交点**则是直线段与窗口矩形边界延长线或直线段的延长线与窗口矩形边界的交点

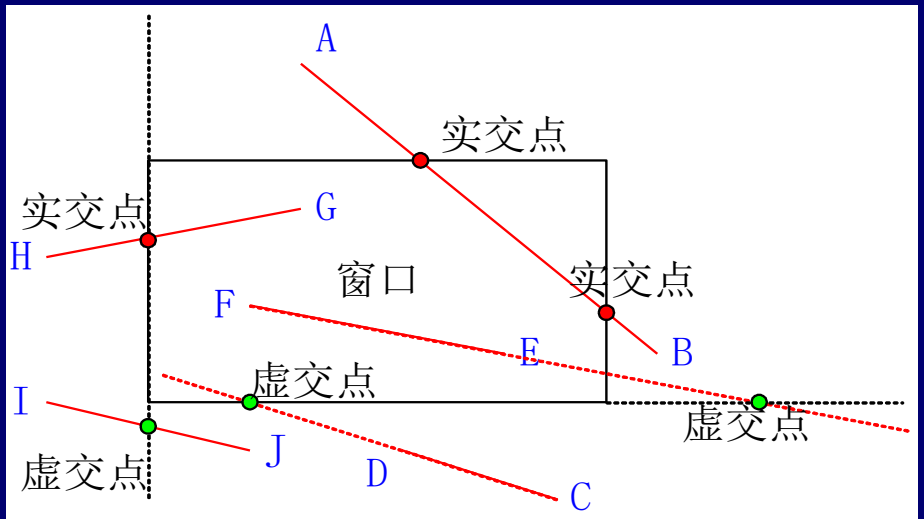


图6-25　实交点与虚交点

（1） Cohen-Sutherland算法

基本思想：对每条直线段$p_1(x_1,y_1)p_2(x_2,y_2)$分三种情况处理：

(1) 直线段完全可见，“简取”之。

(2) 直线段完全不可见，“简弃”之。

(3) 直线段既不满足“简取”的条件，也不满足“简弃”的条件，需要对直线段按交点进行分段，分段后重复上述处理。

编 码

编码：对于任一端点(x,y)，根据其坐标所在的区域，赋予一个4位的二进制码$D_3D_2D_1D_0$。

编码规则如下：

- 若x<wxl，则D_0=1，否则D_0=0；
- 若x>wxr，则D_1=1，否则D_1=0；
- 若y<wyb，则D_2=1，否则D_2=0；
- 若y>wyt，则D_3=1，否则D_3=0。

