第五章　图形运算

- 第一节　线段的交点计算
- 第二节　多边形表面的交线计算
- 第三节　平面中的凸壳算法
- 第四节　包含与重叠
- 第五节　简单多边形的三角剖分

第一节 线段的交点计算

一、两条线段求交

设有两线段AB和CD,其端点坐标分别为$(x_a,y_a),(x_b,y_b)$和$(x_c,y_c),(x_d,y_d)$,它们所在直线的参数方程分别为:

$$\begin{cases} x = x_a + \lambda(x_b - x_a) \\ y = y_a + \lambda(y_b - y_a) \end{cases} \quad \begin{cases} x = x_c + \mu(x_d - x_c) \\ y = y_c + \mu(y_d - y_c) \end{cases}$$

若两线段相交,则交点的参数值,应满足:

$$\begin{cases} x = x_a + \lambda(x_b - x_a) = x_c + \mu(x_d - x_c) \\ y = y_a + \lambda(y_b - y_a) = y_c + \mu(y_d - y_c) \end{cases}$$

即

$$\begin{cases} (x_b - x_a)\lambda - (x_d - x_c)\mu = x_c - x_a \\ (y_b - y_a)\lambda - (y_d - y_c)\mu = y_c - y_a \end{cases}$$

因此,若行列式 $\Delta = \begin{vmatrix} x_b - x_a & -(x_d - x_c) \\ y_b - y_a & -(y_d - y_c) \end{vmatrix} = 0$

表示两线段AB和CD重合或平行。一般做为它们不相交来处理。如果$\Delta \neq 0$,则可求出交点对应的两个参数值:

$$\begin{cases} \lambda = \dfrac{1}{\Delta}\begin{vmatrix} x_c - x_a & -(x_d - x_c) \\ y_c - y_a & -(y_d - y_c) \end{vmatrix} \\ \mu = \dfrac{1}{\Delta}\begin{vmatrix} x_b - x_a & (x_c - x_a) \\ y_b - y_a & (y_c - y_a) \end{vmatrix} \end{cases}$$

需要注意，只有$0 \le \lambda \le 1$，$0 \le \mu \le 1$时两线段才真正相交。否则，交点在两线段或其中某一条线段的延长线上，这时仍然认为是两线段不相交。

两线段AB和CD交点的算法

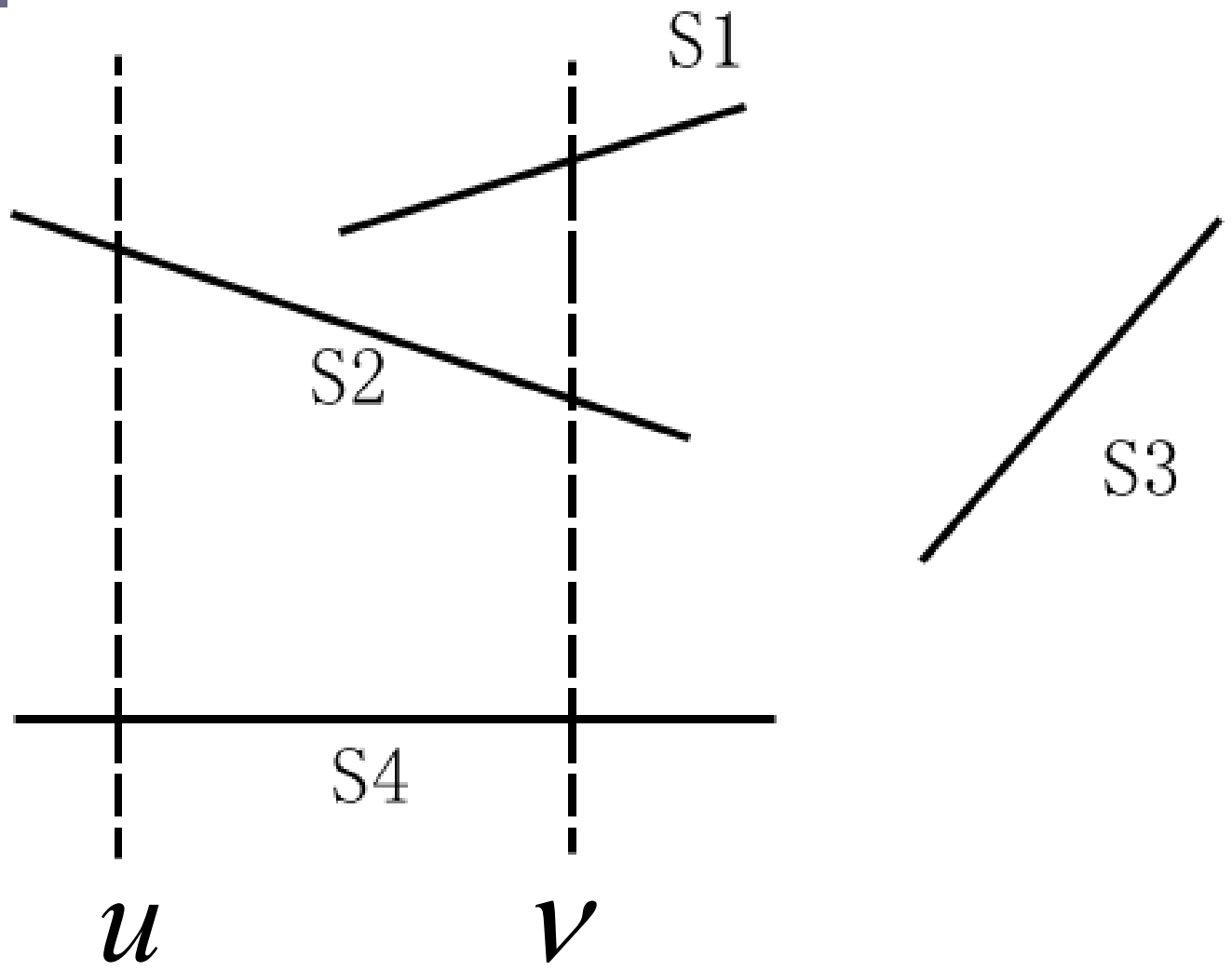
```
int  LineSegmentIntersection (POINT a, POINT b, POINT c, POIN
/* POINT 平面点类型,
 a,b,c,d 两直线的端点, p求解的交点*/
{
      float t,t1,t2;
      t=(b.x-a.x))*(c.y-d.y)-(c.x-d.x))*(b.y-a.y);   //计算行列式
      if( fabs(t)<=1.0e-3) return 0;  //平行或重合，无解返回
      t1=((c.x-a.x))*(c.y-d.y)-(c.x-d.x))*(c.y-a.y)/t;   //计算参数
      if (t1<0 ||t1>1 ) return 0 ;   //无解返回
      t2=((b.x-a.x))*(c.y-a.y)- (c.x-a.x))*(b.y-a.y)/t;  //计算参数
      if (t2<0 ||t2>1 )  return  0; //无解返回
      p.x=a.x+t1*(b.x-a.x);
      p.y=a.y+t1*(b.y-a.y);
      return 1;
}
```

