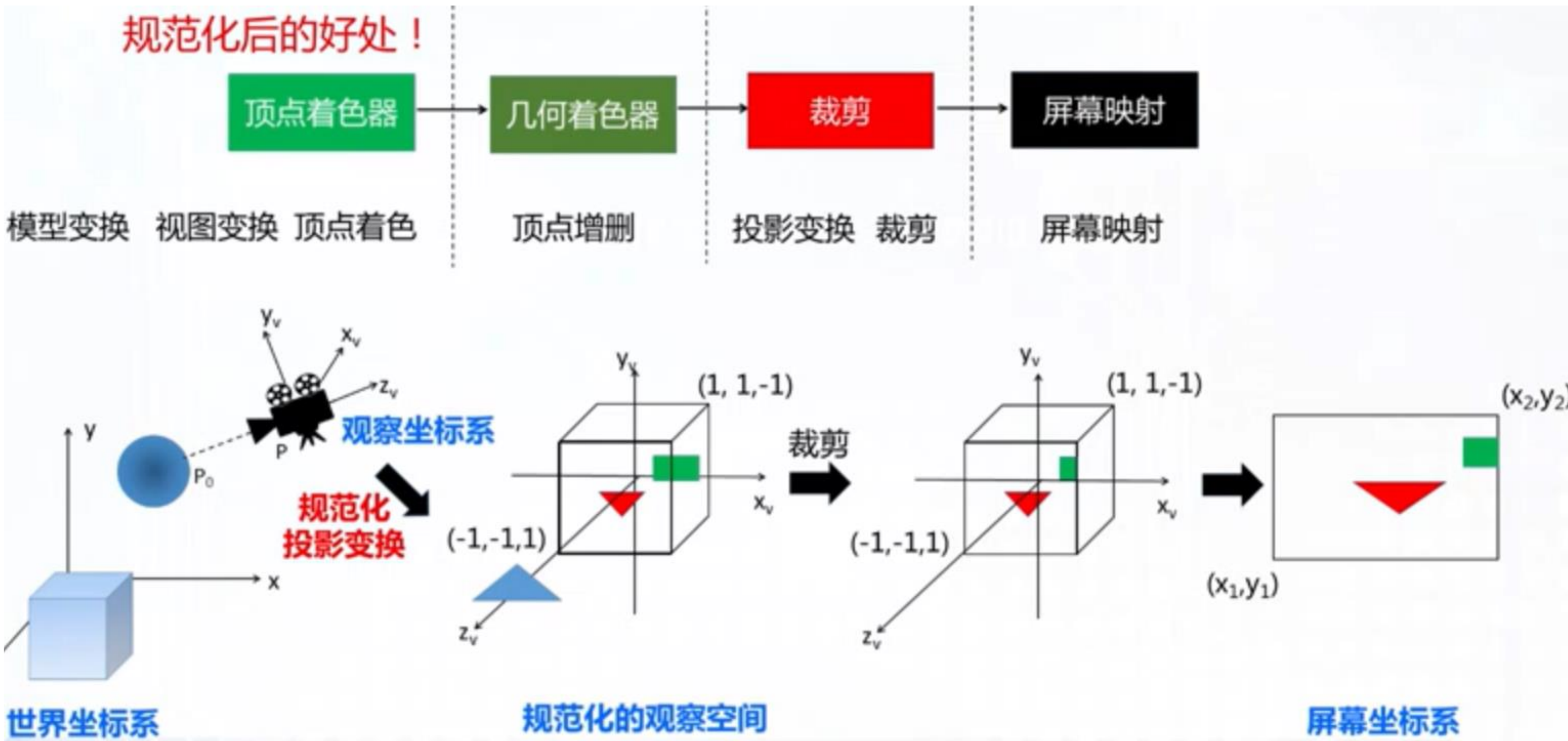
裁剪和屏幕映射



规范化后的好处！
顶点着色器
几何着色器
裁剪
屏幕映射
模型变换 视图变换 顶点着色
顶点增删
投影变换 裁剪
屏幕映射
观察坐标系
规范化
投影变换
(1, 1,-1)
(-1,-1,1)
裁剪
(1, 1,-1)
(-1,-1,1)
(x_2,y_2)
(x_1,y_1)
世界坐标系
规范化的观察空间
屏幕坐标系

内容提要

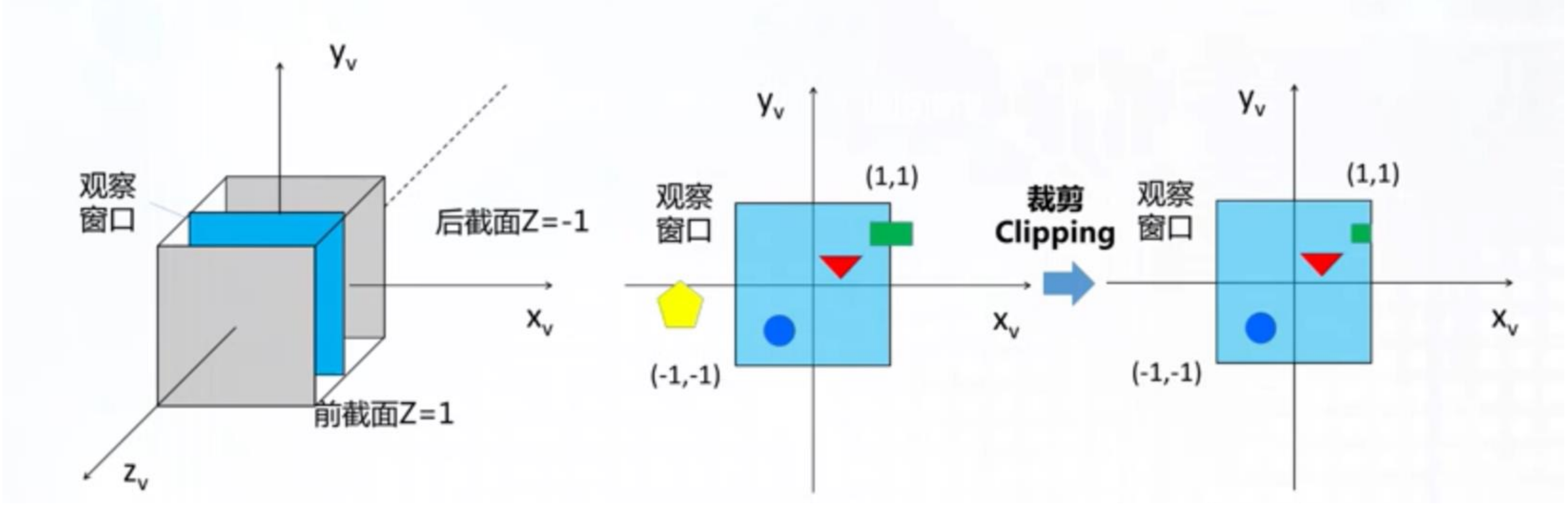
- 裁剪（以2D为例，推广到3D）
  - 点裁剪
  - 直线裁剪
    - Cohen-Sutherland算法（*）
    - Liang-Barsky算法（*）
  - 多边形裁剪
    - Sutherland-Hodgeman逐边裁剪算法（*）

裁剪算法的应用

- 从定义的场景中取出用于观察的部分；
- 显示多窗口的环境；
- 在三维图形中标识出可见面(3D观察)；
- 防止线段或对象的边界混淆；
- 用实体造型来创建对象（计算相交）；
- 允许选择图形的一部分进行拷贝、移动或删除等交互绘图操作。

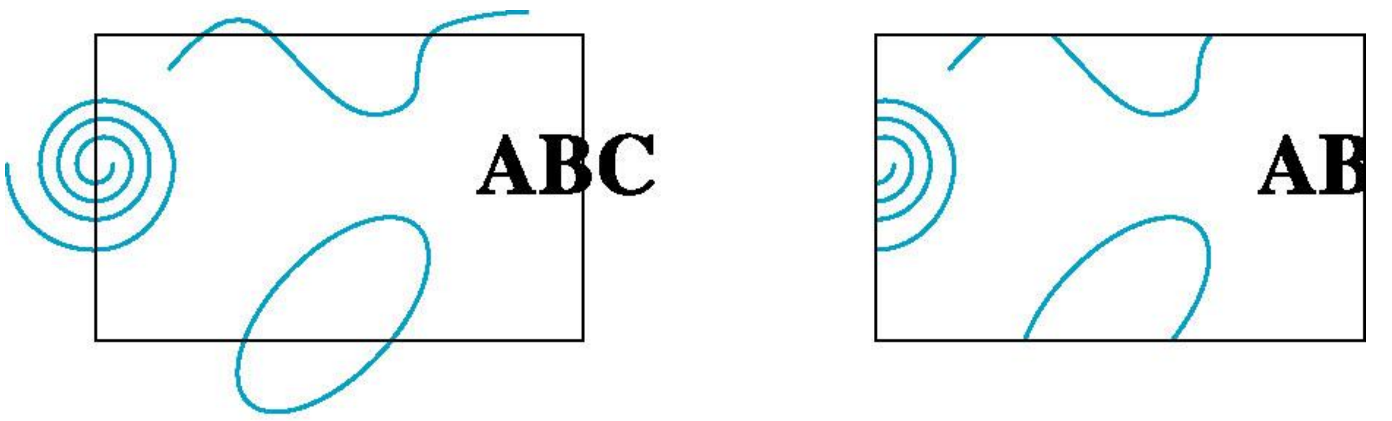
裁剪clipping

- 对图形指定区域内和区域外部分的过程称为裁剪。



裁剪窗口和裁剪对象

- **裁剪窗口:**
  - 一般是矩形: 由上（y=Ymax）、下（y=Ymin）、左（x=Xmin）、右（x=Xmax）四条边描述
- **裁剪对象:**
  - 点，直线段，区域，曲线，文字