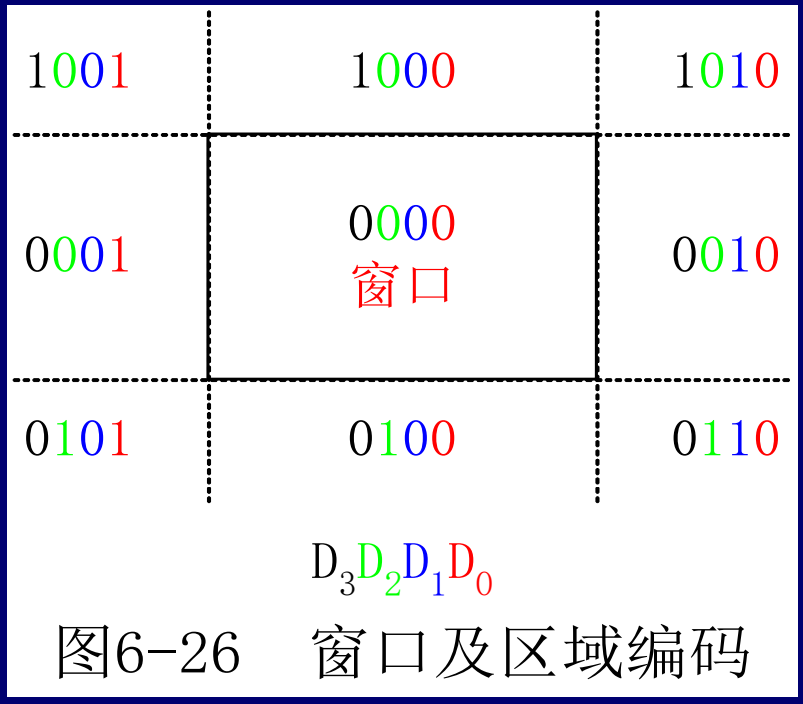


图6-26　窗口及区域编码

裁剪

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。

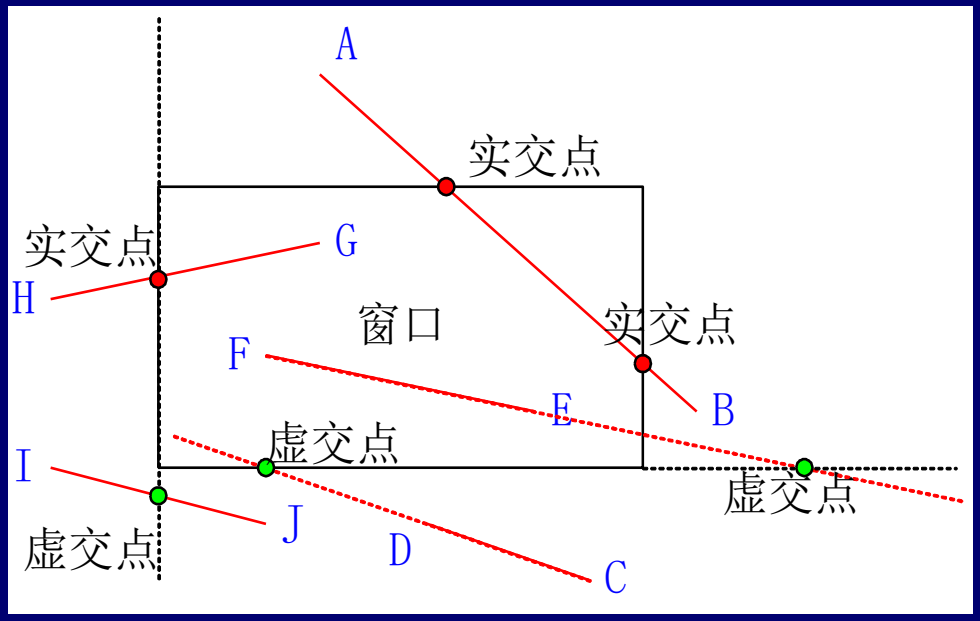
(2)若code1&code2≠0，对直线段可简弃之。

(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。

- **求交**：假定直线的端点坐标为(x_1,y_1)和(x_2,y_2)

左、右边界交点的计算：

上、下边界交点的计算：



算法的步骤：

(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，转(6)；否则，若code1&code2≠0，对直线段可简弃之，转(7)；当上述两条均不满足时，进行步骤(4)。

(4)确保p_1在窗口外部：若p_1在窗口内，则交换p_1和p_2的坐标值和编码。

(5)**按左、右、上、下的顺序求出直线段与窗口边界的交点**，并用该交点的坐标值替换p_1的坐标值。**也即在交点s处把线段一分为二，并去掉p_1s这一段。**考虑到p_1是窗口外的一点，因此可以去掉p_1s。转(2)。

(6)用直线扫描转换算法画出当前的直线段p_1p_2。

(7)算法结束。

例如：

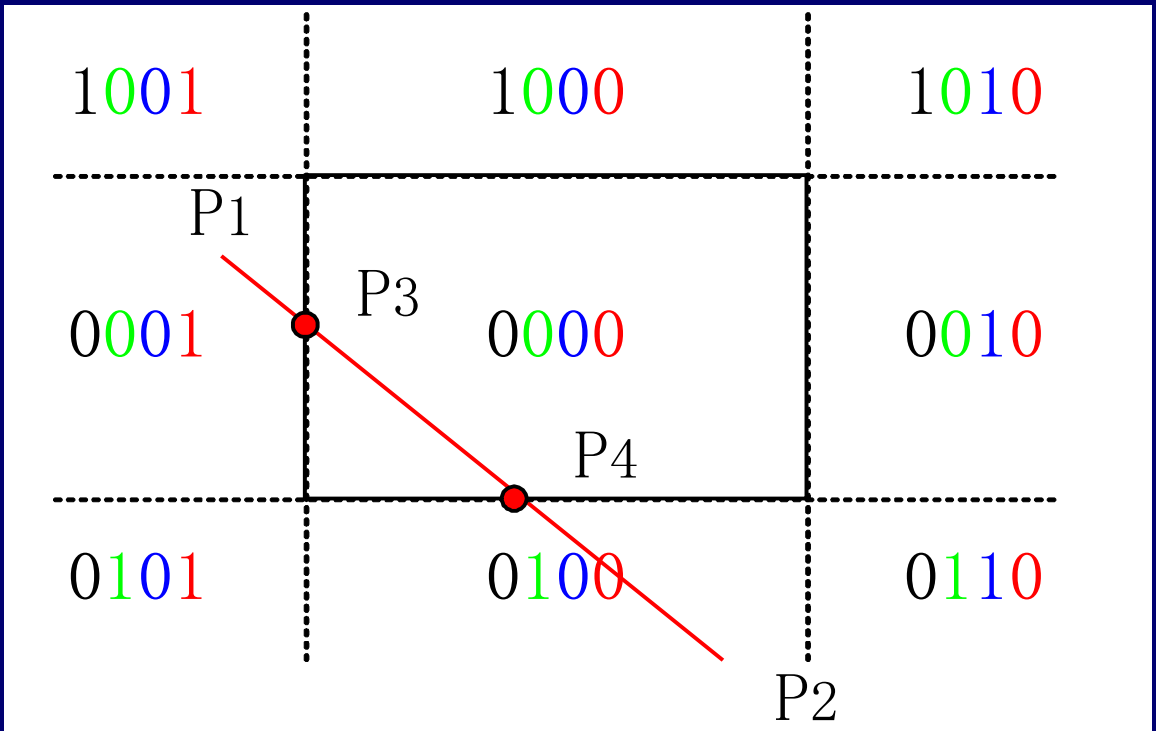


图6-27　直线段p_1p_2的编码裁剪

- 完整的算法:

(1) 检查线段P_1P_2是否为完全可见,或完全不可见,对于这两种情况,或完全取之,或完全弃之,否则执行(2);

(2) 找到P_1P_2在窗口外的一个端点:P_1或P_2.