二.多条线段求交

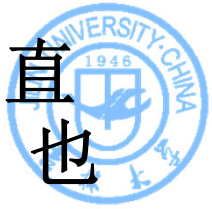
① 穷举法(很多线段不相交时，效率低，计算浪费）
②扫描线思想求交（高效算法只对有可能相交的两线段计算交点,对不可能相交的线段不计算交点）

定义关系
线段之间是可比较的（两条线段与同一垂线有交点）

x处的“上面”关系为：
在x处,线段S_1在S_2的上面,记为$S_1 >_x S_2$,如果在x处可比较,且S_1与垂直线的交点位于S_2与垂直线的交点的上面。



其中，$S_2>_uS_4, S_1>_vS_2, S_2>_vS_4, S_1>_vS_4$



假设：为简便起见，要求交点的n条线段中没有垂直的线段(规定的次序关系对垂直的线段不适合)，也没有三条以上的线段交于一点的情况。

两线段相交的必要条件：

若两线段相交,则必然存在某个x,使它们在规定的次序关系>x下是相邻的。

算法从左向右扫描,在扫描过程维持正确的线段间上述次序关系。这种次序关系只能有三种可能的变化方式:

1. 遇见某条线段S的左端点,此时S应加入次序关系。
2. 遇见某线段S的右端点,此时S应从次序关系中删除。
3. 遇到某两条线段S_1和S_2 的交点,这时在次序关系中S_1和S_2交换位置。

算法的数据结构和实现过程：

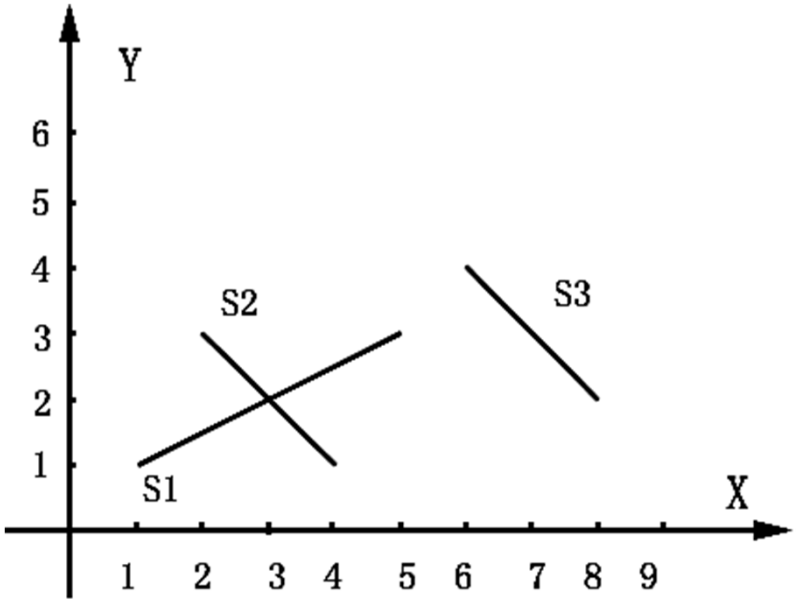
算法实施需要两个基本的数据结构：

扫描线状态表和事件点进度表

扫描线状态表**L**：存放按次序关系$>_x$排序的线段的序列。初始为空，扫描过程中当关系$>_x$改变时变化。

事件点：扫描过程中可能使次序关系$>_x$发生变化的点，

事件点进度表**E**：存放事件点的表，初始为**n**条线段的**2n**个端点，在平面扫描过程中求出的交点，应及时地插入到事件点进度表中。



扫描线状态表应能支持以下四个操作:

(1)INSERT(S,L),把线段S插入到扫描线状态表L中,注意应插入到适当位置以保持正确的次序关系。

(2)DELETE(S,L),从L中删除线段S。

(3)ABOVE(S,L),返回次序关系中S上面紧接着 的线段的编号。

(4)BELOW(S,L),返回次序关系中S下面紧接着的线段的编号。

事件点进度表E应能支持以下三个操作:

(1)MIN(E),取出表E中的最小元素。

(2)INSERT(x,E),把横坐标为x的一个点插入到表 E中,插入要使E中事件点存放保持递增次序。

(3)MEMBER(x,E),判定横坐标为x的点是否在事件点进度表E中。

算法:

1.〔事件点进度表E初始化〕将输入待求交点的n条线段的2n个端点按x,y字典式排序后存放于表E中;

