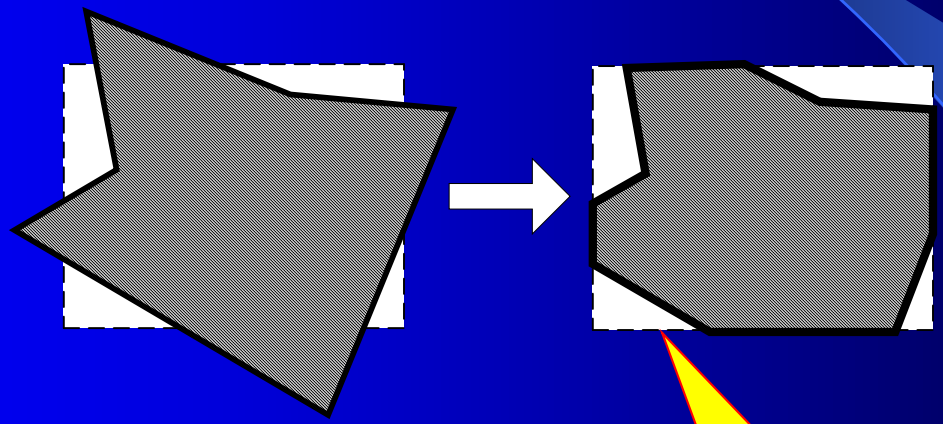
2.5 裁剪

什么叫裁剪？

裁剪：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**。



多边形裁剪实例

2.5.1 直线段裁剪

2.5.2 多边形裁剪

2.5.1 直线段裁剪

直线段裁剪算法是复杂图元裁剪的基础。复杂的曲线可以通过折线段来近似，从而裁剪问题也可以化为直线段的裁剪问题。

2.5.1.1 Cohen-Sutherland算法

2.5.1.2 中点分割算法

2.5.1.3 梁友栋－Barsky算法

2.5.1.1 Cohen-Sutherland裁剪

基本思想：对于每条线段P_1P_2分为三种情况处理：

（1）若P_1P_2在窗口内，则显示该线段P_1P_2，简称“**取**”之。

（2）若P_1P_2在窗口外，则丢弃该线段P_1P_2，简称“**弃**”之。

（3）若线段不满足“**取**”或“**弃**”的条件，则在交点处把线段分为两段。其中**一段完全在窗口外**，弃之。然后对另一段重复上述处理。

为快速判断，采用如下编码方法：

– 每个区域赋予4位编码$C_tC_bC_rC_l$

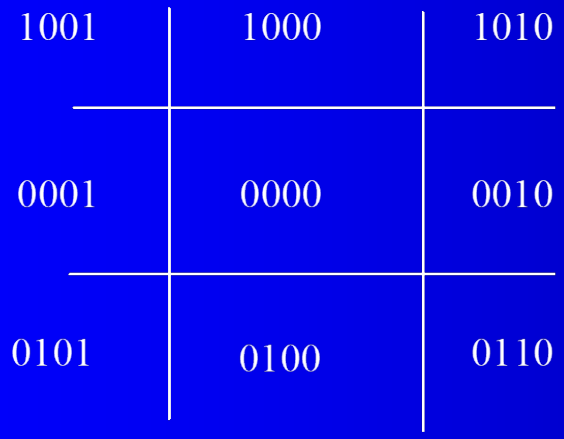
t: top上
b: bottom下
r: right右
l: left左

$$C_t=\begin{cases}1, & y>y_{\max}\\ 0, & \text{other}\end{cases}\qquad C_b=\begin{cases}1, & y<y_{\min}\\ 0, & \text{other}\end{cases}$$