一、点裁剪Point Clipping

- 点P=(x, y)是裁剪窗口内的点，

满足：

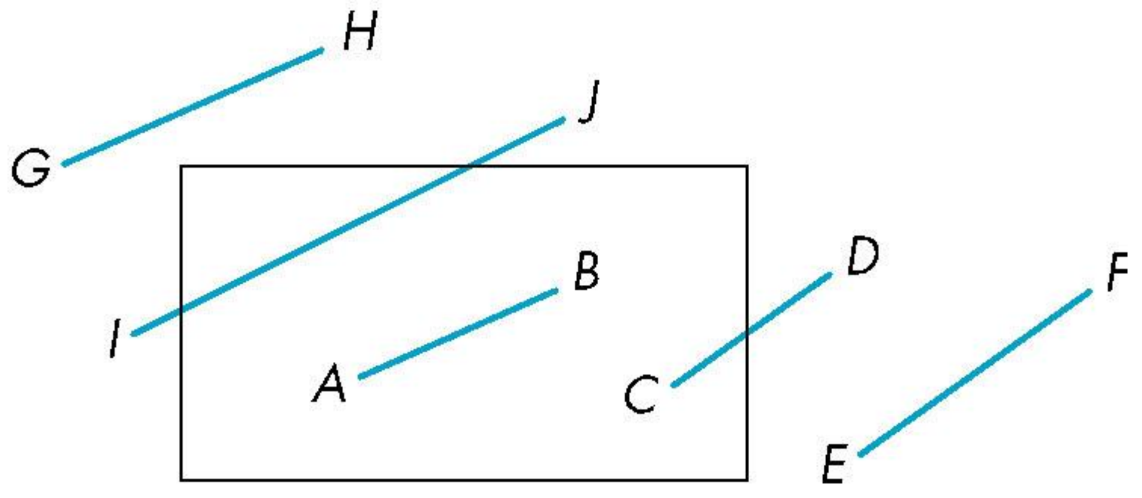
$$x_{wmin} \leqslant x \leqslant x_{wmax}$$

and

$$y_{wmin} \leqslant y \leqslant y_{wmax}$$

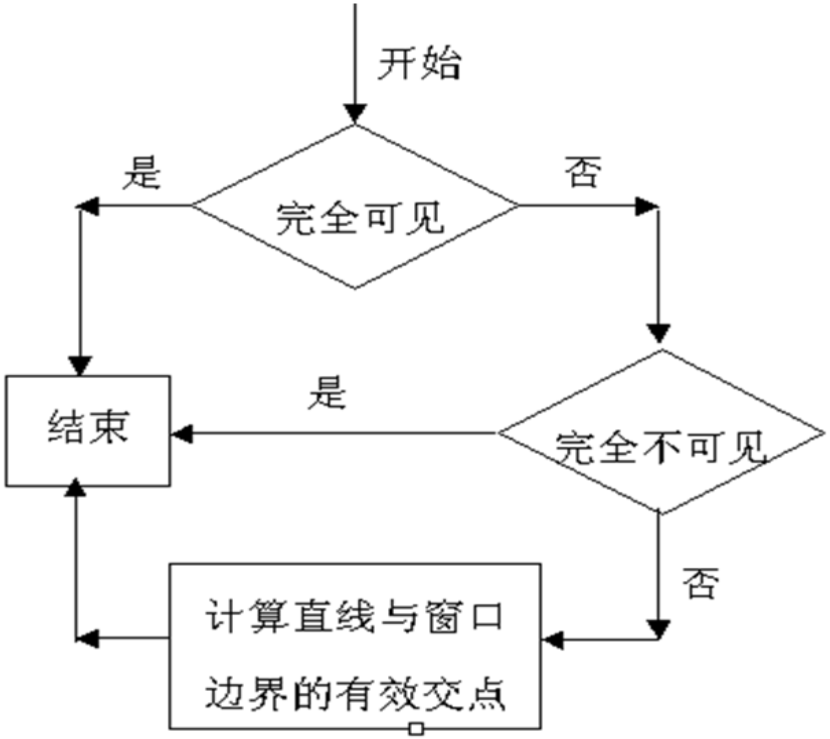
二、线裁剪Line Clipping

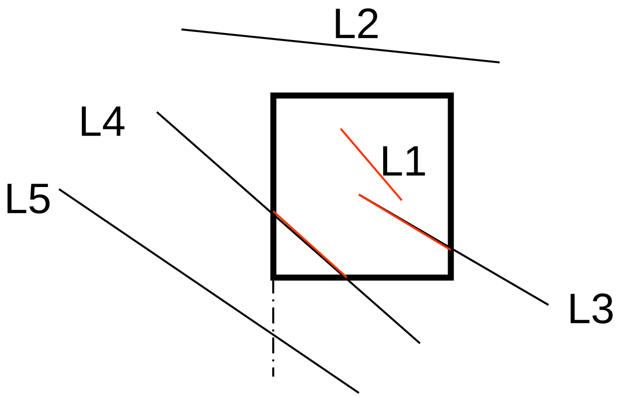
- 直线与窗口3种关系：
  - 完全窗口内，
  - 完全窗口外，
  - 部分窗口内部分窗口外



1、Cohen-Sutherland Algorithm

- 最早最流行的，
- 用编码方式来判定端点在窗口内还是外，
- 通过初始测试来减少计算交点数
- 理想状态下，不求交就去除许多不可见线段



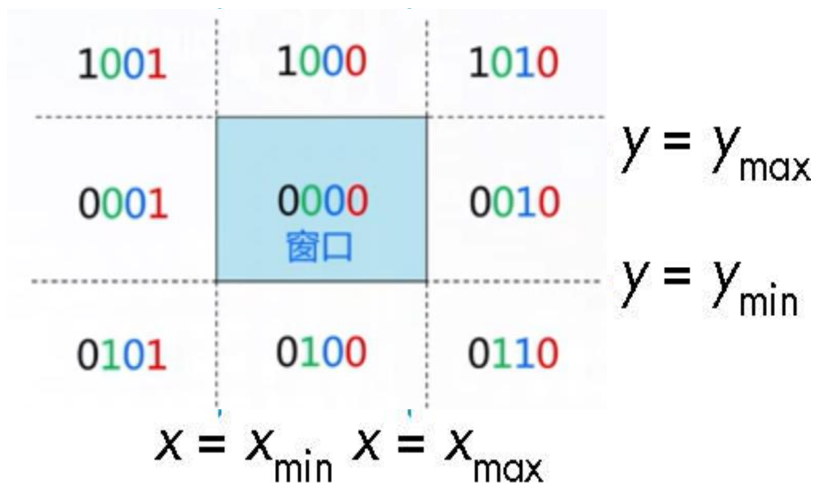


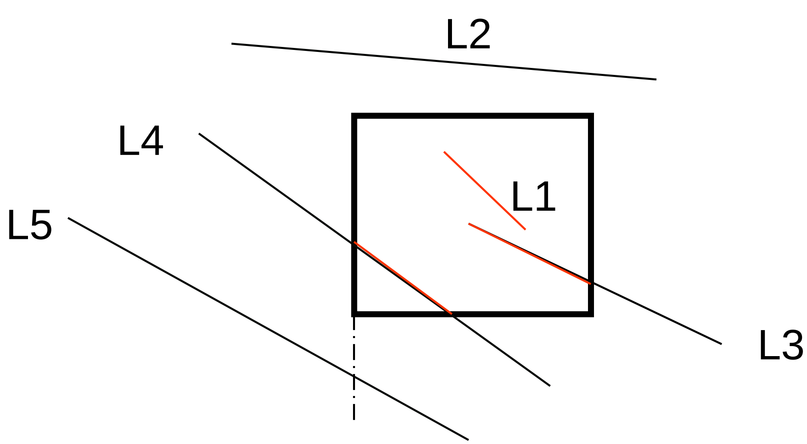
编码Defining Outcodes

For each endpoint, define an outcode，Outcodes divide space into 9 regions

$$b_0b_1b_2b_3$$

b_0 = 1 if $y > y_{max}$, 0 otherwise
b_1 = 1 if $y < y_{min}$, 0 otherwise
b_2 = 1 if $x > x_{max}$, 0 otherwise
b_3 = 1 if $x < x_{min}$, 0 otherwise





L1(0000 0000) 完全可见 **accept**
L2(1010 1001) 完全不可见 **reject**
L3(0000 0110) 部分可见
L4(0001 0100) 部分可见
L5(0001 0100) 完全不可见 **?reject**

算法描述

1.大窗口的场合

2.窗口特别小的场合

演示

- 1. code1|code2=0,两端点P_1、P_2的区域码均为0000，则完全可见（两端点在裁剪窗口内），结束并输出线段。
- 2. code1&code2!=0, 两端点P_1、P_2的区域码做位"与"操作，结果不为0000，则完全不可见（两端点同一外侧），结束。
- 3. 否则，线段有可能是部分可见或完全不可见。则求交点得新线段，再判定其是否可见（转1，2操作）。

  a、判断P_1区域码，如果在窗口内，则交换P_1、P_2,(保证P1在窗口外)。

  b、用线段P_1P_2与窗口某边的有效交点代替P_1,并求出P_1的区域码后，转1进入下一轮循环。

```
#define ROUND(a) ((int)(a+0.5))
#define LEFT_EDGE 0x1//左裁剪边0001
#define RIGHT_EDGE 0x2//右裁剪0010
#define BOTTOM_EDGE 0x4//下裁剪边0100
#define TOP_EDGE 0x8//上裁剪边1000
#define INSIDE(a) (!a)//端点a在区域内
#define REJECT(a,b) (a&b)//线段完全在区域外
#define ACCEPT(a,b) (!(a|b))//线段完全在区域内

/*将某点pt(x,y)根据裁剪窗口得到其区域编码 */
Unsigned char encode(wcPt2 pt,  dcPt winMin, dcPt winMax){
    unsigned char code=0x00;
    if (pt.x<winMin.x) code=code | LEFT_EDGE
    if (pt.x>winMax.x)  code=code | RIGHT_EDGE
    if (pt.y<winMin.y)  code=code | BOTTOM_EDGE
    if (pt.y>winMax.y)  code=code | TOP_EDGE;
    return(code);
}//注意：裁剪边界上的交点的编码为0000
```