(3) 用窗口的边与P_1P_2的交点取代端点:P_1或P_2;

(4) 再判断P_1P_2是否为完全可见，若是则取之，否则执行（2）。

（2） 中点分割算法

基本思想：

当对直线段不能简取也不能简弃时，简单地把线段等分为二段，对两段重复上述测试处理，直至每条线段完全在窗口内或完全在窗口外。

算法步骤：

(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。

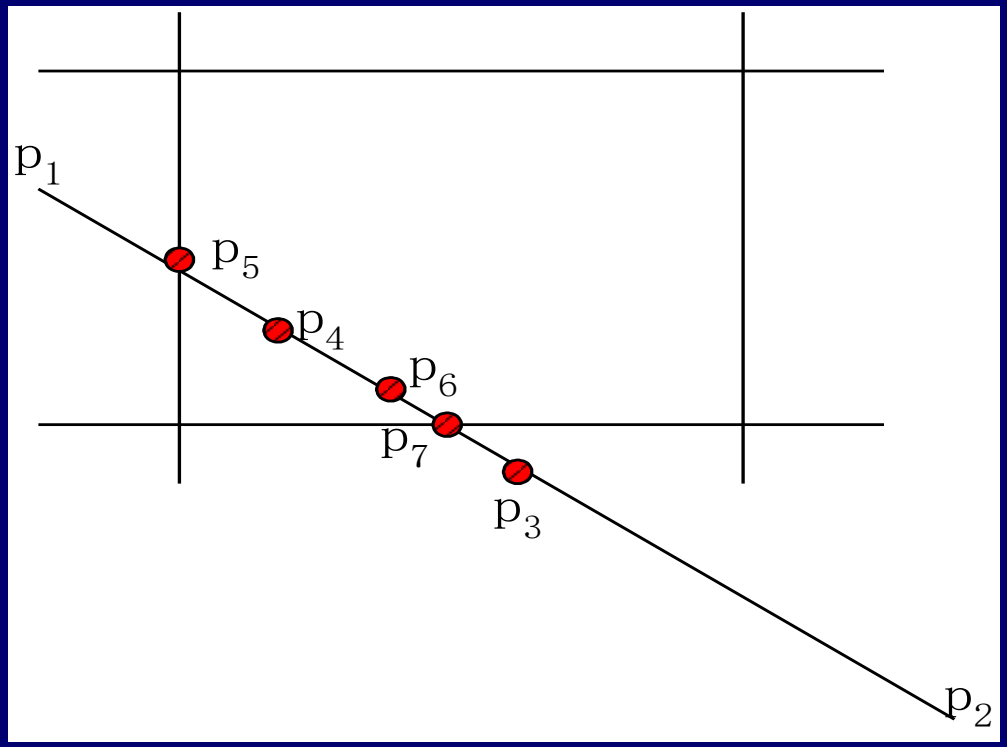
(2)对p_1、p_2进行编码：点p_1的编码为code1，点p_2的编码为code2。

(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。



中点分割算法的**核心思想**是通过二分逼近来确定直线段与窗口的交点。

重新构造**算法步骤**：

(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。

```
int mid_trim(p1,p2,xmin,ymin,xmax,ymax, a)
 {
            float p1[3],p2[3],xmin,ymin,xmax,ymax, a[3];
          float A[3],B[3],M[3];
          set_point(p1, A);
          set_point(p2, B);
           while(TRUE){
           if(Identify_line_out(A, B,xmin,ymin,xmax,ymax)==TRUE)
              retuen FALSE;
           if(Identify_pt_in(B, xmin,ymin,xmax,ymax)==TRUE){
              set_point(B, a);
              return TRUE;
           }
```

```
else{
        get_midpoint(A,B,M);
        if(Identify_line_out(M,B,xmin,ymin,xmax,ymax)==TRUE){
            set_point(M,B);
        }
        else{
            set_point(M,A);
         }
    }
}
```

（3）Liang-Barsky算法

1984年，梁友栋和Barsky共同提出。

RESEARCH CONTRIBUTIONS

A New Concept and Method for Line Clipping

YOU-DONG LIANG and BRIAN A. BARSKY
University of California, Berkeley

YOU-DONG LIANG，1956-1960年师从苏步青先生学习计算几何理论，1960年研究生毕业后任教于浙江大学数学系，1984-1990年任数学系主任。

美国加州大学伯克利分校计算机系终身教授
Brain A. Barsky教授

The line segment to be clipped is mapped into a parametric representation. From this, a set of conditions is derived that describes the interior of the clipping region. Observing that these conditions are all of similar form, they are rewritten such that the solution to the clipping problem is reduced to a simple max/min expression。

图1 算法原文

2.3 Algorithm

The algorithm to perform two-dimensional line clipping is as follows:

```
(*clip 2d line segment with endpoints (x0, y0) and (x1, y1)*)
(*the clip window is xleft <= x <= xright and ybottom <= y <= ytop*)

procedure clip2d (var x0, y0, x1, y1: real);
var t0, t1: real;
  deltax, deltay: real;

function clipt ((*value*) p, q: real; var t0, t1; real): Boolean;
var r: real;
  accept: Boolean;
begin (*clipt*)
  accept := true;
  if p < 0 then begin
    r := q/p;
    if r > t1 then accept := false (*set up to reject*)
    else if r > t0 then t0 := r
  end else if p > 0 then begin
    r := q/p;
    if r < t0 then accept := false (*set up to reject*)
    else if r < t1 then t1 := r
  end else (*p = 0*) if q < 0 then accept := false; (*set up to reject*)
  clipt := accept
end (*clipt*);

begin (*clip2d*)
  t0 := 0;
  t1 := 1;
  deltax := x1 - x0;
  if clipt (- deltax, x0 - xleft, t0, t1) then
    if clipt (deltax, xright - x0, t0, t1) then begin
      deltay := y1 - y0;
      if clipt (- deltay, y0 - ybottom, t0, t1) then
        if clipt (deltay, ytop - y0, t0, t1) then begin
          if t1 < 1 then begin
            x1 := x0 + t1 * deltax;
            y1 := y0 + t1 * deltay
          end;
          if t0 > 0 then begin
            x0 := x0 + t0 * deltax;
            y0 := y0 + t0 * deltay
          end
        end
    end
end (*clip2d*);
```