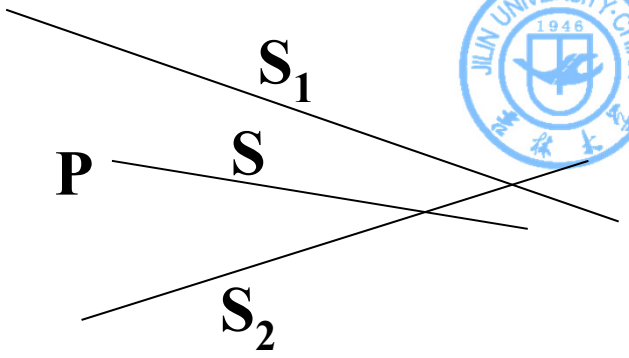
2.〔准备收集交点〕A← φ ;{A是一集合,初为空,准备存入找到的交点;}

3.〔平面扫描〕若表E不为空,则进行3.1~3.3循环。直到表E为空时算法结束。

3.1〔取出当前事件点〕P←MIN(E);

3.2〔当前事件点处理〕考查当前事件点P,分三种情况:

(1) 若P是边S的左端点,则做：



INSERT(S,L);

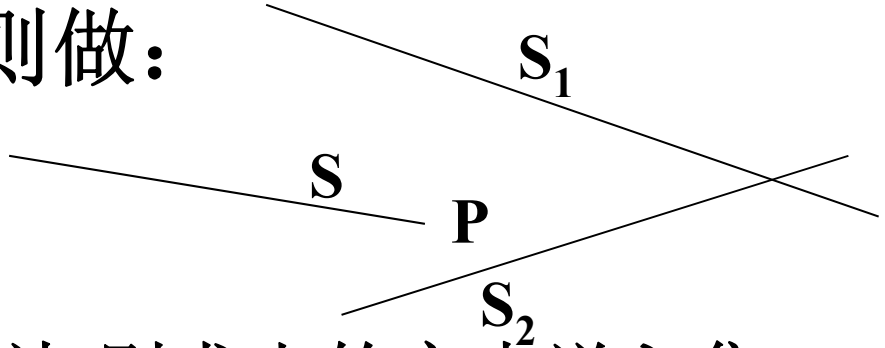
S_1=ABOVE(S,L);

S_2=BELOW(S,L);

若S和S_1相交,则求出的交点送入集合A中;

若S和S_2相交,则求出的交点送入集合A中;

(2) 若P是边S的右端点,则做：



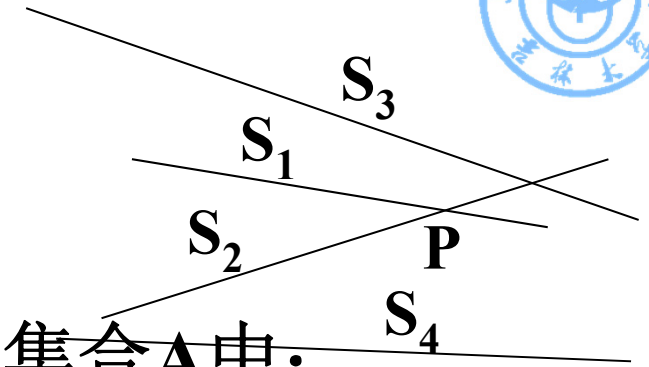
S_1=ABOVE(S,L);

S_2=BELOW(S,L);

若S_1和S_2相交于点P的右边,则求出的交点送入集合A中；

DELETE(S,L);

(3) 若P是边S_1和S_2的交点，且在P的左边



S_1=ABOVE(S_2),则做

S_3=ABOVE(S_1,L);

S_4=BELOW(S_2,L);

若S_3和S_2相交，则求出的交点送入集合A中；

若S_4和S_1相交，则求出的交点送入集合A中；

在L中交换S_1和S_2的位置；

3.3 〔处理找到的交点〕若集合A不为空，做下面循环，直至A为空：

取出集合A中一个交点，其横坐标是x；

若MEMBER(x,E)为FALSE，则输出此交点， 并做INSERT(x,E); (交点是不是第一次求得）

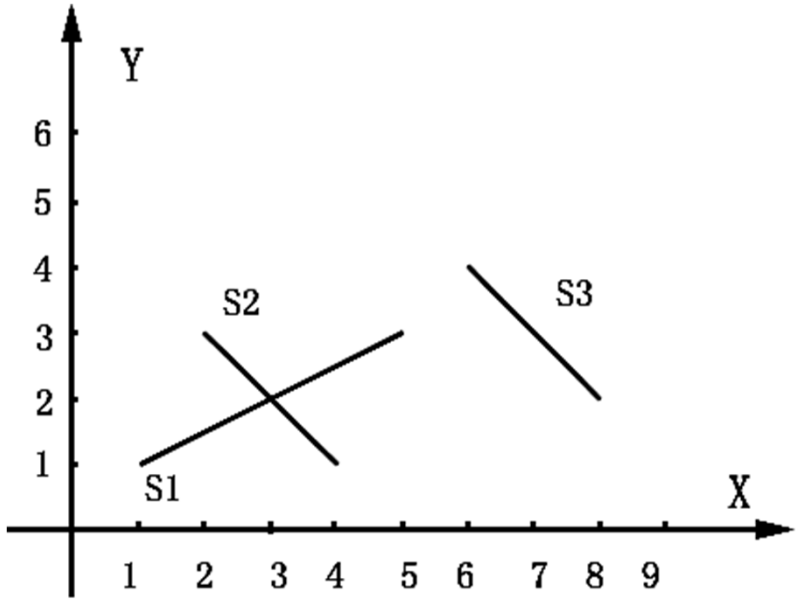
设有三条线段S_1，S_2，S_3,它们的坐标如下(1,1),(5,3,),(2,3),(4,1),(6,4),(8,2).要计算所有交点。

