



# 裁剪

[ray@mail.buct.edu.cn](mailto:ray@mail.buct.edu.cn)

2017/10/18



# 内容

- 直线段裁剪
- 多边形裁剪
- 字符裁剪



# 直线段裁剪

## ■ 裁剪

- 裁剪(clipping)是裁去窗口之外物体的一种操作。
- 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。



# Cohen-Sutherland裁剪

## ■ 基本思想

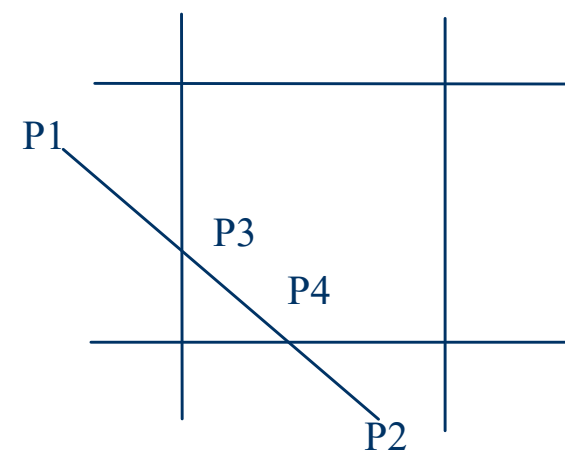
- 对于每条线段 $P_1P_2$ 分为三种情况处理分为三种情况处理:
  1. 若 $P_1P_2$ 完全在窗口内, 则显示该线段 $P_1P_2$ , 简称“取”。
  2. 若 $P_1P_2$ 明显在窗口外, 则丢弃该线段, 简称“弃”。
  3. 若线段不满足“取”或“弃”的条件, 则在交点处把线段分为两段。其中一段完全在窗口外, 可弃之。然后对另一段重复上述处理。

- 用编码方法判断以上三种情况
  - 采用4位编码

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

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

1001	1000	1010
0001	0000	0010
0101	0100	0110



1.  $\text{code1}=0$ 且 $\text{code2}=0$

- $P_1P_2$ 完全在窗口内
- “取”

2.  $\text{code1}\&\text{code2}\neq 0$

- $P_1P_2$ 明显在窗口外
- “弃”

3. 既不是1又不是2的其它情况

- 在交点处把线段分为两段，其中一段完全在窗口外，可弃之，另一段重复上述处理。

## ■ 计算线段与窗口边界交点的具体算法

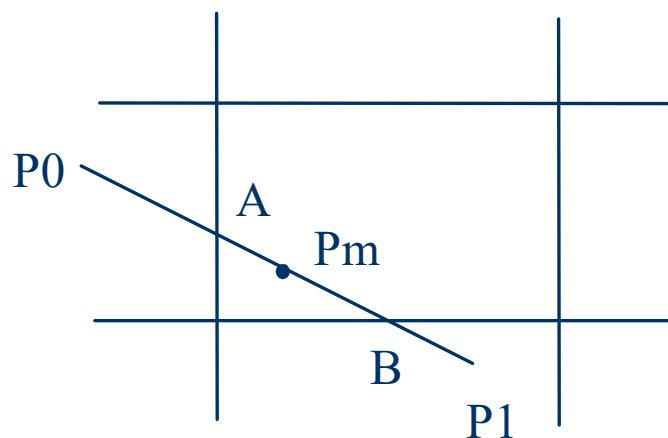
```
#define LEFT 1
#define RIGHT 2
#define BOTTOM 4
#define TOP 8
.....
if(LEFT & code != 0)
    {x=XL;y=y1+(y2-y1)*(XL-x1)/(x2-x1);}
else if(RIGHT & code != 0)
    {x=XR;y=y1+(y2-y1)*(XR-x1)/(x2-x1);}
else if(BOTTOM & code != 0)
    {y=YB;x=x1+(x2-x1)*(YB-y1)/(y2-y1);}
else if(TOP & code != 0)
    {y=YT;x=x1+(x2-x1)*(YT-y1)/(y2-y1);}
.....
```