算法原文

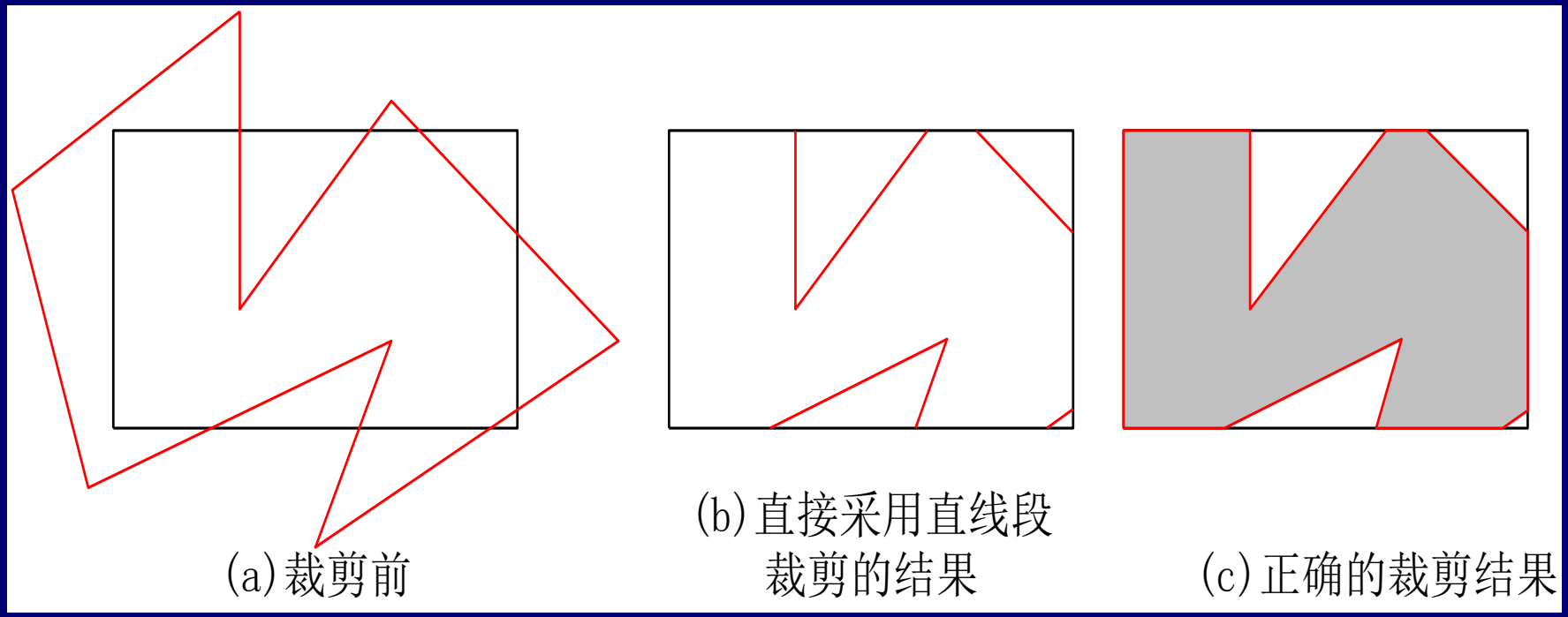
核心亮点：
把二维裁剪问题转化为二次一维裁剪

（4） 其它直线裁剪算法

- Cyrus-Beck算法
- Nicholl-Lee-Nicholl算法

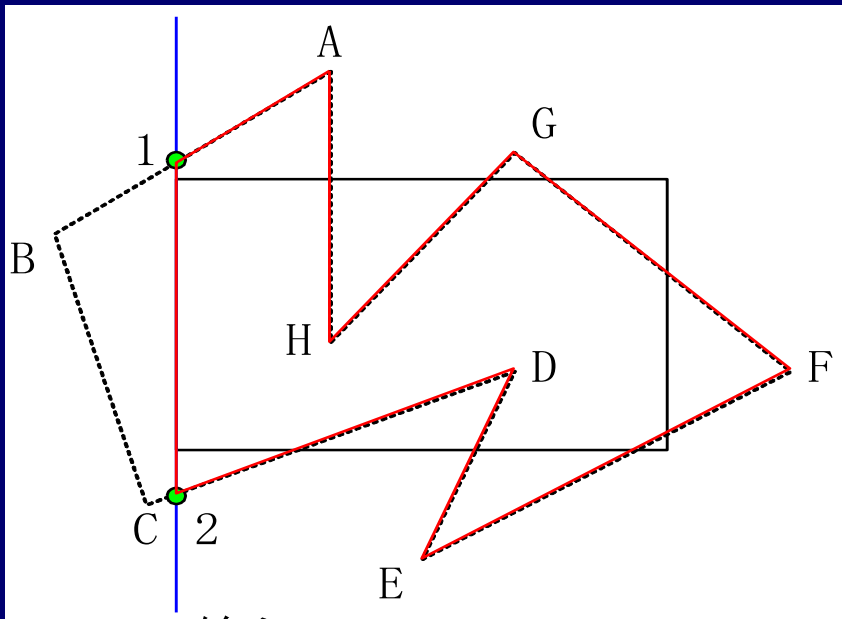
三、多边形的裁剪

- 问题的提出：

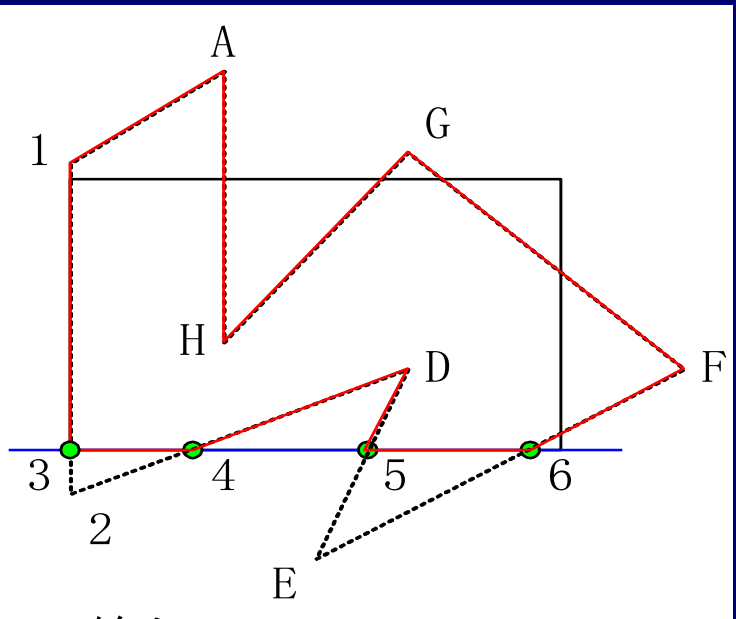