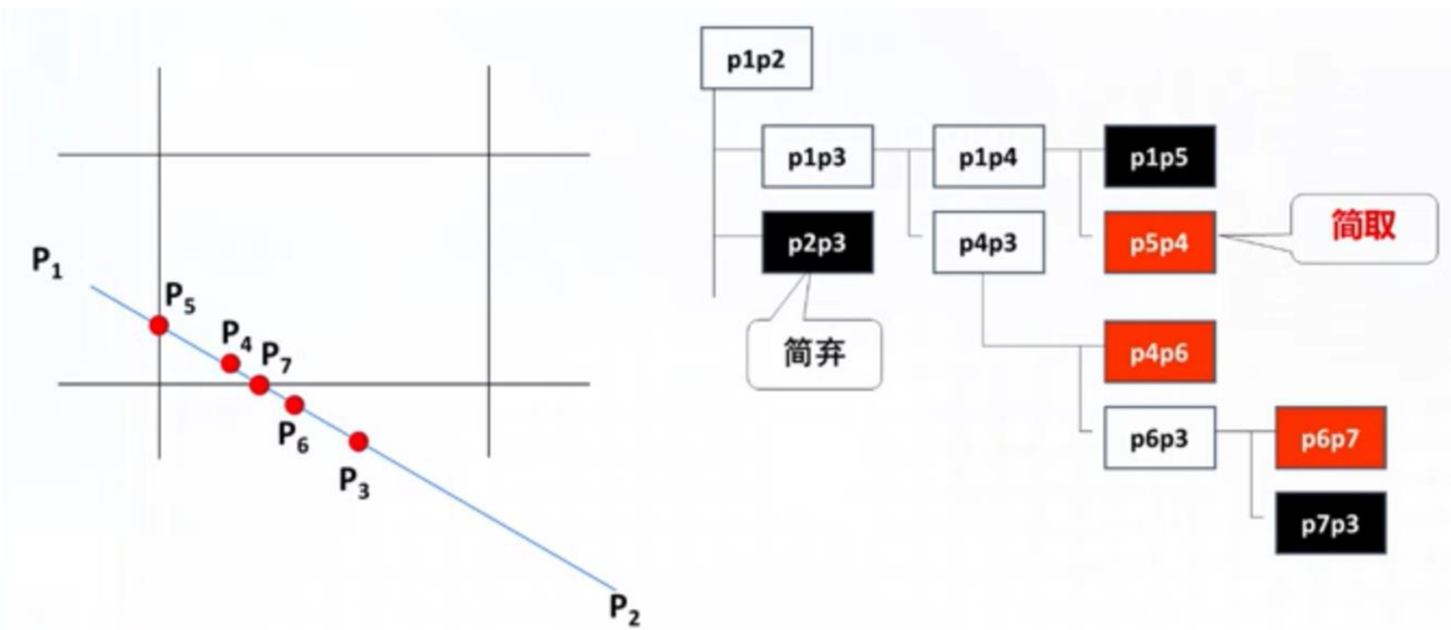
```
/*交换p1点和p2点的坐标*/
void swapPts(wcPt2 *p1,wcPt2 *p2)
{
   wcPt2 tmp;
    tmp=*p1;    *p1=*p2;     *p2=tmp;
}

/*交换p1点和p2点的区域编码*/
void swapCodes(unsigned char *c1,unsigned char *c2)
{
   unsigned char tmp;
   tmp=*c1;    *c1=*c2;      *c2=tmp;
}
```

```
Void clipLine(dcPt winMin,dcPt winMax,wcPt2 p1,wcPt2 p2){
 unsigned char code1,code2;                    float m;//求交时计算的斜率
 int done=FALSE; //循环结束标志;             int draw=FALSE; //有裁剪结果标志
while(!done){
 code1=encode(p1,winMin,winMax);  code2=encode(p2,winMin,winMax);
 if (ACCEPT(code1,code2)){  //完全在区域内：设置循环结束，绘制标志
       done=true;        draw=true;    }
 else if (REJECT(code1,code2)) //完全在区域外：设置循环结束标志。
        done=true;
 else {//不能判定，求交得新线段后，继续循环
   if (INSIDE(code1)){
    swapPts(&p1,&p2);  swapCodes(&code1,&code2);  //让p1点在窗口外}
    if(p2.x!=p1.x){
     m=(p2.y-p1.y)/(p2.x-p1.x); );
     if (code1&LEFT_EDGE){ //p1和左边界在同一外侧面才有交
       p1.y= p1.y +(winMin.x-p1.x)*m；  p1.x=winMin.x; //左裁剪边求交点,赋给p1}
       else if (code1 & RIGHT_EDGE){//p1和右边界在同一外侧面才有交
           p1.y= p1.y+(winMax.x-p1.x)*m;   p1.x=winMax.x; //右裁剪边求交点,赋给p1 }
           else   if (p2.y!=p1.y){
                 m=(p2.x-p1.x)/(p2.y-p1.y);   //x=x1+ (Ymin-y1)*m
                 if(code1 &BOTTOM_EDGE){//p1和下边界在同一外侧面才有交
                     p1.x= p1.x+(winMin.y-p1.y)*m;  p1.y=winMin.y; //下裁剪边求交点赋给P1}
                 else if (code1&TOP_EDGE){//p1和上边界在同一外侧面才有交
                      p1.x= p1.x+(winMax.y-p1.y)*m;     p1.y=winMax.y; //上裁剪边交点赋给p1  }
    }   //结束一次求交，只用了四个裁剪边中的一个.end - if(p2.x!=p1.x){
   } //end -if (INSIDE(code1)){
}//end while ,循环结束条件done=true(完全可见或完全不可见时可以退出循环
```

改进：中点分割裁剪Mid-Point Line Clipping

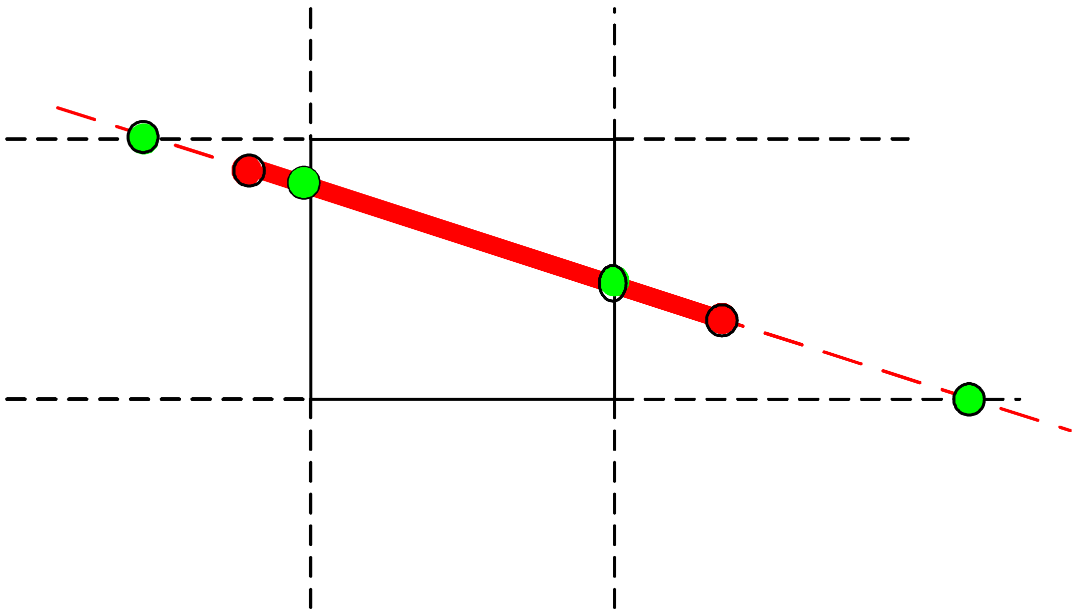
- 二分法求交点
  - 计算((x1+x2)/2,(y1+y2)/2)只用到加法和除2运算，效率高
  - 适合并行计算



本质：用二分逼近的方法求线段与窗口边界的交点

3、Liang-Barsky Line Clipping

- 思想
  - 裁剪线段的端点是窗口边界上交点或线段端点
  - 基于直线的参数化表示，直接计算得裁剪后线段端点

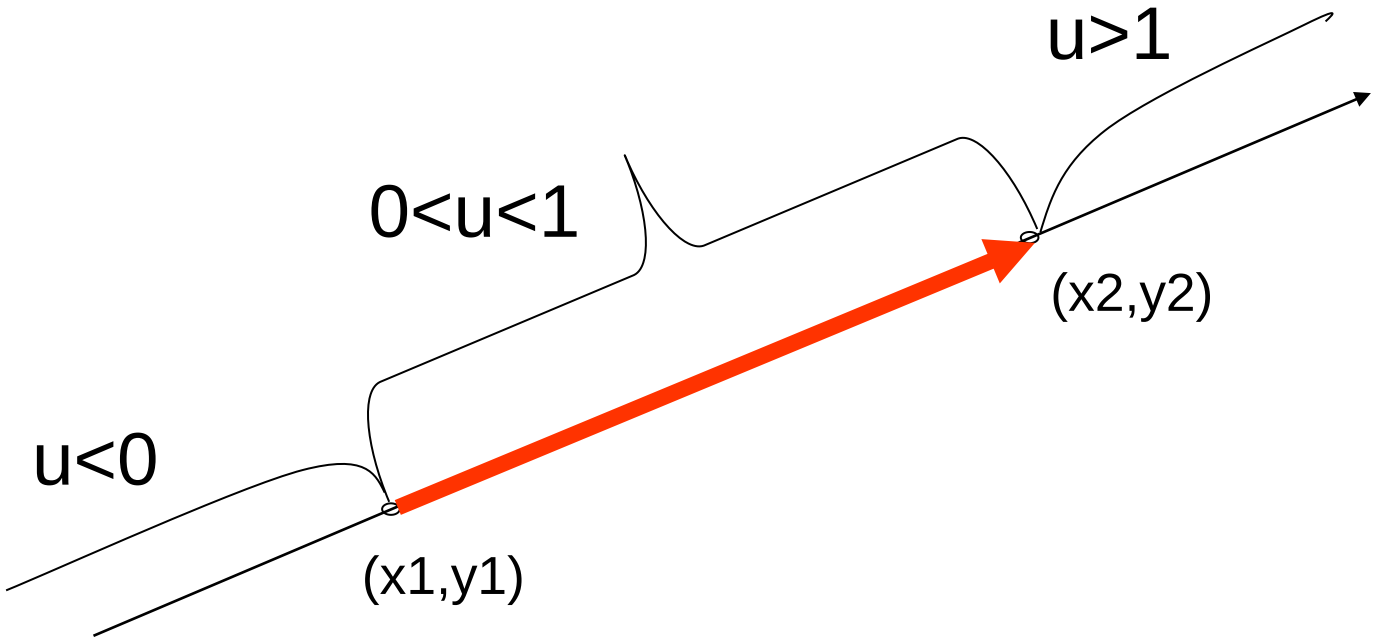


直线的参数化表示

$x = x_1 + u(x2 - x1)$

$y = y_1 + u(y2 - y1)$

$P(u) = P1 + u(P2 - P1) \quad 0 \le u \le 1$



窗口内线段的限定条件

1）在线段上　　$0 \le u \le 1$

2）在窗口内

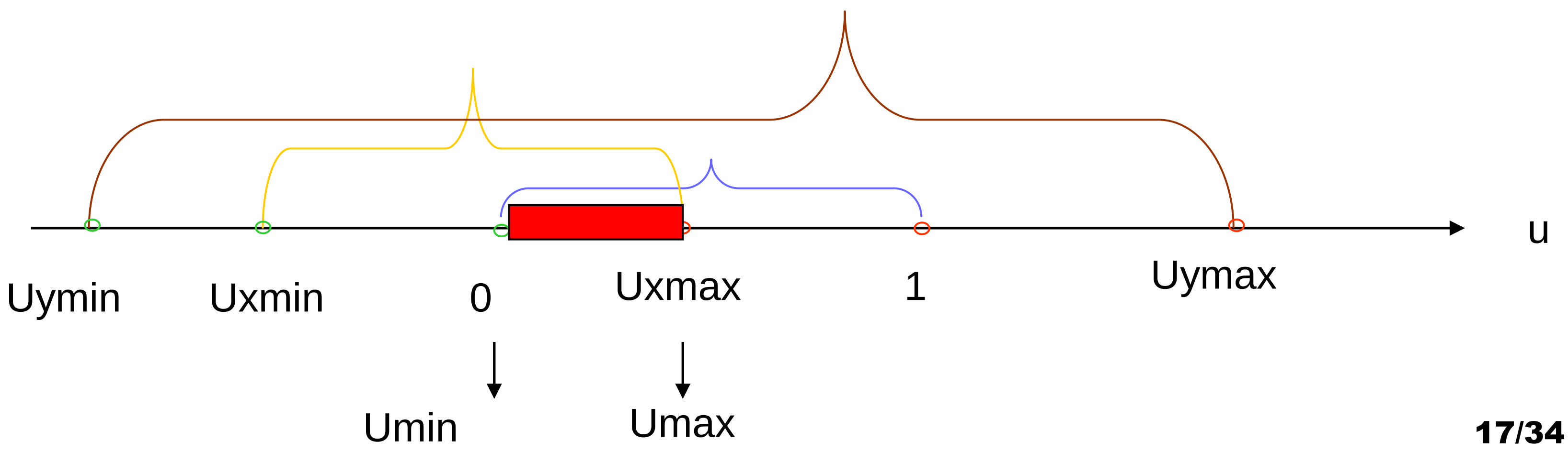
$x_{wmin} \le x1 + u\triangle x \le x_{wmax}$