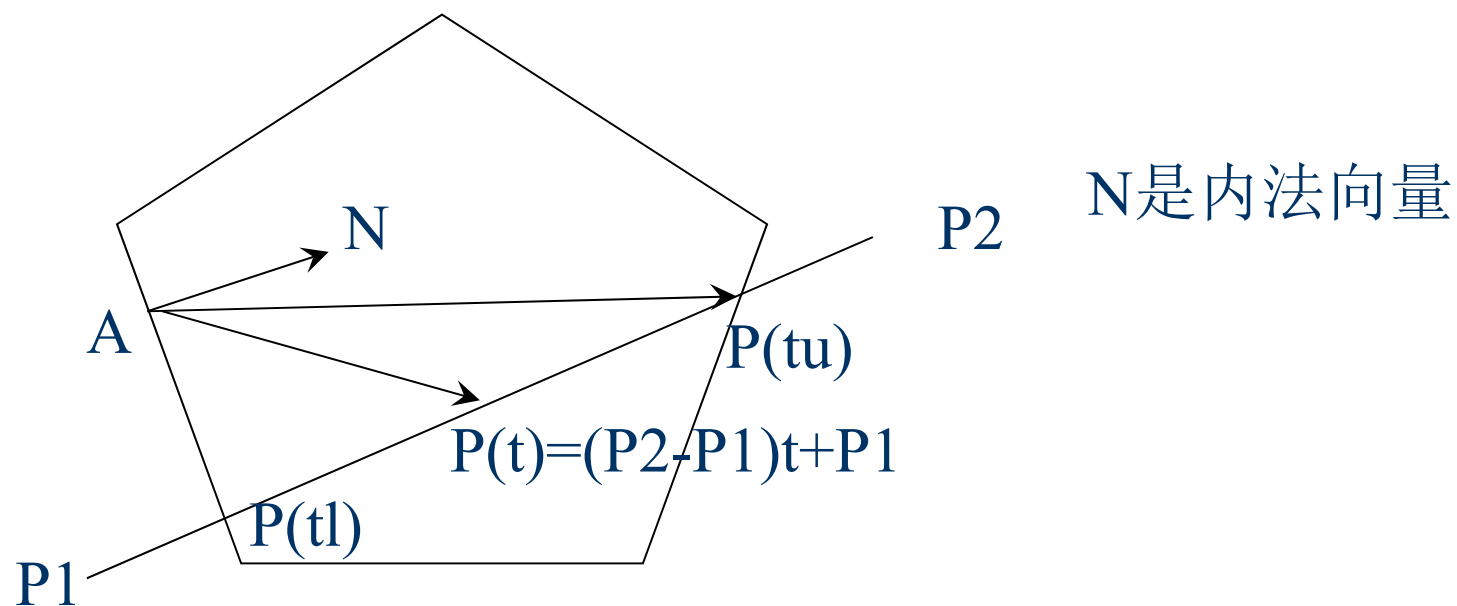
# 中点分割法求线段与窗口的交点

A、B分别为距 $P_0$ 、 $P_1$ 最近的可见点， $P_m$ 为 $P_0P_1$ 中点



- 从 $P_0$ 出发找最近可见点的方法
  1. 先求出 $P_0P_1$ 的中点 $P_m$
  2. 若 $P_0P_m$ 不是显然不可见的，并且 $P_0P_m$ 在窗口中有可见部分，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 $P_0P_1$ ；
  3. 否则取 $P_mP_1$ 代替 $P_0P_1$ ；
  4. 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上面类似方法