（1） Sutherland-Hodgeman多边形裁剪

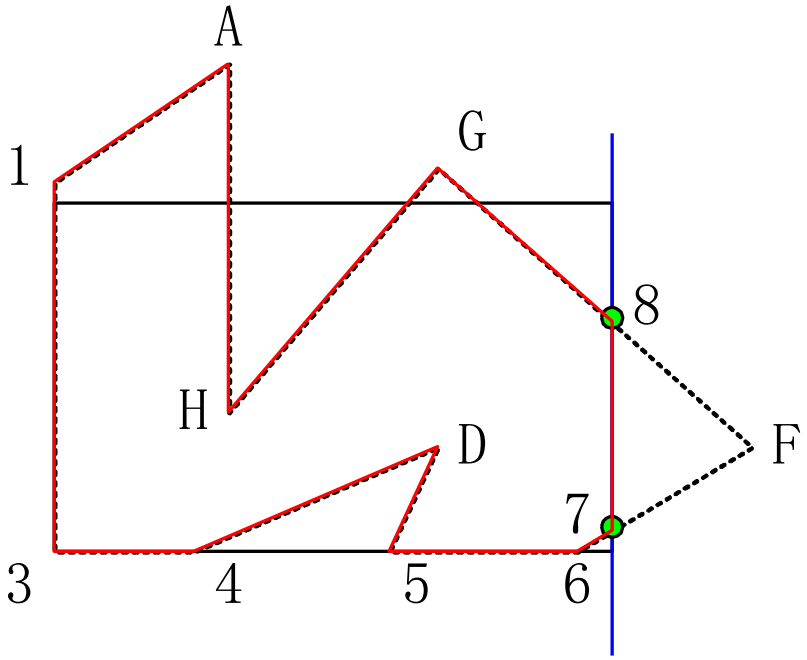
基本思想



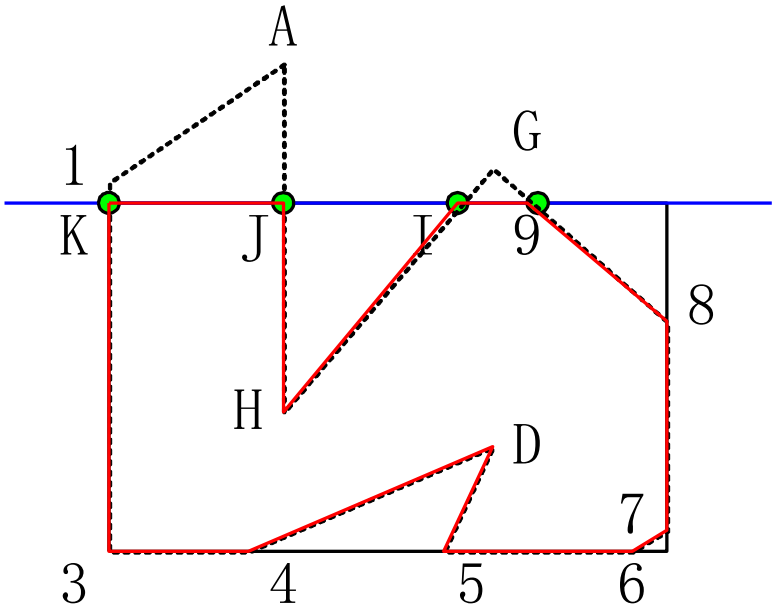
输入： ABCDEFGH
输出： A12DEFGH
(a)用左边界裁剪



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

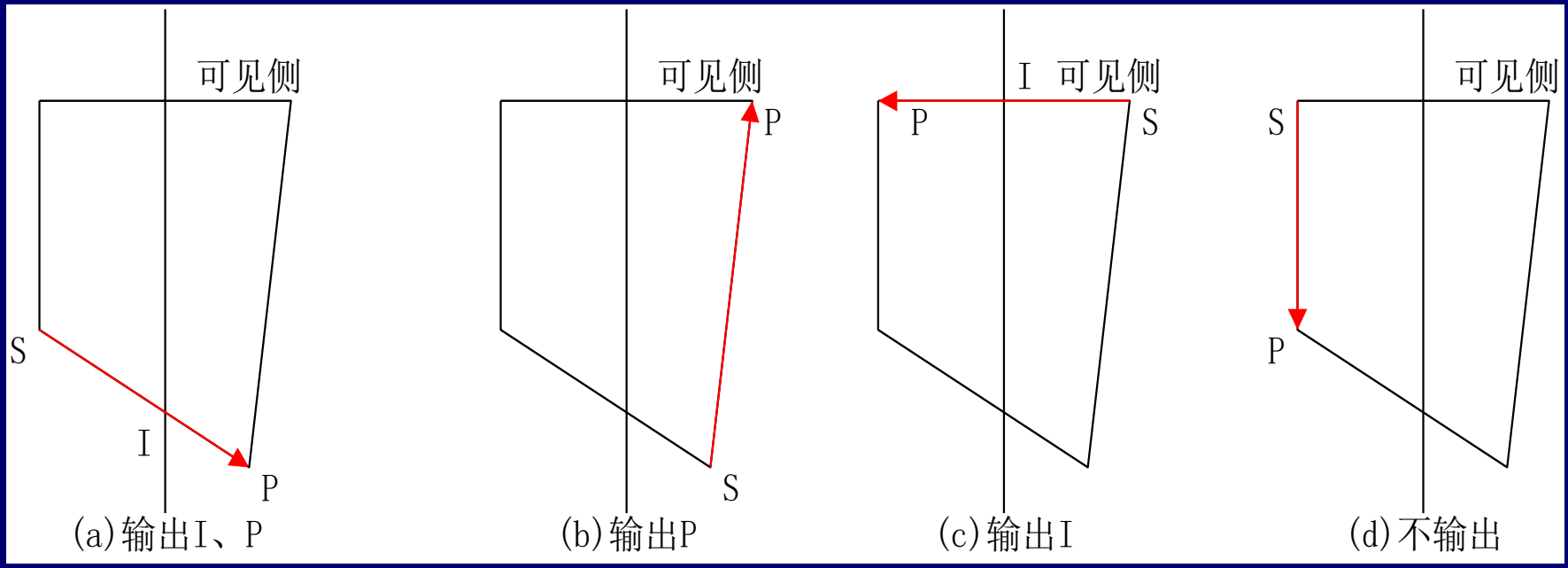