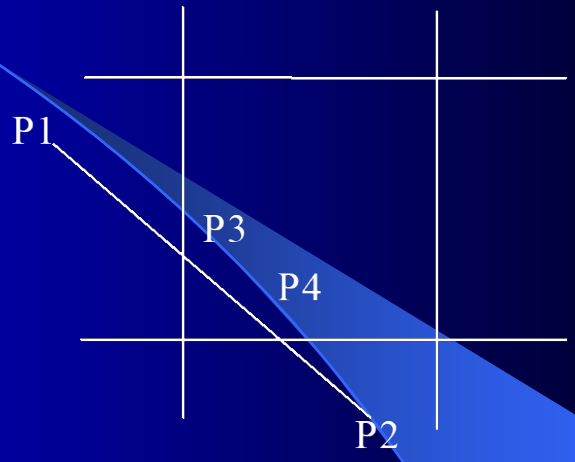
$$C_r=\begin{cases}1, & x>x_{\max}\\ 0, & \text{other}\end{cases}\qquad C_l=\begin{cases}1, & x<x_{\min}\\ 0, & \text{other}\end{cases}$$

这一编码的特点是：窗口某一条边的外侧的三个区域的编码有一位全为1。对于要被裁剪的线段，确定其两个端点的编码$code_1$和$code_2$：

- 若P_1P_2完全在窗口内$code_1 = 0$，且$code_2 = 0$，则“**取**”
- 若P_1P_2完全在窗口外$code_1 \,\&\, code_2 \neq 0$，则“**弃**”
- **在交点处把线段分为两段**。其中一段完全在窗口外，可弃之。然后对另一段重复上述处理。



编码

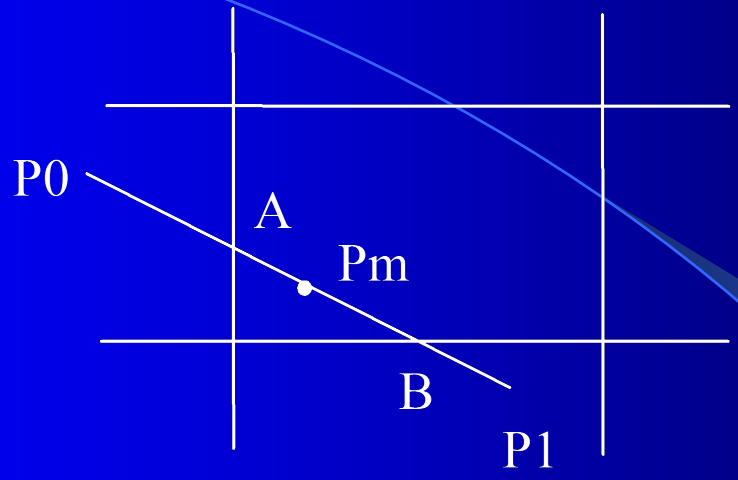


线段裁剪

2.5.1.2 中点分割裁剪算法

基本思想：与前一种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和从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。



A、B分别为距P_0、P_1最近的可见点，P_m为P_0P_1中点

求线段与窗口的交点

从P_0出发找最近可见点采用中点分割方法

- 先求出P_0P_1的中点P_m
- 若P_0P_m不是完全不可见的，则距P_0最近的可见点一定落在P_0P_m上，所以用P_0P_m代替P_0P_1
- 否则取P_mP_1代替P_0P_1
- 再对新的P_0P_1求中点P_m。重复上述过程，直到P_mP_1长度小于给定的控制常数为止，此时P_m收敛于交点

从P_1出发找最近可见点采用与上面类似方法。

2.5.1.3 梁友栋－Barsky算法

梁友栋和Barsky提出了更快的**参数化裁剪算法**。过线段P_1P_2的直线的参数方程为:

$$P = P_1 + u(P_2 - P_1), \qquad (0 \le u \le 1)$$

即:

$$x = x_1 + u\Delta x \qquad (\Delta x = x_2 - x_1)$$
$$y = y_1 + u\Delta y \qquad (\Delta y = y_2 - y_1)$$

参数化形式写出裁剪条件：

$$XL \le x_1 + u\Delta x \le XR$$
$$YB \le y_1 + u\Delta y \le YT$$

What is u?