# Cyrus-Beck算法

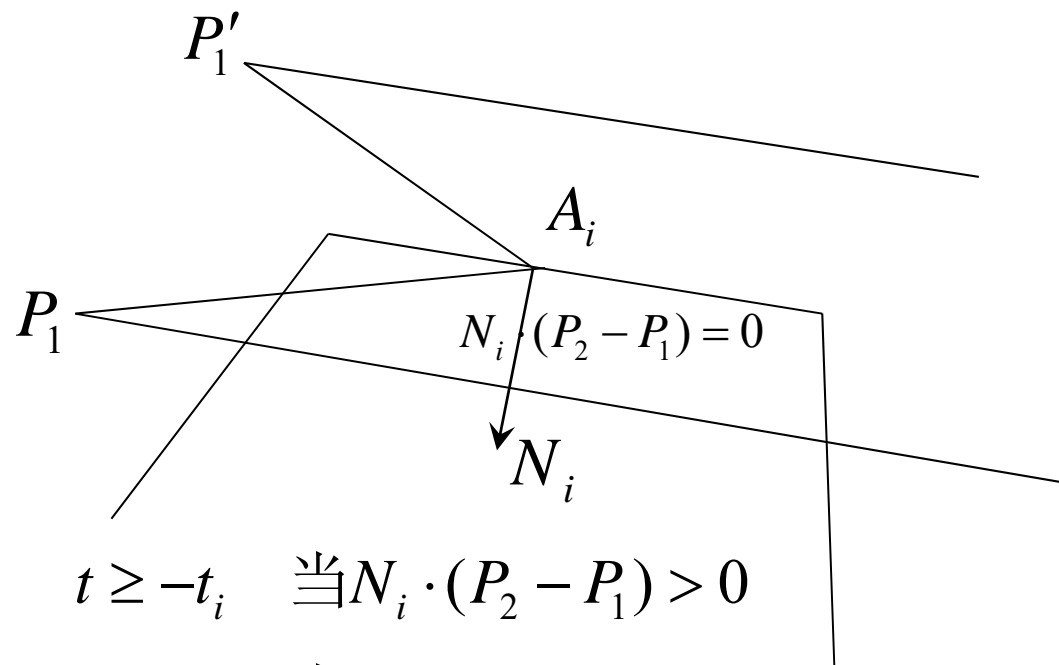


- (1)  $N(P(t) - A) > 0$ ,  $P(t)$  在 多边形内侧
- (2)  $N(P(t) - A) = 0$ ,  $P(t)$  在 多边形边线
- (3)  $N(P(t) - A) < 0$ ,  $P(t)$  在 多边形外侧

$$P(t) = (P_2 - P_1)t + P_1$$

$$\begin{cases} N_i \cdot (P(t) - A_i) \geq 0 (i = 1, 2, \dots, k) \\ 0 \leq t \leq 1 \end{cases}$$

$$\begin{cases} N_i \cdot (P_1 - A_i) + N_i \cdot (P_2 - P_1)t \geq 0 \\ 0 \leq t \leq 1 \end{cases}$$



$$t \geq -t_i \quad \text{当 } N_i \cdot (P_2 - P_1) > 0$$

$$t \leq -t_i \quad \text{当 } N_i \cdot (P_2 - P_1) < 0$$

$$tl = \max \{0, \max \{-t_i : N_i \cdot (P_2 - P_1) > 0\}\}$$

$$tu = \min \{1, \min \{-t_i : N_i \cdot (P_2 - P_1) < 0\}\}$$



# 梁友栋算法

## ■ 算法简介

- 梁友栋算法是Cyrus-Beck算法的特例
- 当凸多边形是矩形窗口时，Cyrus-Beck算法便退化为梁友栋算法

边	内法向量	边上点A	P1-A	$t = -[N(P1-A)]/[N(P2-P1)]$
X=XL	(1,0)	(XL,Y)	(X1-XL,Y1-Y)	$-(X1-XL)/(X2-X1)$
X=XR	(-1,0)	(XR,Y)	(X1-XR,Y1-Y)	$-(X1-XR)/(X2-X1)$
Y=YB	(0,1)	(X,YB)	(X1-X,Y1-YB)	$-(Y1-YB)/(Y2-Y1)$
Y=YT	(0,-1)	(X,YT)	(X1-X,Y1-YT)	$-(Y1-YT)/(Y2-Y1)$