输入：A134D5678GH
输出：K34D56789IHJ
(d)用上边界裁剪

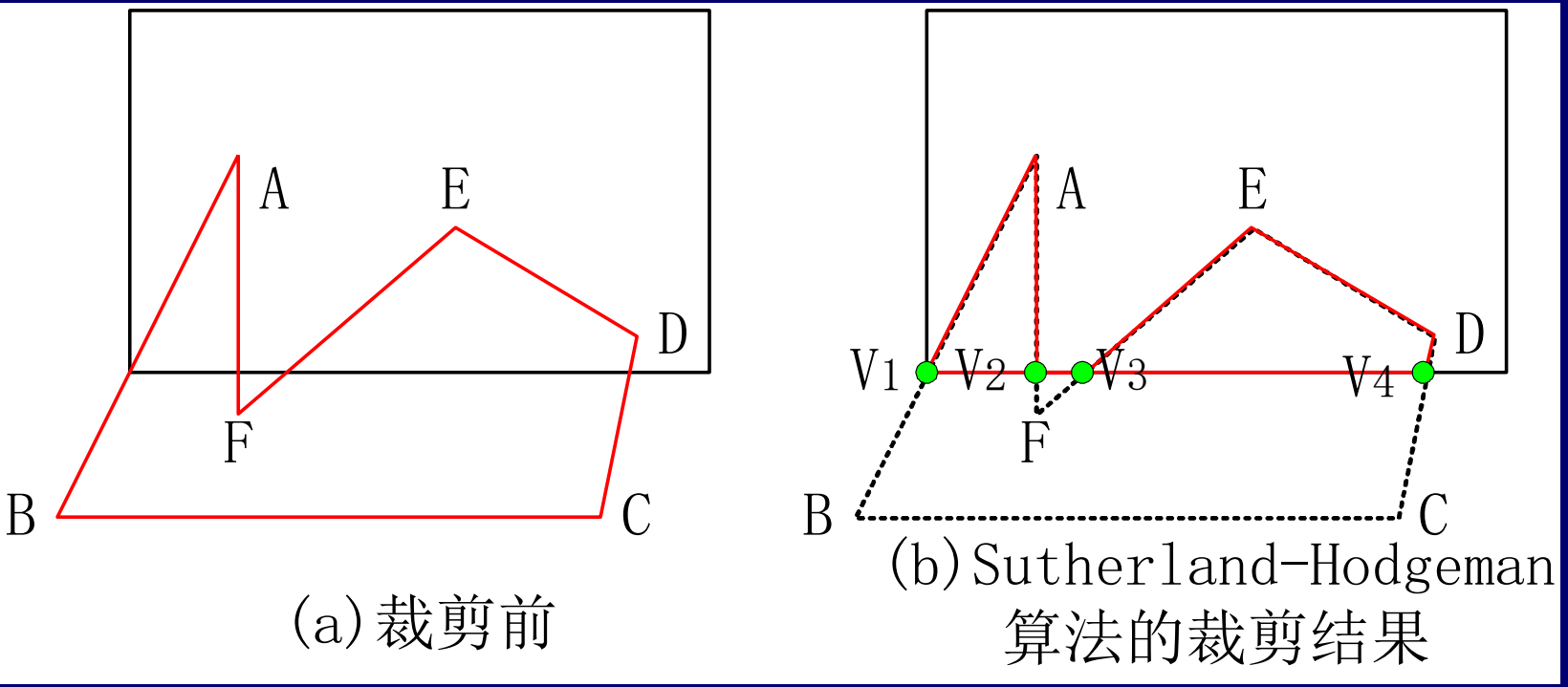
算法实施策略：

- 为窗口各边界裁剪的多边形存储输入与输出顶点表。在窗口的一条裁剪边界处理完所有顶点后，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，包含有窗口区域的一个域称为**可见侧**；不包含窗口区域的域为**不可见侧**。