算法初始形成的事件点进度表E，可有形式

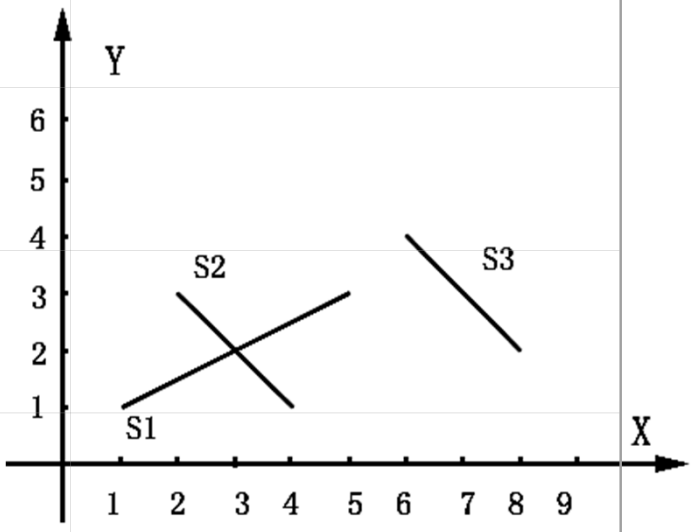
((1,1),S1左端点)，((2,3),S2左端点)，

((4,1),S2右端点)，((5,3),S1右端点)，

((6,4),S3左端点)，((8,2),S3右端点))



算法步骤	从表E前面取出的扫描到达的事件点P	扫描线状态表L	工作解释
3.2(1)	((1,1),S1左端点)	(S1)	
3.2(1)	((2,3),S2左端点)	(S2,S1)	发生S1,S2求交,求出交点(3,2)插入E((4,1),S2右端点)前
3.2(3)	((3,2),S1和S2的交点	(S1,S2)	交换位置
3.2(2)	((4,1),S2右端点)	(S1)	
3.2(2)	((5,3),S1右端点)	( )	
3.2(1)	((6,4),S3左端点)	(S3)	
3.2(2)	((8,2),S3右端点)	( )	



第二节　多边形表面的交线计算

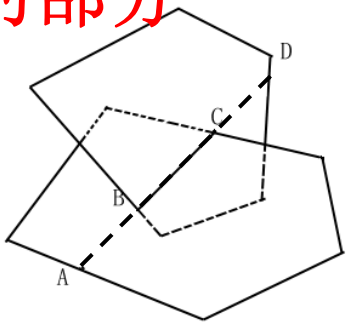
多边形：设空间多面体的表面为凸多边形表面,分别由它们的顶点坐标逆时针方向的序列确定,即约定按顶点序列前行时内部在左侧。

求交步骤：(已知两个多边形顶点序列)

1.根据顶点坐标求出两个多边形表面分别所在平面的方程

2.根据平面方程判断多边形所在平面是否相交

3.确定出交线同时在两个多边形表面内部的部分



1.求平面方程

采用多个顶点位置坐标来计算平面方程可以减少由于不共面而引起的偏差。

设要求出通过若干顶点的平面方程 **Ax+By+Cz+D=0**,即要确定系数**A,B,C,D**,可采用如下做法

平面方程**Ax+By+Cz+D=0**的系数**A,B,C**与该平面上多边形分别在**x=0,y=0,z=0**三个坐标平面上投影的面积成比例

α,β,γ分别为平面Ax+By+Cz+D=0与x=0,y=0,z=0平面的夹角。

Ax+By+Cz+D=0的法向量为(A,B,C)， x=0的法向量为(1,0,0),y=0的法向量为(0,1,0),z=0的法向量为(0,0,1)

$\cos\alpha = (A,B,C)\cdot(1,0,0) = A = S_x/S$

$\cos\beta = (A,B,C)\cdot(0,1,0) = B = S_y/S$

$\cos\gamma = (A,B,C)\cdot(0,0,1) = C = S_z/S$

- 多边形在z=0平面上投影的面积S可如下求出：

$$s = \frac{1}{2}\sum_{i=1}^{n}(x_i - x_j)(y_i + y_j)$$

式中若i=n则j=1,否则j=i+1。

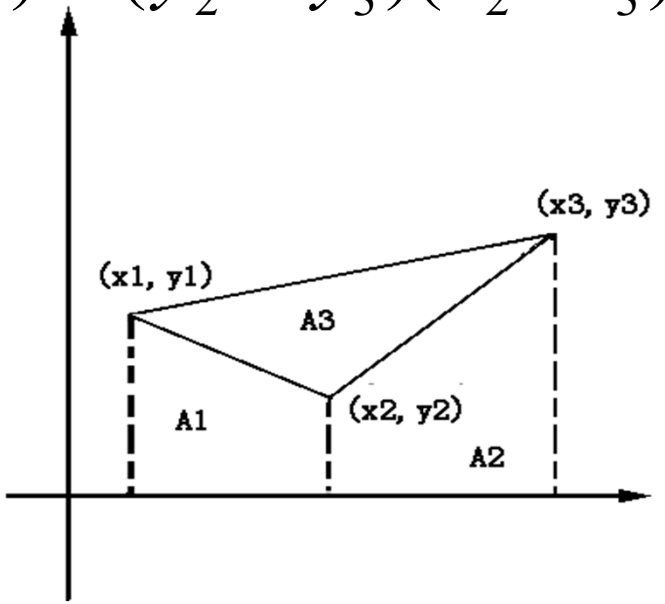
类似地可计算多边形表面在x=0和y=0平面上投影的面积,从而确定A,B,然后D可通过代入平面上一点坐标数值来求出。