# 程序实现

```
void LB_LineClip(x1,y1,x2,y2,XL,XR,YB,YT)
float x1,y1,x2,y2,XL,XR,YB,YT;
{
    float dx,dy,u1,u2;
    tl=0;tu=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(p,q,u1,u2)
float p,q,*u1,*u2;
{
    float r;
    if(p<0)
    {
        r=q/p;
        if(r>*u2)return FALSE;
        else if(r>*u1)
        {
            *u1=r;
            return TRUE;
        }
    }
}
```

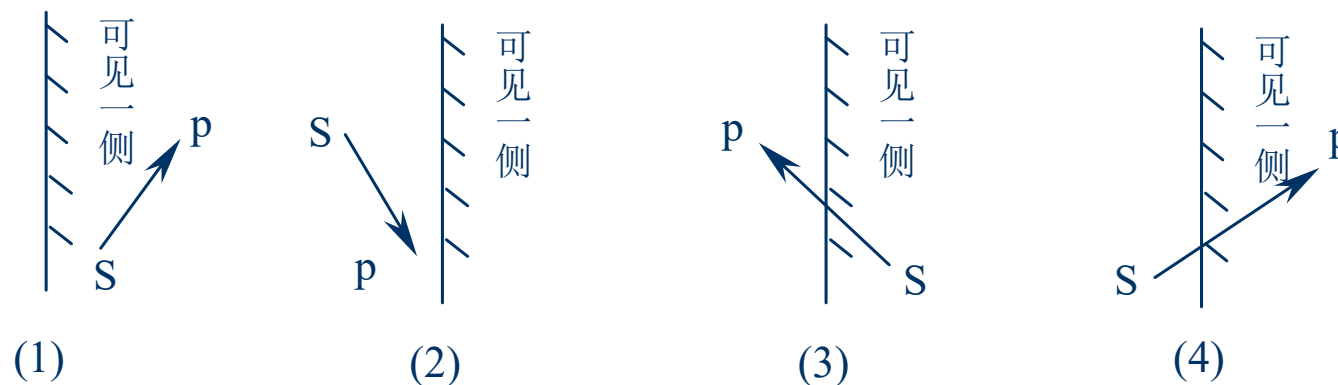


```
    else if(p>0)
    {
        r=p/q;
        if(r<*u1)return FALSE;
        else if(r<*u2)
        {
            *u2=r;
            return TRUE;
        }
    }
    else if(q<0) return FALSE;
    return TRUE;
}
```

# 多边形裁剪

## ■ 基本思想

- 一次用窗口的一条边裁剪多边形。
- 考虑窗口的一条边以及延长线构成的裁剪线该线把平面分成两个部分:可见一侧; 不可见一侧。
- 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:



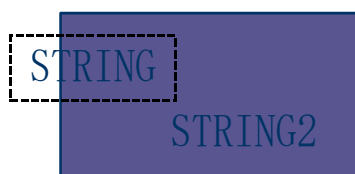
- 情况（1）仅输出顶点P；
- 情况（2）输出0个顶点；
- 情况（3）输出线段SP与裁剪线的交点I；
- 情况（4）输出线段SP与裁剪线的交点I和终点P.



# 字符裁剪

## ■ 字符裁剪分类

- 串精度：将包围字符串的外接矩形对窗口作裁剪。
- 字符精度：将包围字的外接矩形对窗口作裁剪。
- 象素精度：将笔划分解成直线段对窗口作裁剪。



待裁剪字符串



串精度裁剪



字符精度裁剪



象素精度裁剪



# 三维裁剪

- 编码方法
  - Cohen-Sutherland算法
  - 中点分割算法
- 参数化裁剪算法
  - Cyrus-Beck算法
    - 二维向量换成三维向量
    - 边法向量换成面法向量