可以统一表示为形式：　$up_k \le q_k$

k=1：Left左

$p_1 = -\Delta x$　　$q_1 = x_1 - XL$

$p_3 = -\Delta y$　　$q_3 = y_1 - YB$

k=3：Bottom下

k=2：Right右

$p_2 = \Delta x$　　$q_2 = XR - x_1$

$p_4 = \Delta y$　　$q_4 = YT - y_1$

k=4：Top上

即 Δx或 Δy为0--垂线或水平线

内部:包含裁剪窗口的**半平面**

– 当$p_k = 0$且$q_k < 0$，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**在内部**

Line1

$p_3 = 0 \quad q_3 = y_1 - YB > 0$

Line inside

Line3

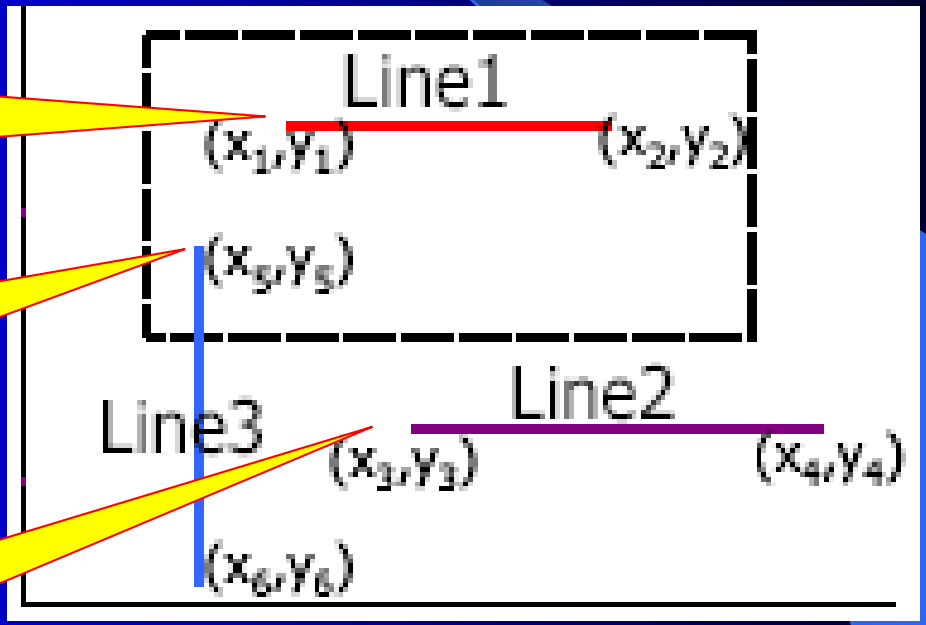
$p_1 = 0 \quad q_1 = x_5 - XL > 0$

Line inside

Line2

$p_3 = 0 \quad q_3 = y_3 - YB < 0$

Line outside, Reject!!



直线方向：参数u增加的方向，即$P_1 \rightarrow P_2$

内部:包含裁剪窗口的**半平面**

– 当$p_k<0$, 线段从**裁剪边界延长线**的**外部**延伸到**内部**。

Line2 k=4 (Top)