$y_{wmin} \le y1 + u\triangle y \le y_{wmax}$,

$$xw_{min} \le x_1 + u\triangle x \le xw_{max}$$
$$yw_{min} \le y_1 + u\triangle y \le yw_{max}$$
$$0 \le u \le 1$$

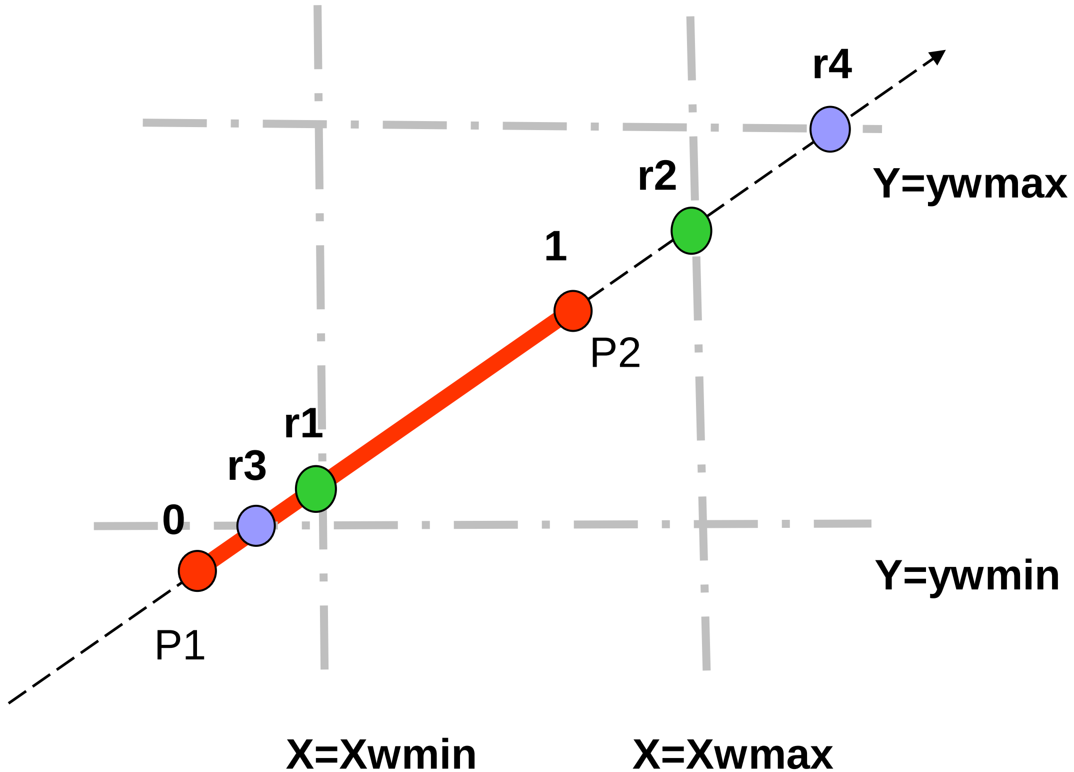
$\Rightarrow$

$$Uxmin \le u \le Uxmax$$
$$Uymin \le u \le Uymax$$
$$0 \le u \le 1$$

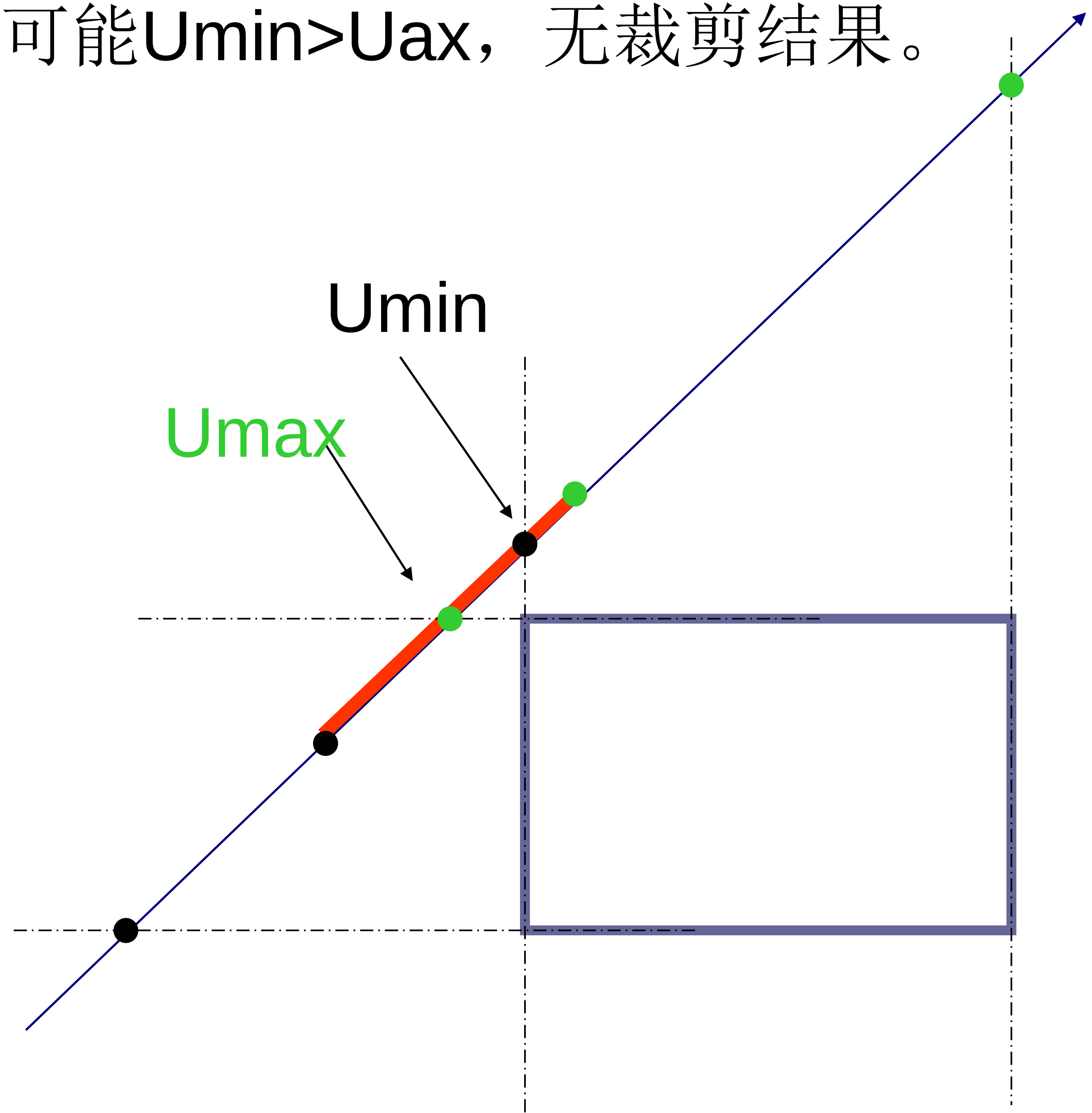


计算思路

- Umin=MAX(0，r1，r3)=r1
- Umax=MIN(1, r2， r4)=1
- Umin<Umax时，umin,umax带入线段参数方程，求出端点。



- 可能Umin>Uax，无裁剪结果。



主要公式推导

- $xw_{min} \le x_1 + u\triangle x \le xw_{max}$ 和 $yw_{min} \le y_1 + u\triangle y \le yw_{max}$
- $up_k \le q_k$, k=1,2,3,4 （四个不等式统一形式）

  u (-△x) ≤x1-xwmin ,

  u (△x) ≤xwmax-x1,

  u (-△y) ≤ y1 – ywmin

  u (△y) ≤ ywmax-y1
- 若pk!=0,计算 rk=qk/pk

  $p_1=-\triangle x$ $q_1= x_1- xw_{min}$ $r_1=q_1/p_1$

  $p_2=\triangle x$ $q_2= xw_{max}- x_1$ $r_2=q_2/p_2$

  $p_3=-\triangle y$ $q_3= y_1- yw_{min}$ $r_3=q_3/p_3$

  $p_4=\triangle y$ $q_4= yw_{max}- y_1$ $r_4=q_4/p_4$

计算方法

$p_k<0$：

计算小端的r_k；小端参数 Umin=取0和各个r_k值中的最大值

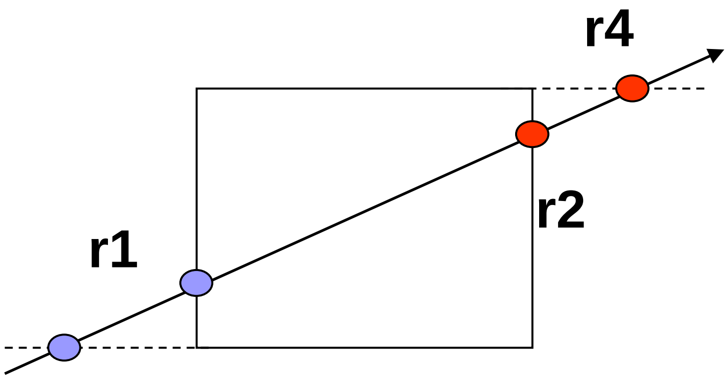
$p_k>0$,

计算大端的r_k；大端参数Umax=取1和各个r_k值中的最小值

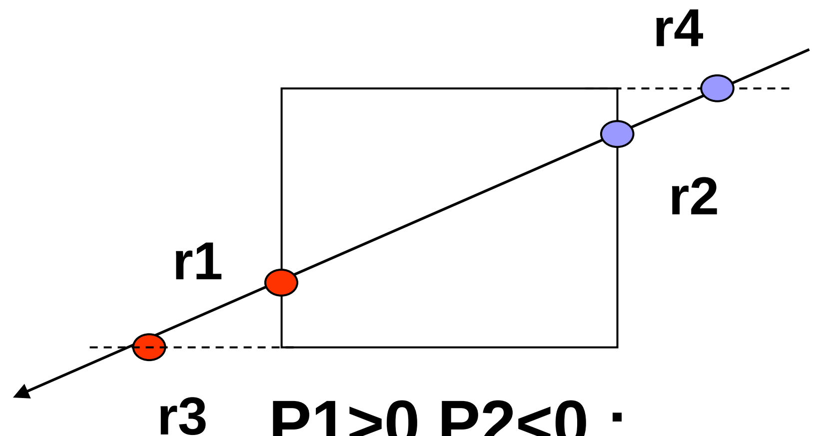
裁剪结果判定并求交：

- **Umin>=Umax**，则线段完全落在窗口之外，被舍弃；
- **Umin<Umax**，将参数带入直线参数方程，计算出裁剪结果线段的端点坐标.