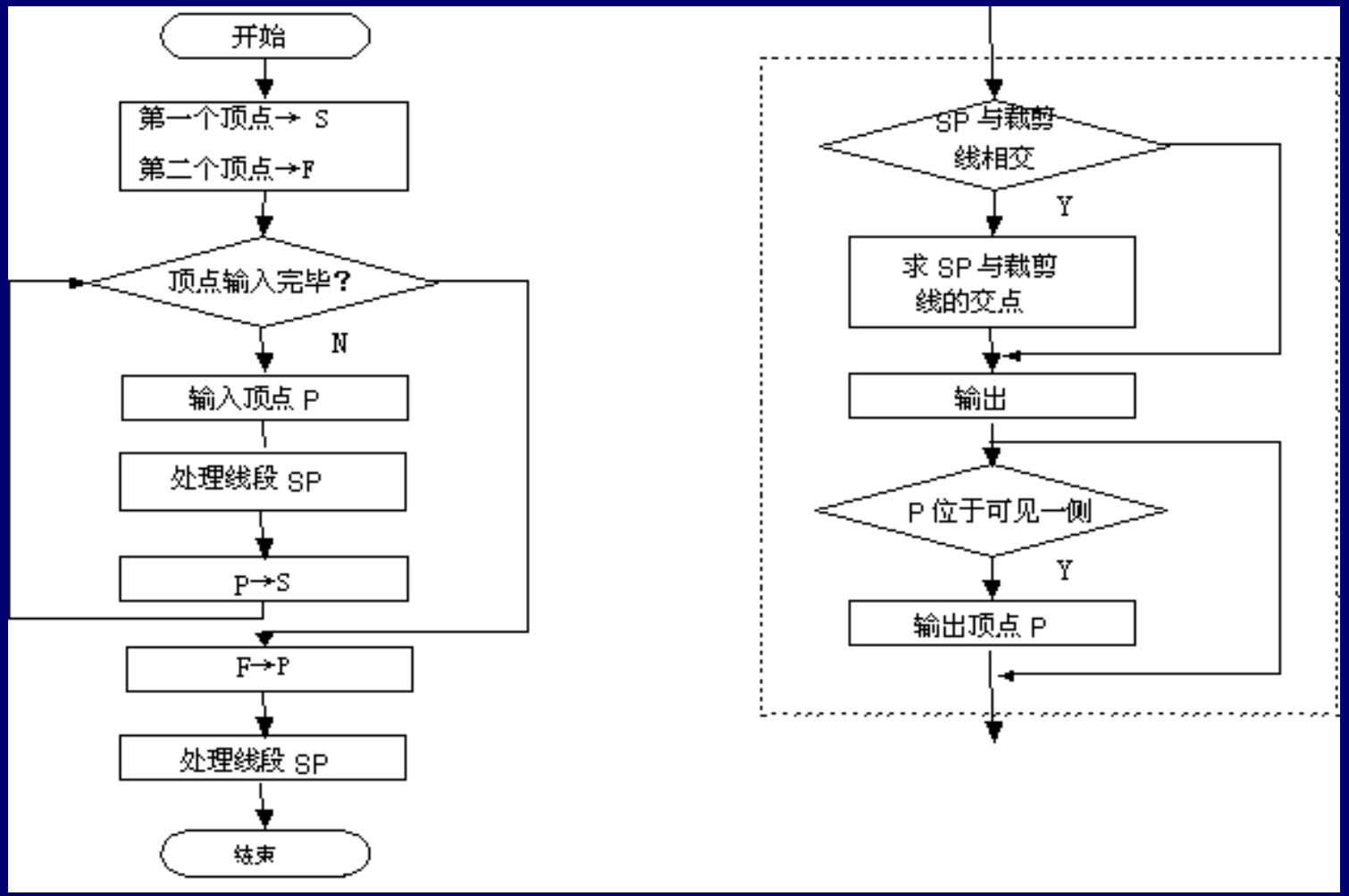
- 沿着多边形依次处理顶点会遇到四种情况：



特点：

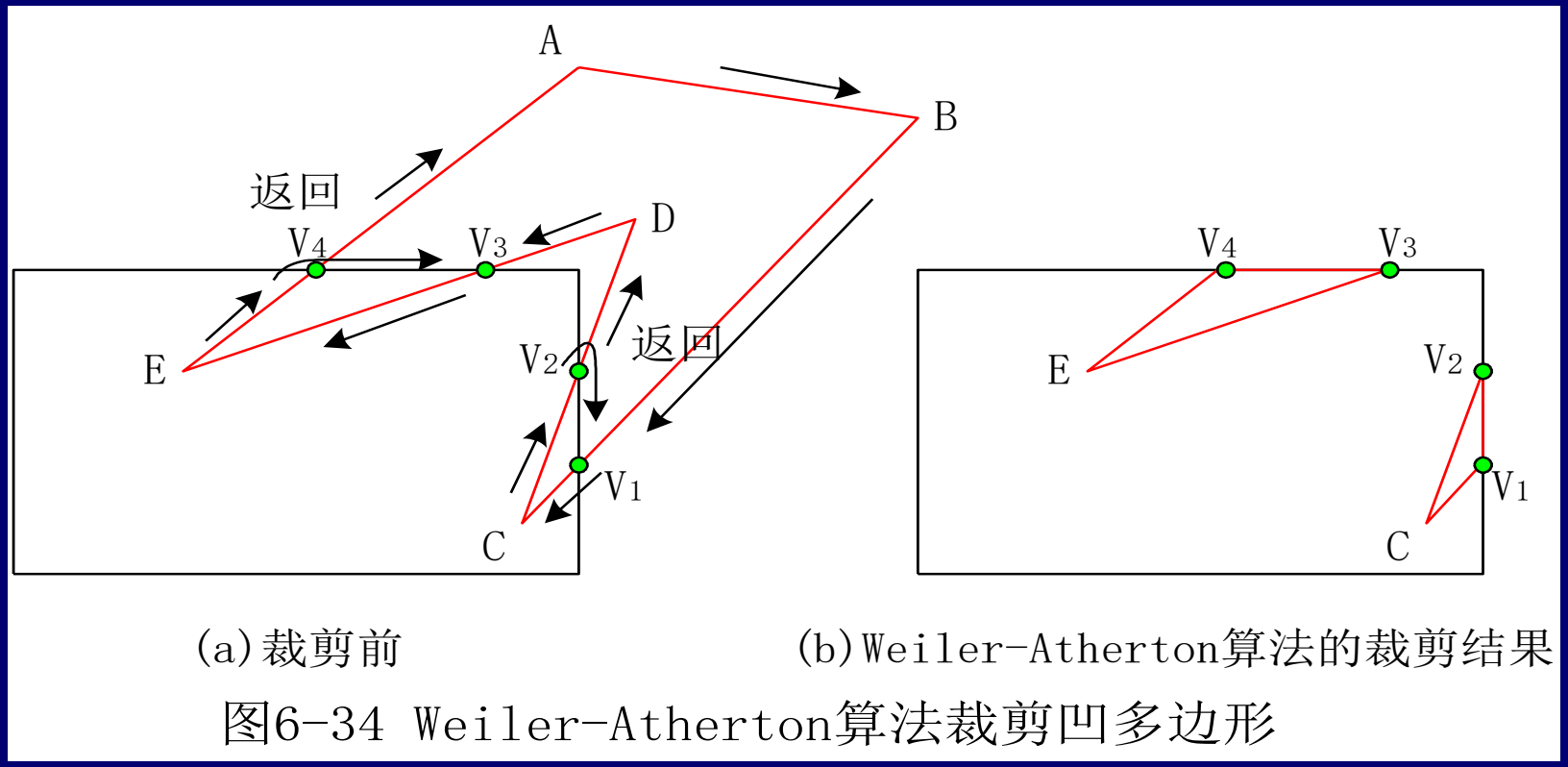

A
E
D
F
B
C
(a)裁剪前
A
E
D
V1
V2
V3
V4
F
B
C
(b)Sutherland-Hodgeman
算法的裁剪结果

逐次多边形裁剪算法框图



（2） Weiler-Atherton多边形裁剪

- 下图示了Weiler-Atherton算法裁剪凹多边形的过程和结果



(a)裁剪前　　(b)Weiler-Atherton算法的裁剪结果

图6-34 Weiler-Atherton算法裁剪凹多边形

假定按顺时针方向处理顶点，且将用户多边形定义为Ps，窗口矩形为Pw。算法从Ps的任一点出发，跟踪检测Ps的每一条边，当Ps与Pw相交时（实交点），按如下规则处理：

(1)若是由不可见侧进入可见侧，则输出可见直线段，转(3)；

(2)若是由可见侧进入不可见侧，则从当前交点开始，沿窗口边界顺时针检测Pw的边，即用窗口的有效边界去裁剪Ps的边，找到Ps与Pw最靠近当前交点的另一交点，输出可见直线段和由当前交点到另一交点之间窗口边界上的线段，然后返回处理的当前交点；

(3)沿着Ps处理各条边，直到处理完Ps的每一条边，回到起点为止。

四、其它裁剪

1. 文字裁剪

文字裁剪的策略包括几种：

- 串精度裁剪
- 字符精度裁剪
- 笔划、象素精度裁剪

2. 外部裁剪

保留落在裁剪区域外的图形部分、去掉裁剪区域内的所有图形，这种裁剪过程称为**外部裁剪**，也称**空白裁剪**。

END