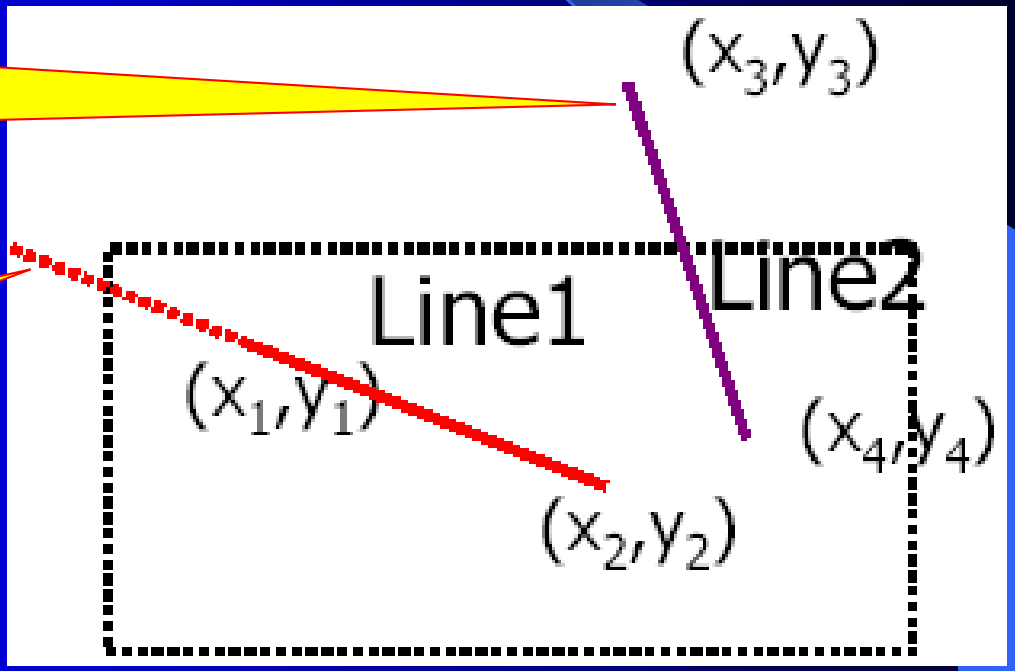
$p_4 = \triangle y = y_4 - y_3 < 0$

$\Rightarrow y_4 < y_3$

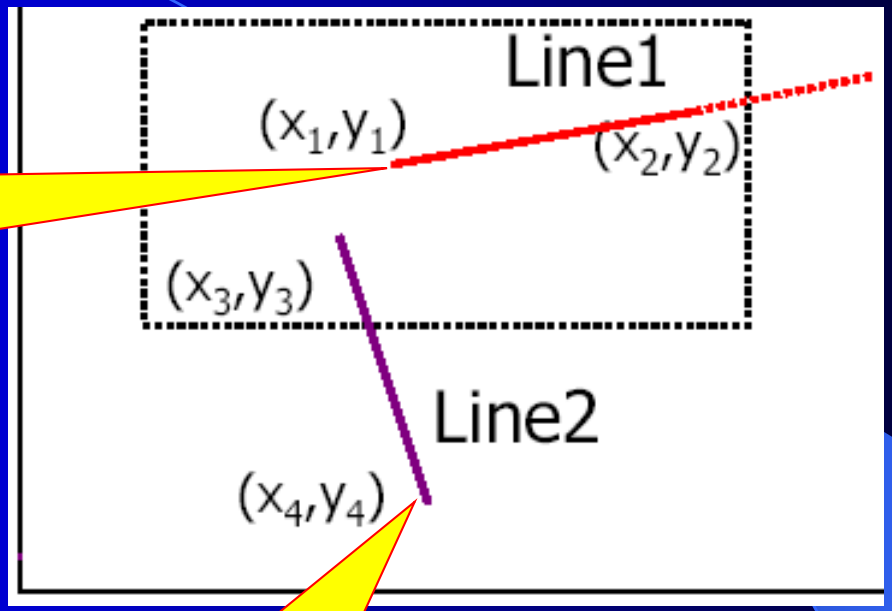
Line1 k=1 (Left)

$p_1 = -\triangle x = -(x_2 - x_1) < 0$

$\Rightarrow x_1 < x_2$



– 当$p_k > 0$, 线段从**裁剪边界延长线**的内部延伸到外部。



Line1 k=2 (Right)

$p_2 = \Delta x = x_2 - x_1 > 0$

$\longrightarrow x_2 > x_1$

Line2 k=3 (Bottom)

$p_3 = -\Delta y = -(y_4 - y_3) > 0$

$\longrightarrow y_4 < y_3$

For each k, calculate $r_k=q_k/p_k$

$u_1=\max(0, r_k)$ for all k where $p_k<0$
[outside to inside]

$u_2=\min(1, r_k)$ for all k where $p_k>0$
[inside to outside]

即边界

If (p_k=0) and (q_k=0) reject line

If $u_1 > u_2$ then discard line,
else use u_1 and u_2 to calculate end points of the line