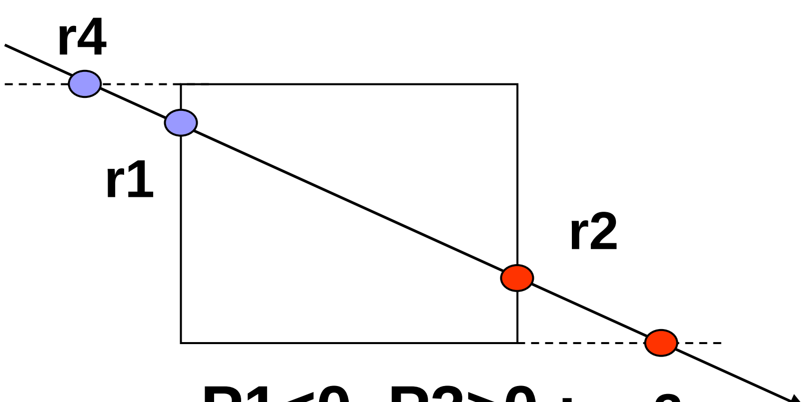
讨论：$P_k!=0$



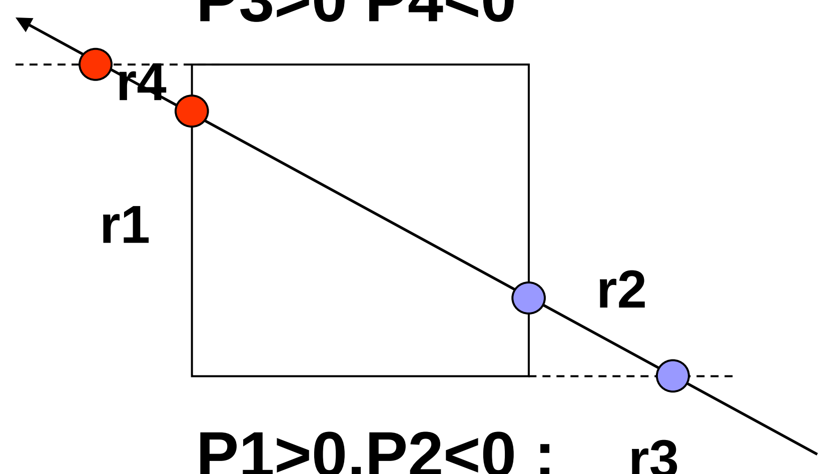
P1<0, P2>0 ;
P3<0, P4>0



P1>0 P2<0 ;
P3>0 P4<0



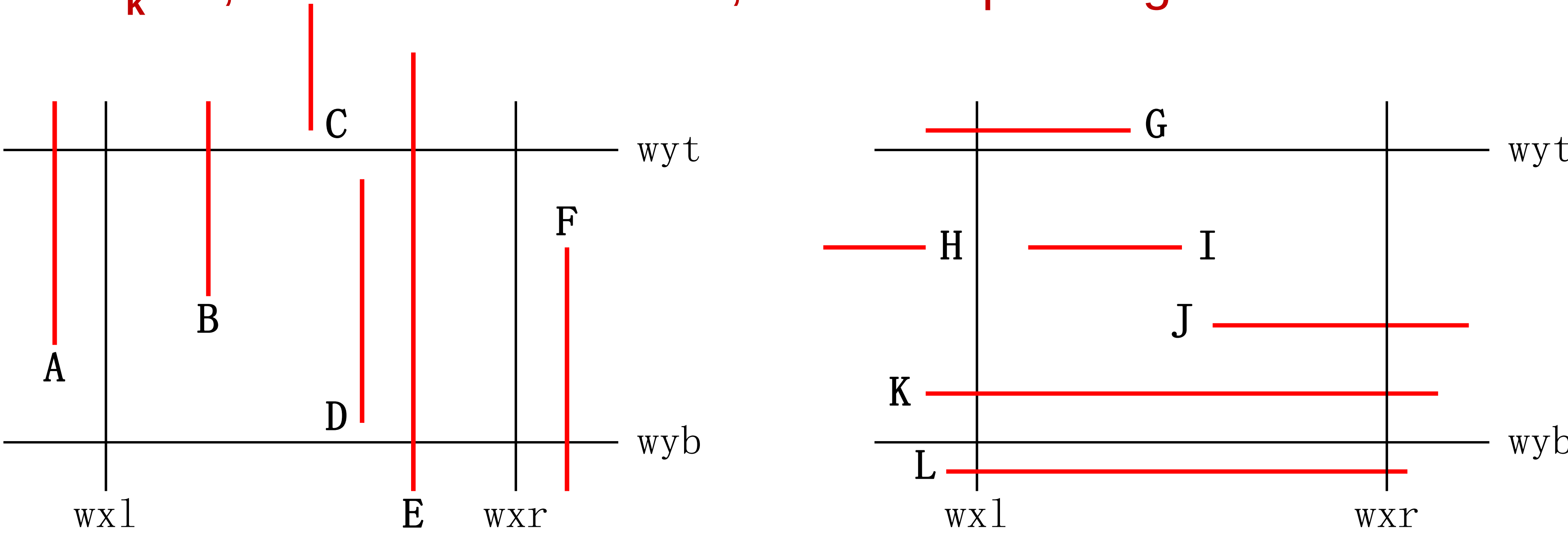
P1<0, P2>0 ;
P3>0, P4<0



P1>0,P2<0 ;
P3<0,P4>0

讨论：当p_k=0时

If P_k =0, horizontal or vertical, no corresponding r value

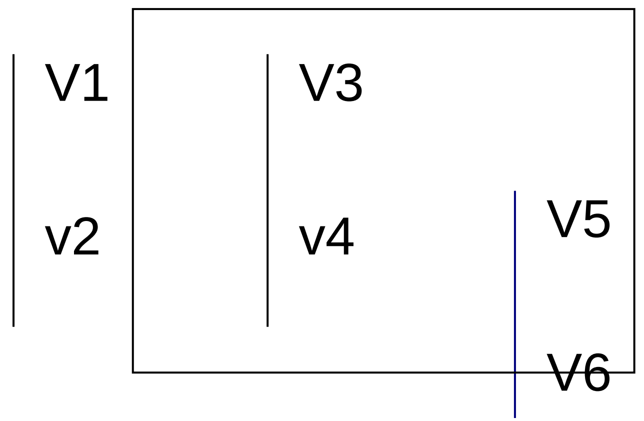


(a) 直线段与窗口边界wxl和wxr平行的情况

(b) 直线段与窗口边界wyb和wyt平行的情况

- **当pk=0时， 该线段与某组边界平行：**
  - **p1 =-△x =0, p2= △x= 0，线与Y平行**
  - **p3=-△y =0，p4=△y=0 ，线与X平行**

当p_k=0时（续）



pk=0时,如果有某个q_k<0，则该线段完全不可见（退出裁剪）

- 对于线段V_1V_2，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时$q_1=x_1- xw_{min}<0$，$q_2 =xw_{max} - x_1>0$，因此V_1V_2在窗口外。

pk=0时,如果其对应的两个q_k>0，则有裁剪结果。

- 对于直线V_3V_4，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时，$q_1>0$，$q_2>0$，则V_3V_4在窗口X方向的两裁剪边内.
- 对于直线V_5V_6，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时，$q_1>0$，$q_2>0$，则V_5V_6在窗口X方向的两裁剪边内.

完整算法描述

(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。
令Umin=0; Umax=1.

(2)若Δx=0，则$p_1=p_2=0$。若$q_1<0$或$q_2<0$，则该直线段不在窗口内，算法转(7)。否则$q_1>0$且$q_2>0$必有裁剪结果，算法转(4)。

(3)若Δy=0，则$p_3=p_4=0$。若$q_3<0$或$q_4<0$，则该直线段不在窗口内，算法转(7)。否则$q_3>0$且$q_4>0$必有裁剪结果，算法转(4)。

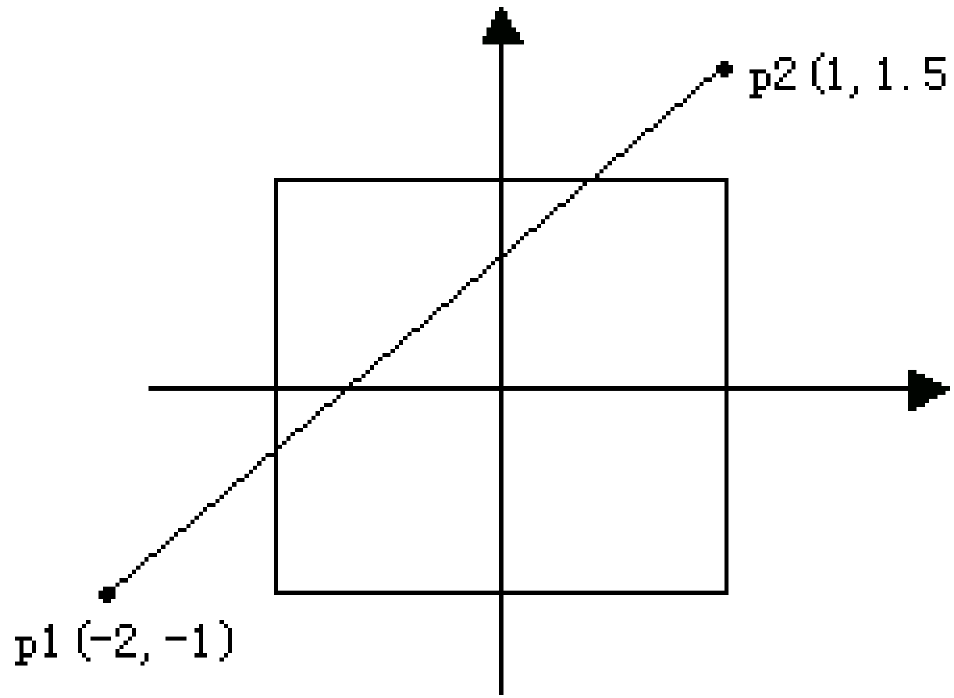
(4) 若Δx=0，则有pk≠0(k=3,4),计算r3,r4，
若Δy=0， pk≠0(k=1,2),，计算r1,r2。
若pk≠0（k=1,2,3,4），计算r1,r2,r3,r4。
将pk<0的rk与 Umin比较后取大值赋予Umin ，**将pk>0的rk与Umax比较后取小值赋予Umax。**