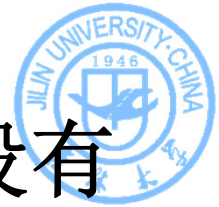
$$-A1 = \frac{1}{2}(y1 + y2)(x1 - x2)$$

$$-A2 = \frac{1}{2}(y2 + y3)(x2 - x3)$$

$$A1 + A2 + A3 = \frac{1}{2}(y3 + y1)(x3 - x1)$$

$$S = \frac{1}{2}[(y_1 + y_2)(x_1 - x_2) + (y_2 + y_3)(x_2 - x_3) + (y_3 + y_1)(x_3 - x_1)]$$





若给出空间若干点的坐标

$(x_1,y_1,z_1),(x_2,y_2,z_2),\ \ldots\ ,(x_n,y_n,z_n)$,注意这里没有要求这些点共面或围成了凸多边形,都可以求出通过或接近这些点的一个平面方程

Ax+By+Cz+D=0:

$$A=\frac{1}{2}\sum_{i=1}^{n}(y_i-y_j)(z_i+z_j)$$

$$B=\frac{1}{2}\sum_{i=1}^{n}(z_i-z_j)(x_i+x_j)$$

$$C=\frac{1}{2}\sum_{i=1}^{n}(x_i-x_j)(y_i+y_j)$$

$$D=-Ax_1-By_1-Cz_1$$

式中若i=n,则j=1,否则j=i+1

2.平面方程的求交 $A_1x+B_1y+C_1z+D_1=0$

$A_2x+B_2y+C_2z+D_2=0$

$$\frac{A_1}{A_2}=\frac{B_1}{B_2}=\frac{C_1}{C_2}$$

两平面重合或平行，没有交点

否则，两平面方程联立，可以认为是空间的交线式方程

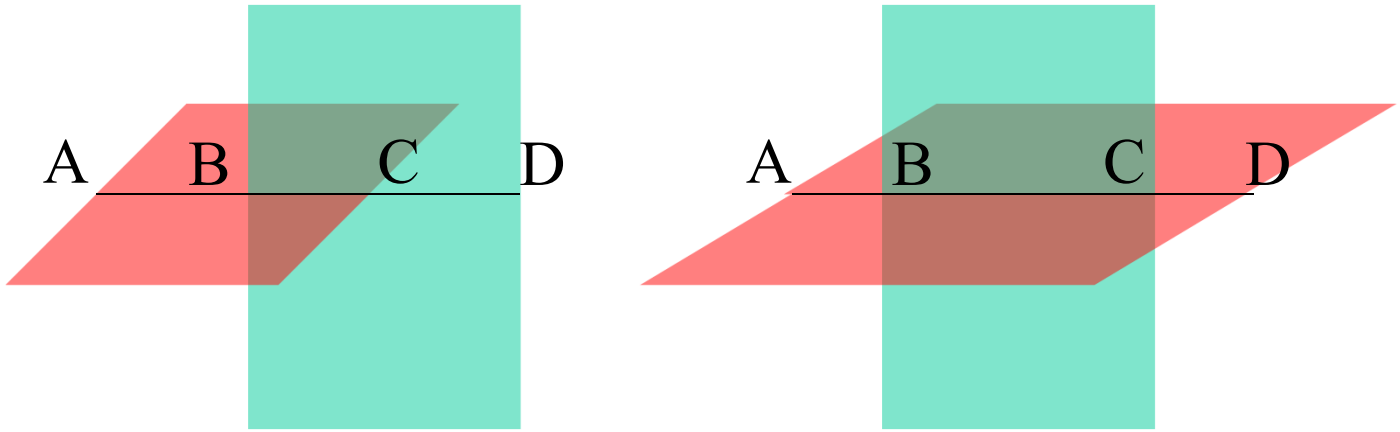
3.确定交线同时在两个多边形内部的部分

分别对每个多边形表面各边相应的线段,计算它与另一个多边形表面所在平面的交点。

注意：求线段（有限）与平面（无限）的交点,即交点在线段延长线上时算不相交。

假定两个多边形表面都是凸的,故共可以交出四个交点。

交得的四个交点可按x,y,z坐标字典式排序，中间两个交点的连线是两个多边形表面相交所得的线段。



求线段（有限）与平面（无限）的交点

空间线段两个端点的坐标(x1,y1,z1)和(x2,y2,z2)给出,平面方程Ax+By+Cz+D=0。

$$\begin{cases} x = x_1 + (x_2 - x_1)t \\ y = y_1 + (y_2 - y_1)t \\ z = z_1 + (z_2 - z_1)t \end{cases}$$

代入平面方程,得:

$A(x_1+(x_2-x_1)t)+B(y_1+(y_2-y_1)t)+C(z_1+(z_2-z_1)t)+D=0$

整理得到:

$[A(x_2-x_1)+B(y_2-y_1)+C(z_2-z_1)]t=-(Ax_1+By_1+Cz_1+D)$

若 $A(x_2-x_1)+B(y_2-y_1)+C(z_2-z_1)=0$

即(A,B,C)和$(x_2-x_1, y_2-y_1, z_2-z_1)$点积为零，则所给线段在平面上或与平面平行，没有唯一确定的交点。否则，交点对应的参数t可以求出:

$$t=-\frac{Ax_1+By_1+Cz_1+D}{A(x_2-x_1)+B(y_2-y_1)+C(z_2-z_1)}$$

- 若t<0或t>1,交点在线段延长线上，没有交点
- 若0≤t≤1，将参数t代回直线的参数方程求出交点的坐标

空间两多边形的交线计算算法如下：


```
int Polygon_Polygon(POINT P[],int n,POINT Q[],int m ){
/* P、Q表示空间两多边形，n、m表示边数目 */
        PLANE(P,n,A1,B1,C1,D1);//计算P的平面方程
        PLANE(Q,m,A2,B2,C2,D2);// 计算Q的平面方程
        // 无交线，算法结束
        if(A1/A2== B1/B2 && A1/A2==C1/C2) return 0;
        //计算P中每边与Q面的交点，若有保存两交点,s为交点个数
        CACULATE(P,n,A2,B2,C2,D2,P1,P2,s);
        if (s==0) return 0; // 无交线，算法结束
        //计算Q中每边与P面的交点，若有保存两交点,s为交点个数
        CACULATE(Q,m,A1,B1,C1,D1,P3,P4,s);
        if(s==0) return 0; // 无交线，算法结束
        //测试四点的位置关系，输出交线部分
        OUTPUT(P1,P2,P3,P4);
}
```