$\triangle x = 13-3 = 10 \quad \triangle y = 8-1 = 7$

Left (k =1)

$p_1 = -\triangle x = -10 \quad q_1 = x_1 - XL = 3-4 = -1$

$r_1 = -1/-10 = 1/10$

$u_1 = \max(0, 1/10) = 1/10$

Right (k =2)

$p_2 = \triangle x = 10 \quad q_2 = XR - x_1 = 11-3 = 8$

$r_2 = 8/10$

$u_2 = \min(1, 8/10) = 8/10 = 0.8$

Bottom(k =3)

$p_3 = -\triangle y = -7 \quad q_3 = y_1 - YB = 1-2 = -1$

$r_3 = -1/-7 = 1/7$

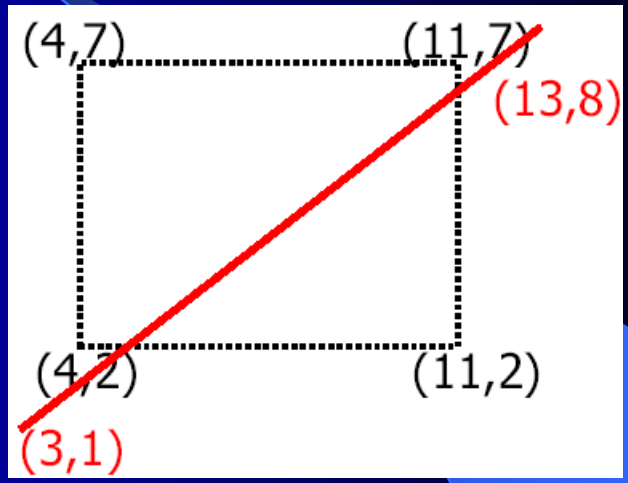
$u_1 = \max(1/10, 1/7) = 1/7$

Top(k =4)

$p_4 = \triangle y = 7 \quad q_4 = YT - y_1 = 7-1 = 6$

$r_4 = 6/7 = 0.85$

$u_2 = \min(0.8, 0.85) = 0.8$



u_1=1/7 u_2 = 0.8

$$x_1^{'} = x_1 + u_1 \Delta x$$

= 3+(1/7)*10=3+1.42=4.42

$$y_1^{'} = y_1 + u_1 \Delta y$$

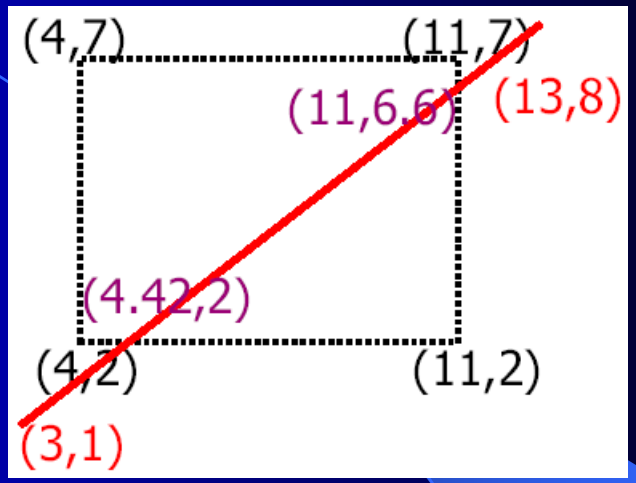
= 1+(1/7)* 7 = 2

$$x_2^{'} = x_1 + u_2 \Delta x$$

= 3+(8/10)*10=3+8=11

$$y_2^{'} = y_1 + u_2 \Delta y$$

= 1+(8/10)* 7 = 1+5.6=6.6



```
void LB_LineClip(float x1, float y1, float x2, float y2,
                 float XL, float XR, float YB, float 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(float p, float q, float *u1, float *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=q/p;
                if(r<*u1)  return false;
                else if(r<*u2)
                        { *u2=r; return true;}
              }
             else if(q<0) return false;
      return true;
}
```