(5) 若Umin>Umax，则直线段在窗口外，算法转(7)。

(6)若Umin<Umax，利用直线的参数方程求得直线段在窗口内的两端点坐标，调用画线程序绘制裁剪后线段，结束。

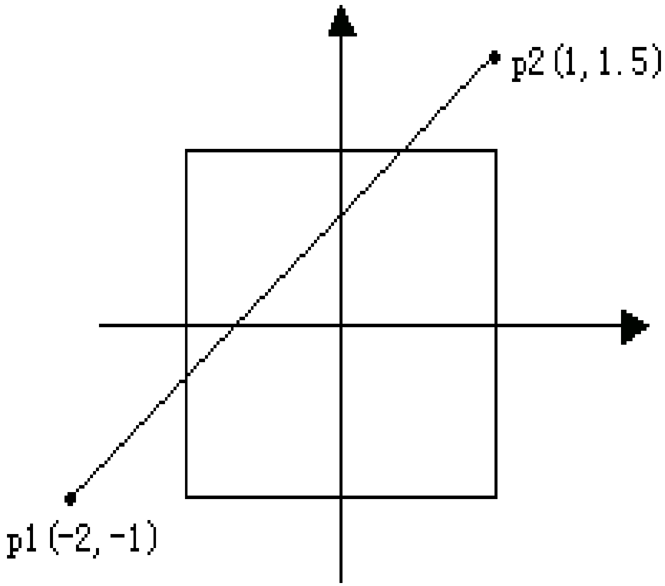
(7) 线段在窗口外，无裁剪结果，结束。

Ex.



- 窗口为(-1,-1),(1,1)
- 被裁剪线段为p1p2(-2,-1)(2,1.5)

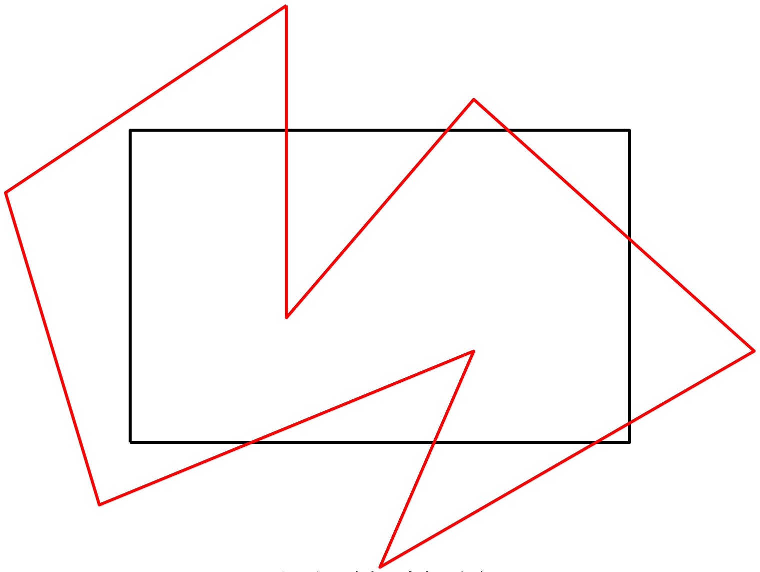
$p_1=-\triangle x \quad q_1= x_1- xw_{min} \quad r_1=q_1/p_1$
$p_2=\triangle x \quad q_2= xw_{max}- x_1 \quad r_2=q_2/p_2$
$p_3=-\triangle y \quad q_3= y_1- yw_{min} \quad r_3=q_3/p_3$
$p_4=\triangle y \quad q_4= yw_{max}- y_1 \quad r_4=q_4/p_4$



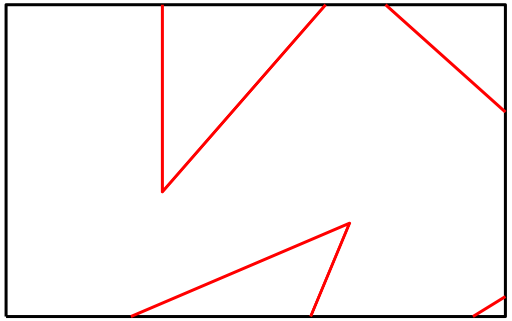
- $\triangle x=3, \triangle y=2.5$
- $p_1=-3 \quad q_1=-1;$
- $p_2=3 \quad q_2= 3$
- $p_3=-2.5 \quad q_3= 0;$
- $p_4=2.5 \quad q_4= 2$
- 对于p<0，计算小端u
  - r1=1/3, r3=0，(X1,Y1)对应的u=0,
  - 取三个数值中的最大值u_1=max(1/3,0,0)=1/3。
- 对于p>0，计算大端u
  - r2=1， r4=4/5，(X2,Y2)对应的u=1,
  - 取三个数值中的最小值u_2=min(1,4/5,1)=4/5
- 因为$u_1<u_2$，则可见线段的端点坐标可计算方法出：
  - $x=x_1+u1*\triangle x=-1, y=y_1+u1*2.5=-1/6$即(-1,-1/6)
  - $x=x_1+u2*\triangle x=2/5, y=y_1+u2*2.5=1$ 即(2/5,1)