第三节 平面中的凸壳算法

凸壳 包含一个平面点集的最小凸区域

凸区域指要求区域内任意两点的连线仍在该区域内。

设S是平面上n个点的集合,则S的凸壳是一个凸多边形,它包含所有n点且面积最小。

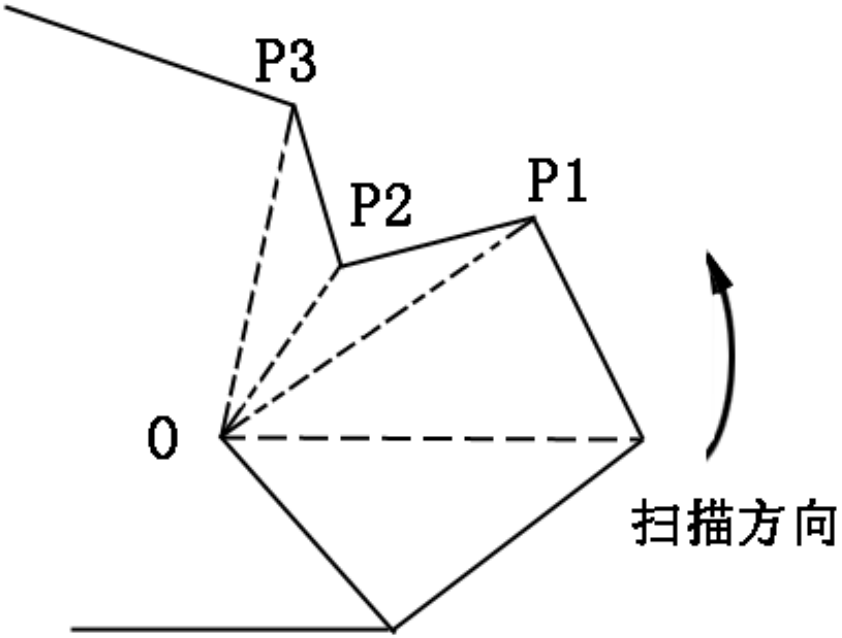
求点集S的凸壳:

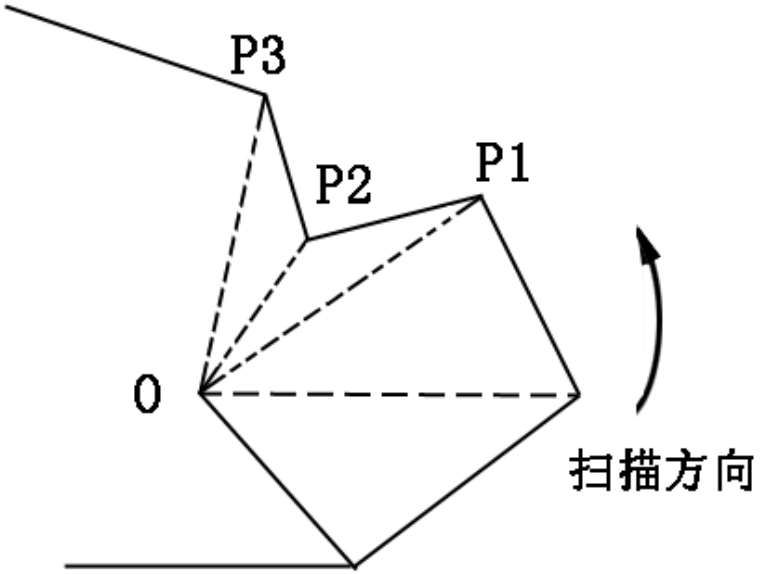
1）在S中选出壳上的点

2）给出围成凸多边形的序列。

- **1.Graham扫描算法**

  思想：设有一内点O，设为原点,对点集S中所有的点计算与OX轴的倾角，并按照倾角递增排序,若某一点不是凸顶点,则它必然在两个壳顶点与点O形成的三角形内部,则删除这个内点。





Graham扫描的实质是围绕已经按“倾角”排序的各顶点进行一次扫描,在扫描过程中消去在凸壳内部的点,留下按希望次序排列的凸壳顶点。

由于是按倾角递增排序,可知若连续三个顶点P_1,P_2,P_3是一个“右转”，则P_2是一个应去掉的内点。

对给出的三点P1,P2,P3,设它们的坐标是(x1,y1),(x2,y2),(x3,y3),这时要判断三点在P2处形成一个右转还是左转,可以计算下面的行列式

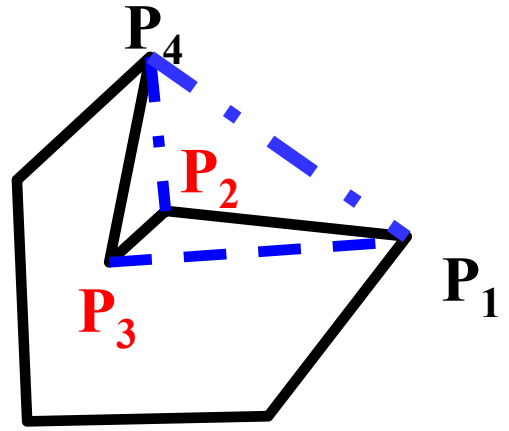
$$\Delta = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

$$= \frac{1}{2}[(x_1 - x_2)(y_1 + y_2) + (x_2 - x_3)(y_2 + y_3) + (x_3 - x_1)(y_3 + y_1)]$$