三.多边形填充区域的裁剪

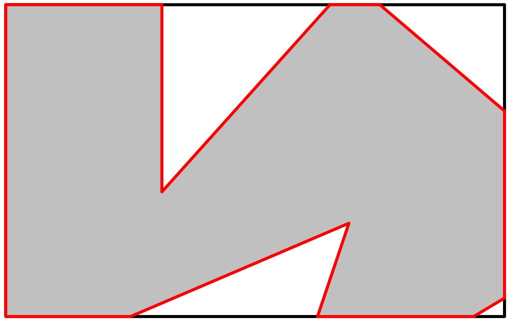
问题的提出：



(a)裁剪前



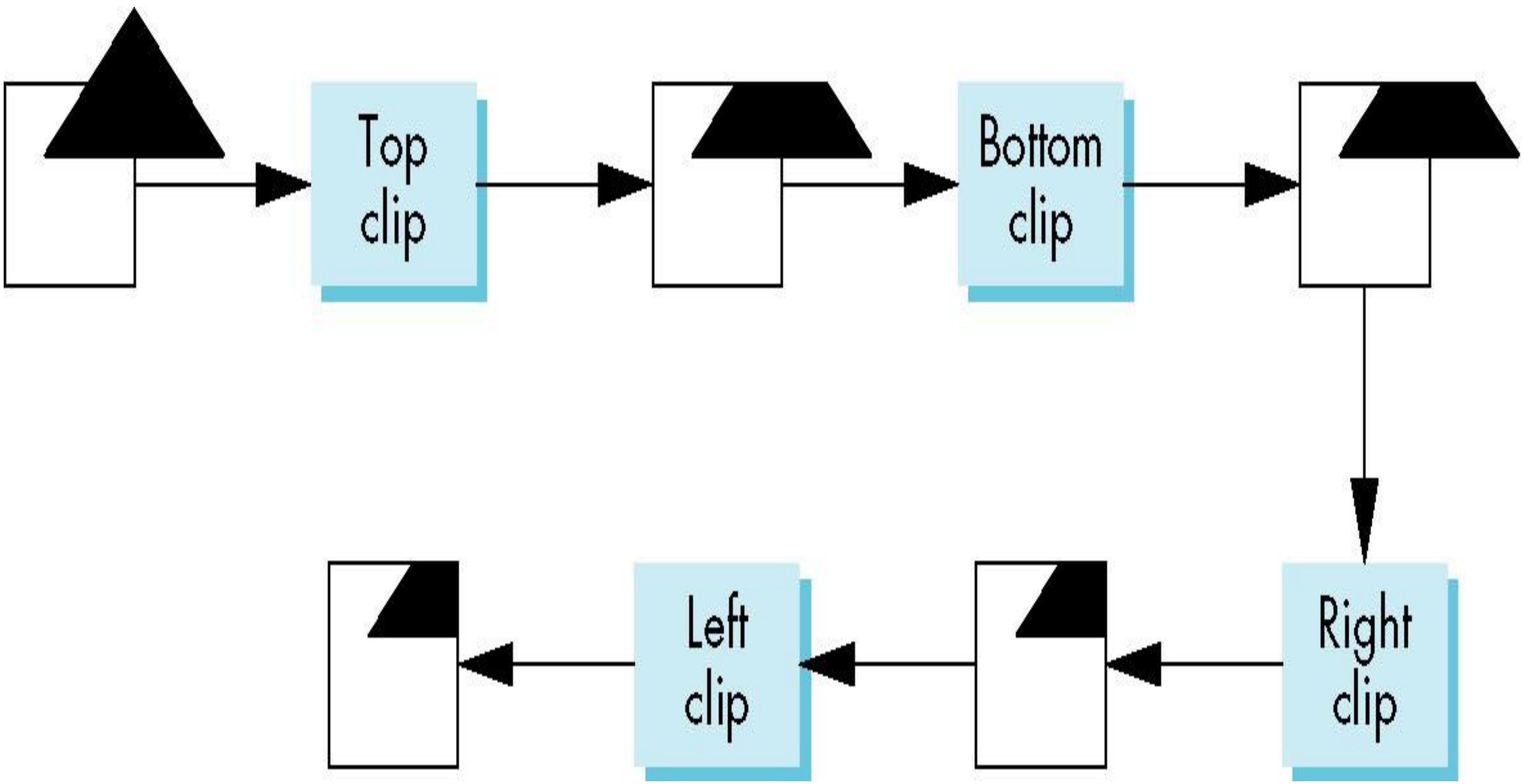
(b)直接采用直线段裁剪的结果



(c)正确的裁剪结果

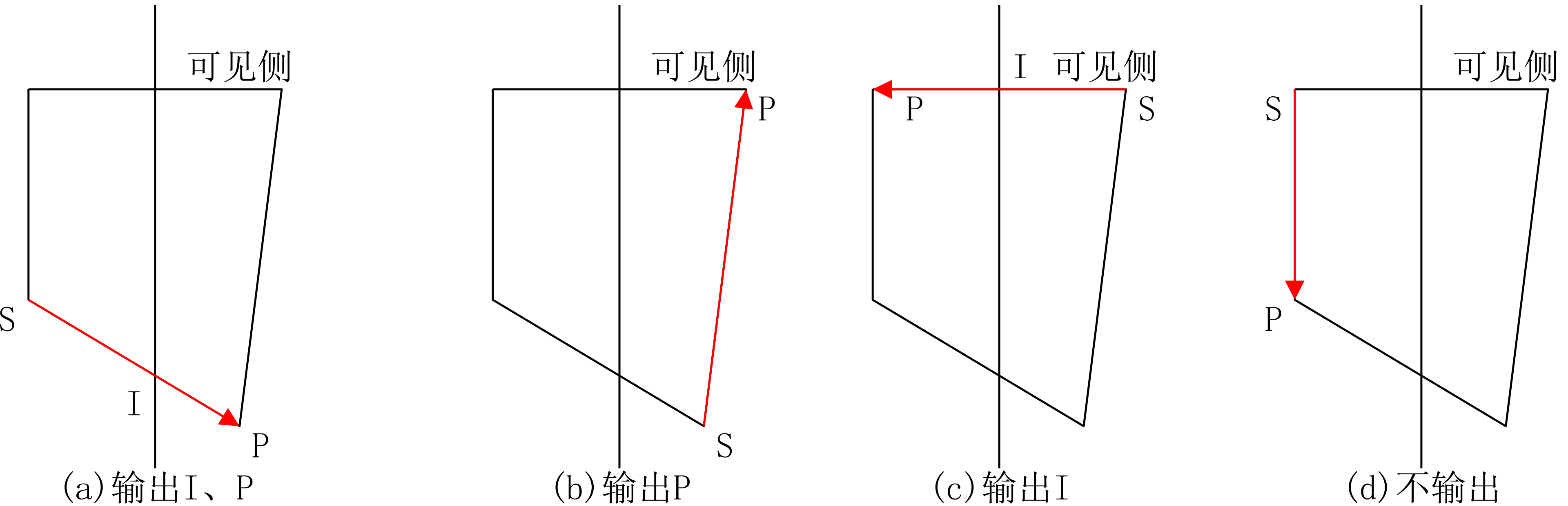
Pipeline Clipping of Polygons

多边形裁剪演示



顶点输出

对于一条裁剪边和一条多边形边，顶点处理四种情况：

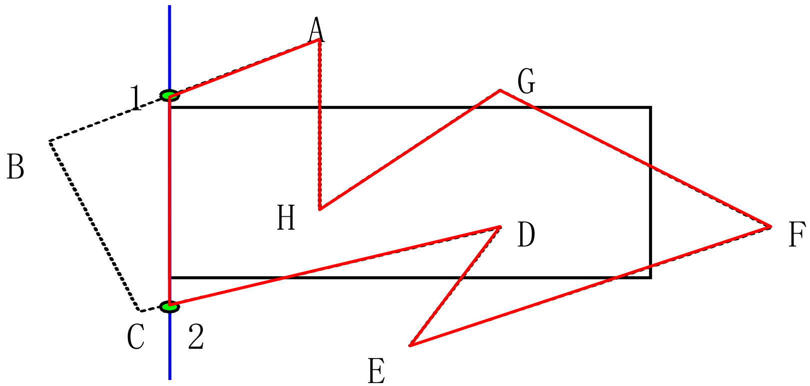


原则：**输出交点和边顶点终点**

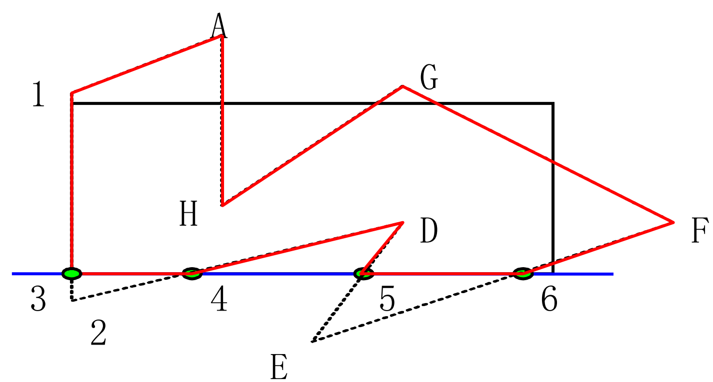
目的：避免重复输出顶点

Sutherland-Hodgeman逐边裁剪算法

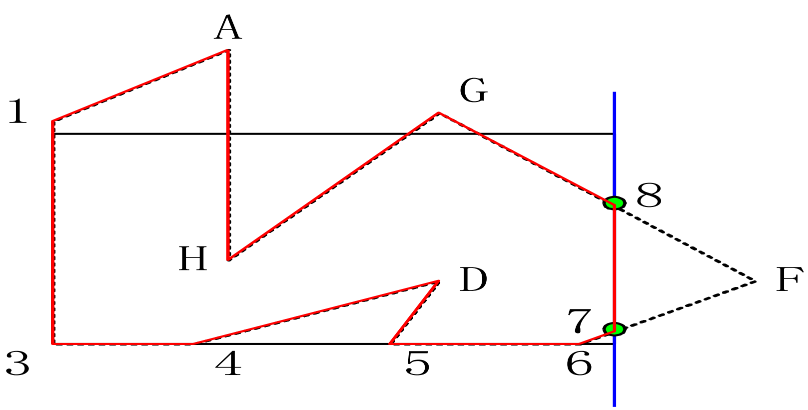
输出规律：输出交点和可见边顶点起始点



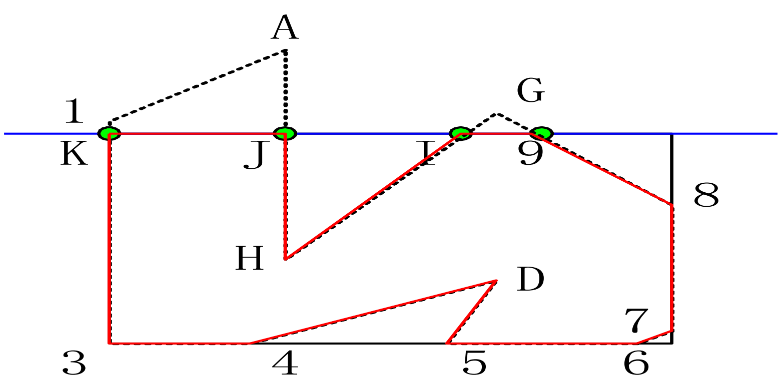
输入：ABCDEFGH
输出：A12DEFGH
(a)用左边界裁剪



输入：A12DEFGH
输出：A134D56FGH
(b)用下边界裁剪

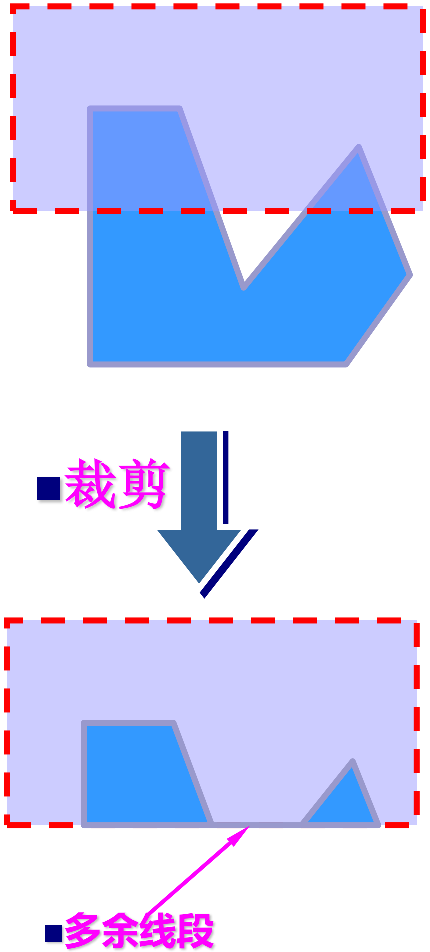


输入： A134D56FGH
输出： A134D5678GH
(c)用右边界裁剪



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

Sutherland-Hodgman算法存在的问题



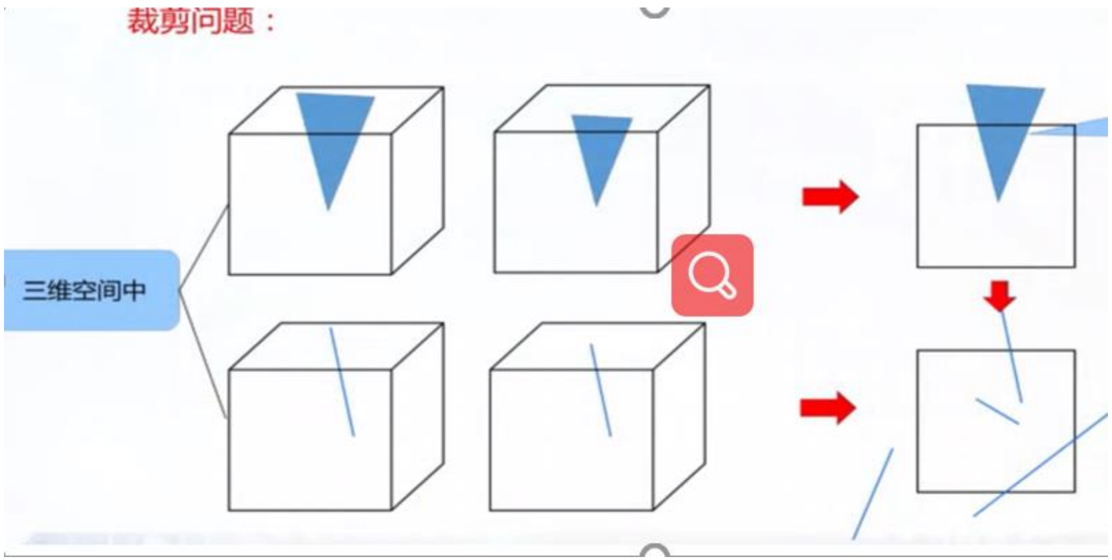
- 利用*Sutherland-Hodgeman*算法对**凸多边形**获得正确的裁剪结果。对**凹多边形**裁剪将可能出现多余的线。

◆ 正确裁剪凹多边形的方法:

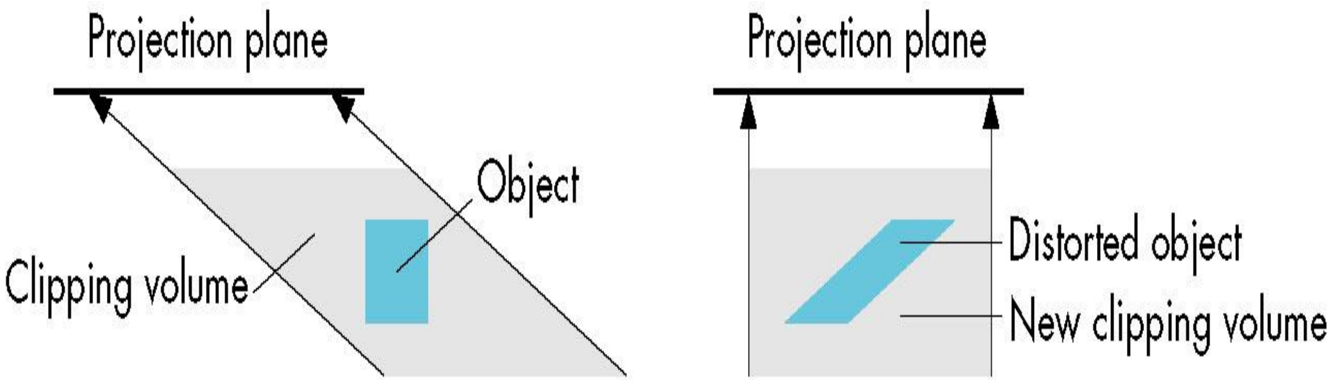
- 方法1：将凹多边形分割成两个或者更多的凸多边形，然后分别处理各个凸多边形。
- 方法2：修改Sutherland-Hodgeman算法，沿着任何一个裁剪窗口边界检查顶点表，正确地连接顶点对。
- 方法3:用更通用的多边形裁剪方法，Weiler-Atherton算法或Weiler算法。