其中△给出的是带有正负号的三角形$P_1P_2P_3$面积的,因此若△>0,则P_1,P_2,P_3是左转; △ <0,则是右转; △ =0,则三点共线。

内点O： S中x坐标最小的点

```
void  Graham(POINT S[],int n){// S为点集数组，n为点的个数
        //依据边界点O将S中点按倾角排成序放置在Q中，
        //Q中起始点为O
        sortangle(S,n,Q);
        v=first(Q);//取出Q 中起始点
        // 若Q中v的下一个点不是起始点
        while (next(v,Q)!=first(Q)){
                //若连续三点左转
                if (left(v,next(v,Q),next(next(v,Q),Q)))
                        v=next(v,Q);//v前进至下一个点
                else{
                        delete(next(v, Q),Q); //删除v的下一个点
                        v=pred(v,Q);      //v退回至前一个点
                }
        }
}
```

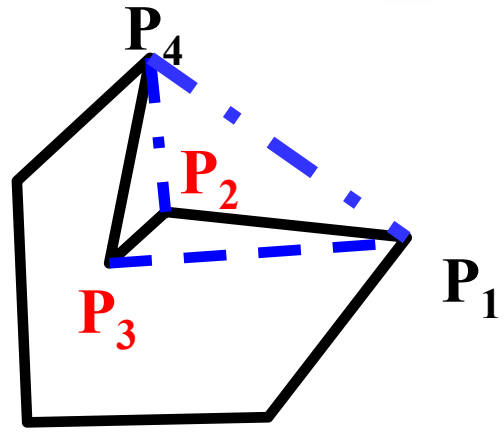


$P_1P_2P_3$左转，
$P_2P_3P_4$右转，
$P_1P_2P_4$右转，
$P_0P_1P_4$左转，

next(v)：返回点序列中的v的下一个点（后继节点）
pred(v)：返回点序列中的v的前一个点（前驱节点）
循环链表：$next(Q_n)=Q_1, pred(Q_1)=Q_n$

内点O： S中x坐标最小的点

```
void  Graham(POINT S[],int n){// S为点集数组，n为点的个数
        //依据边界点O将S中点按倾角排成序放置在Q中，
        //Q中起始点为O
        sortangle(S,n,Q);
        v=first(Q);//取出Q 中起始点
        // 若Q中v的下一个点不是起始点
        while (next(v,Q)!=first(Q)){
                //若连续三点左转
                if (left(pred(v,Q),v,next(v,Q))
                        v=next(v,Q);//v前进至下一个点
                else{
                        delete(next(v, Q),Q); //删除v的下一个点
                        v=pred(v,Q);      //v退回至前一个点
                }
        }
}
```



$P_1P_2P_3$左转，
$P_2P_3P_4$右转，
$P_1P_2P_4$右转，
$P_0P_1P_4$左转，

next(v)：返回点序列中的v的下一个点（后继节点）
pred(v)：返回点序列中的v的前一个点（前驱节点）
循环链表：$next(Q_n)=Q_1, pred(Q_1)=Q_n$

倾角计算

记P点坐标为(Xp,Yp),O点坐标(X_0,Y_0),这个角度数可如下简单地计算:

$$A=\frac{(x_p-x_0)}{\sqrt{(x_p-x_0)^2+(y_p-y_0)^2}}=\cos\theta$$

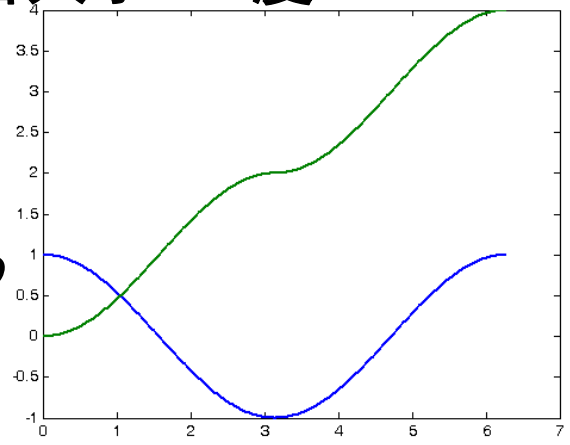
$B=y_p-y_0$

若$B\geq 0$，则角度数=1-A, $(0\leq\theta\leq 180^o)$

若$B<0$，则角度数=3+A。$(180^o\leq\theta\leq 360^o)$

这里A是向量OP与OX向上单位向量的内积,因此是倾角的余弦数值。B用以判断该倾角是否大于180度。

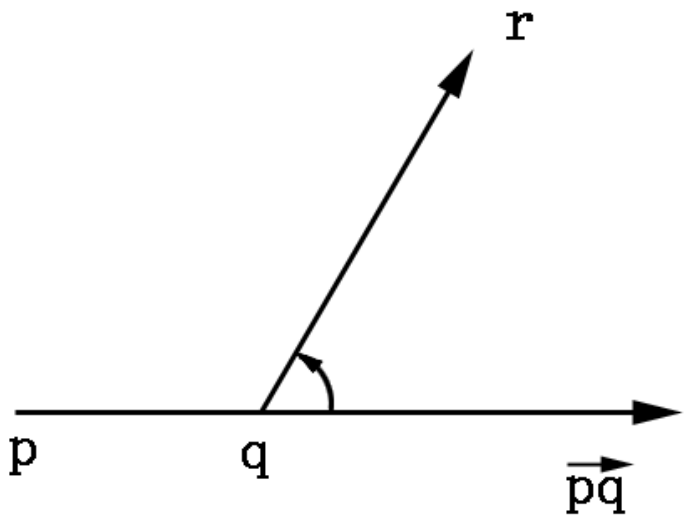
$$y=\begin{cases}1-\cos\theta & 0\leq\theta\leq 180^o\\ 3+\cos\theta & 180^o\leq\theta\leq 360^o\end{cases}$$