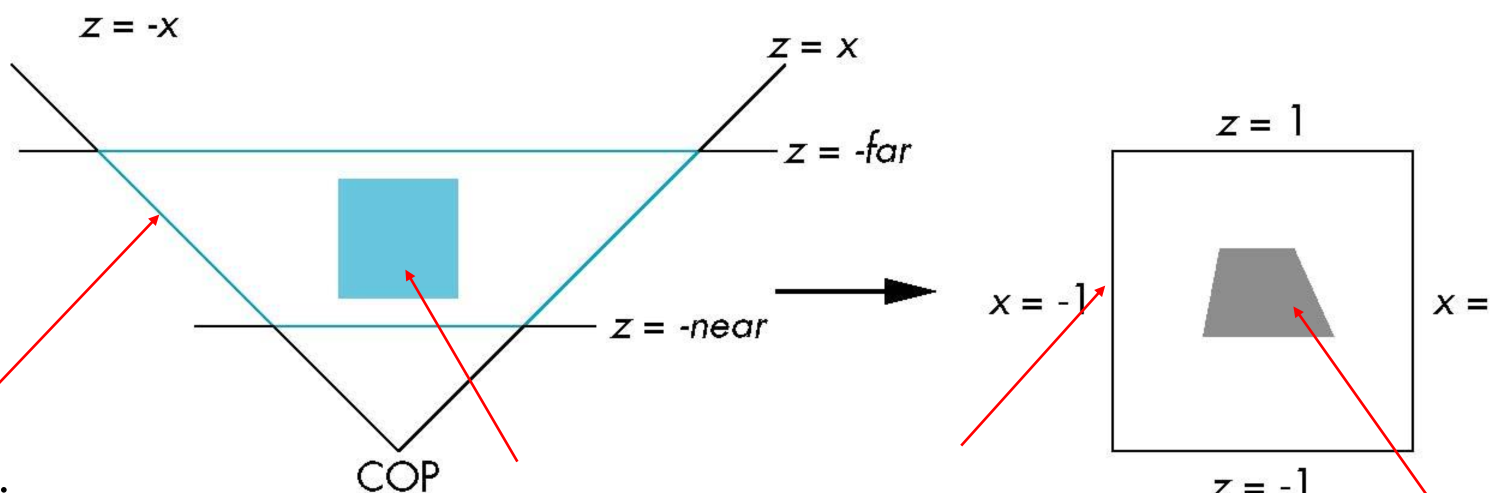
3D观察流程中的裁剪

- **观察流程中，观察窗口在什么坐标下指定？**
  - openGL是在VC中指定。
- **裁剪工作在那个坐标系下做？**
  - 通常在NC中做裁剪，裁剪边简单，效率高
  - 这时的裁剪窗口是单位化立方体边界



规范化及视见体变换之后，再对图元进行裁剪





original clipping volume

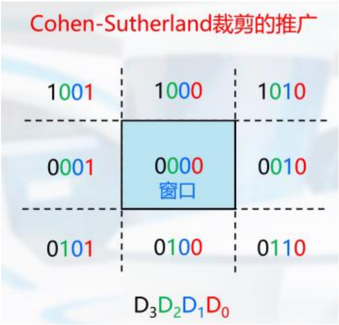
original object

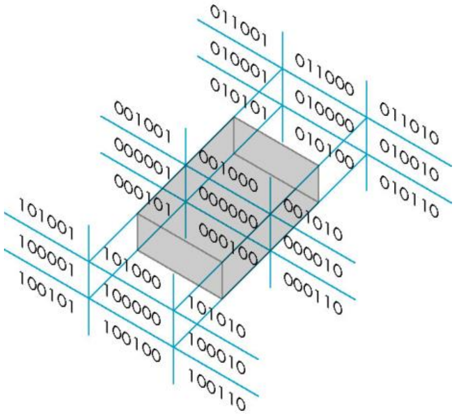
new clipping volume

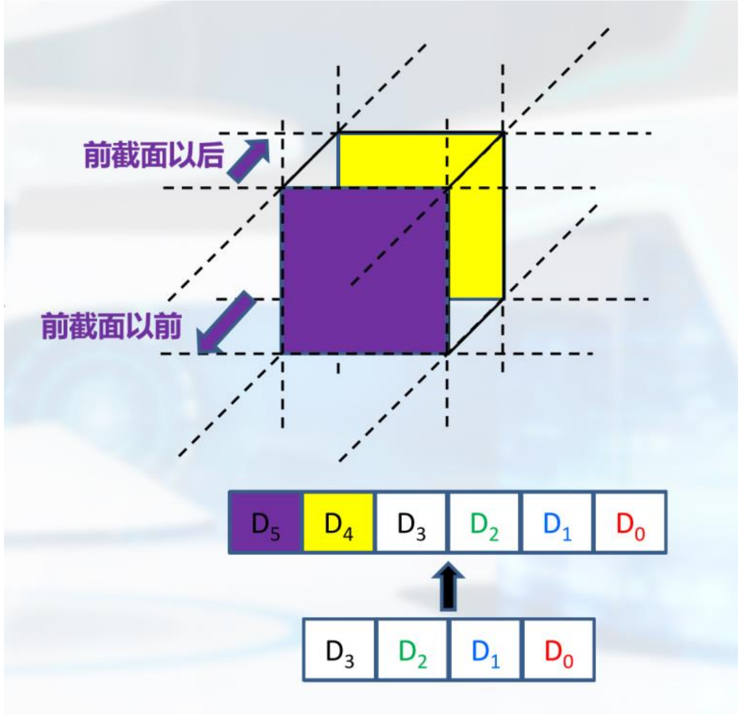
distorted object projects correctly

Cohen Sutherland in 3D

- 采用六位编码长度6-bit outcodes，27个区域
- 4个裁剪边转化为6个裁剪面

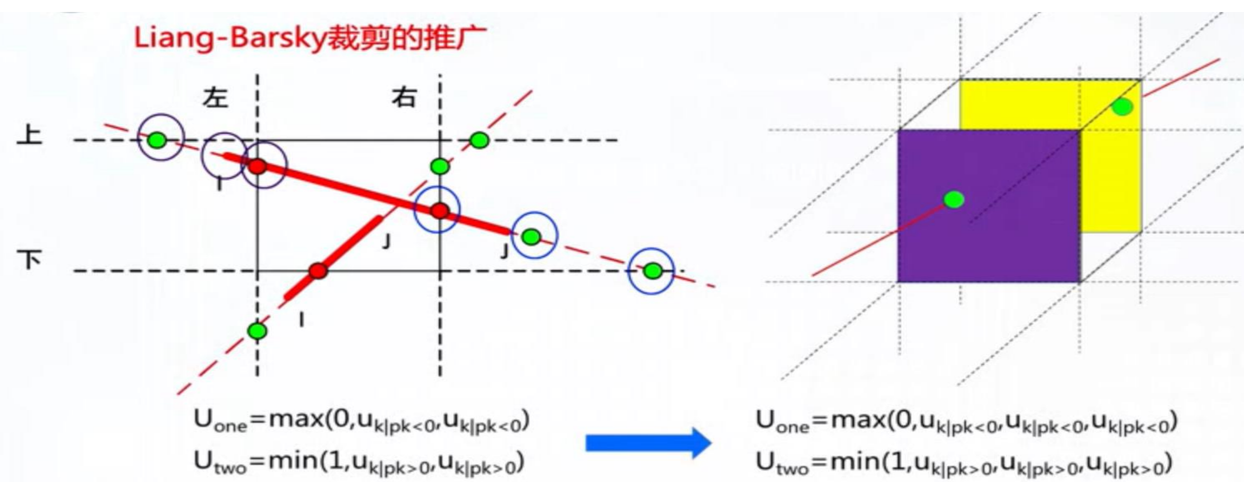






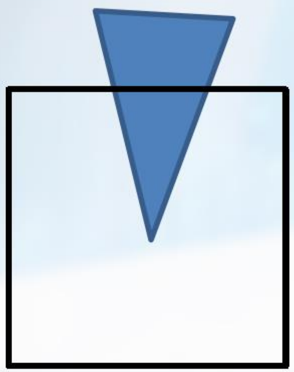
Liang-Barsky Line Clipping in 3D

- 裁剪后的线段上的点应该满足的条件是：
  - 1）在线段上
    - $0 \le u \le 1$
  - 2）在窗口内
    - $Xwmin \le X1+u\triangle X \le Xwmax$, $\triangle x=x2-x1$
    - $Ywmin \le Y1+u\triangle Y \le Ywmax$，$\triangle y=y2-y1$
    - $Zwmin \le Z1+u\triangle Z \le Zwmax$，$\triangle z=z2-z1$
- 求出参数U取值的范围，再求出端点坐标



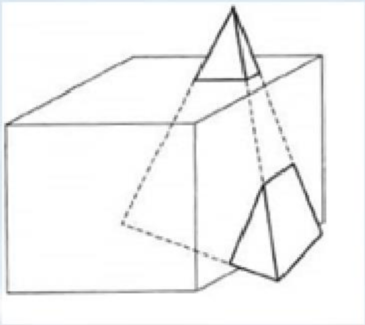
Sutherland-Hodgman in 3D

Sutherland-Hodgeman算法的推广



逐边裁剪

逐边裁剪多边形:
逐边裁剪多边形的每条边
输出：顶点序列构成多边形



逐面裁剪

逐面逐个裁剪多个多边形:
逐面裁剪多边形的每条边
输出：顶点序列，构成多面体

小结

- 直线裁剪算法
  - **Cohen-Sutherland算法**
  - **Liang-Barsky算法（*）**
- 多边形填充区域裁剪
  - **Sutherland-Hodgeman逐边裁剪算法**

练习作业

用Liang-Barsky线段裁剪算法，使用窗口（-1，-1)(1，1）裁剪以下线段：

- 线段A（-2,-2）B(2，2).
- 线段A（0,2）B(2,0).
- 线段A（0,-3）B(-3,0.)

要求给出计算步骤。