2.Jarvis行进算法

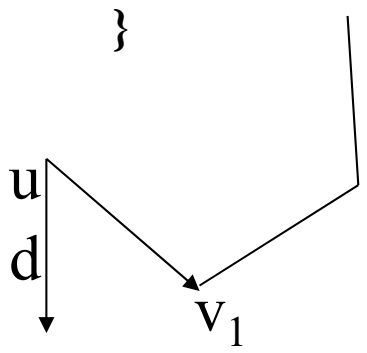
思想：若相继两点构成的边是凸壳的一条边则对于过该边的直线，所有凸壳中点集中的点在该直线同侧。因此若找到pq是壳上一边,则以q为端点的下一条壳边qr可以如下求出：计算点集中其余各点相对q点发出沿向量pq向的射线的倾角,若倾角最小者对应的点是r,则qr是下一条壳边。

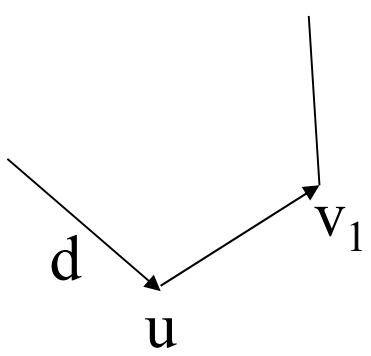
第一条壳边的确定：把点集S中所有点按x,y字典式排序，取最低点，该点必定是一个壳顶点。 从该点引一条竖直向下的射线，在此做一个行进步就找到了第一条壳边。

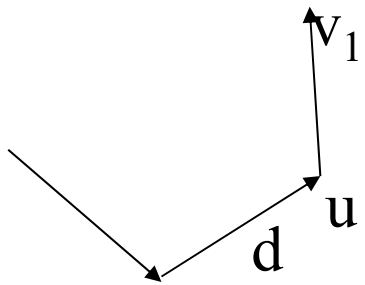


Jarvis算法

```
void  Jarvis(POINT S[],int n){// S为点集数组，n为点的个数
        v0=xy_minsort (S,n);//xy_minsort求S中xy排序最小点
        d=(0,-1);//d送竖直向下向量
        Q=null;//队列Q置空
        add(Q,v0);//v0添加到队列Q中
        S1=delete(v0,S);//将S1设置成仅在S集合中删去v0点的集合
        u=v0;
        //取出S1中与u点连接并与d方向夹角最小的点，若角度一样取最远点
        v1=wrapping(u,d,S1);
        while(v1!=v0){
                add(Q,v1);//v1加入队列Q中
                //S1置为删除v1、u两点的后包含其余点的点集合
                S1=delete(v1,S); S1=delete(u,S1);
                d= vector(u,v1);//d置为由u到v1的向量
                u=v1;
                v1=wrapping(u,d,S1);
        }
}
```

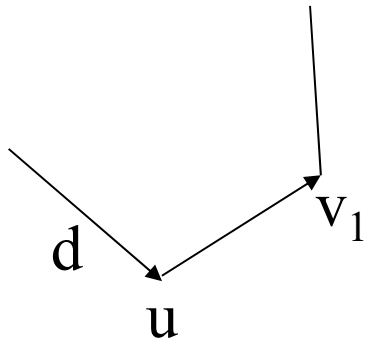






函数**Wrapping**来实现一步行进。此函数的工作是,对S_1中所有各点, 计算射线**d**与自**u**向各顶点发出的射线所成倾角。在所有倾角中取最小，若有多个，则选与**u**距离最远的那个，它对应的点为函数的返回值。因此在步**2**求得的v_1是一个行进步找到的下一个凸壳顶点。

若点集**S**中只有较少数点在壳上,则**Jarvis**算法效率很高。若点集**S**中绝大多数点都在壳上,算法的效率一般来说就不如**Graham**扫描了。



第四节　包含与重叠

一.点对简单多边形的包含算法

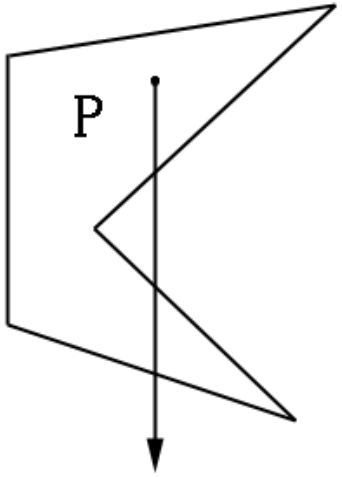
平面上的简单多边形是不相邻的边不能相交的多边形。

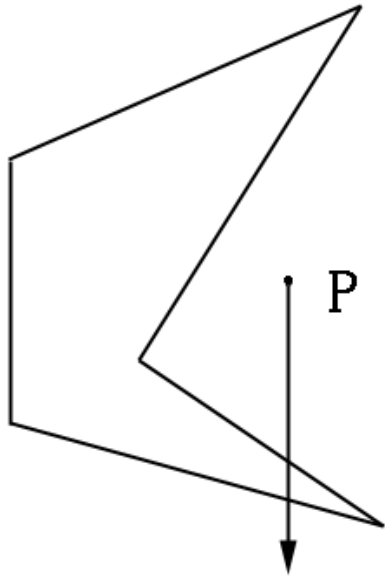
设它用顶点坐标的逆时针序列$(x_0,y_0),(x_1,y_1),\ldots,(x_{n-1}, y_{n-1})$确定，即沿顶点序列前行时内部在左侧。对平面上坐标为(x_p,y_p)的任意一点P,包含性检验问题是判断该点是否在所给出简单多边形的内部。

思想：由P竖直向下引射线,计算此射线与多边形各边交点的个数。

奇数：P在多边形内部

偶数：P在多边形外部

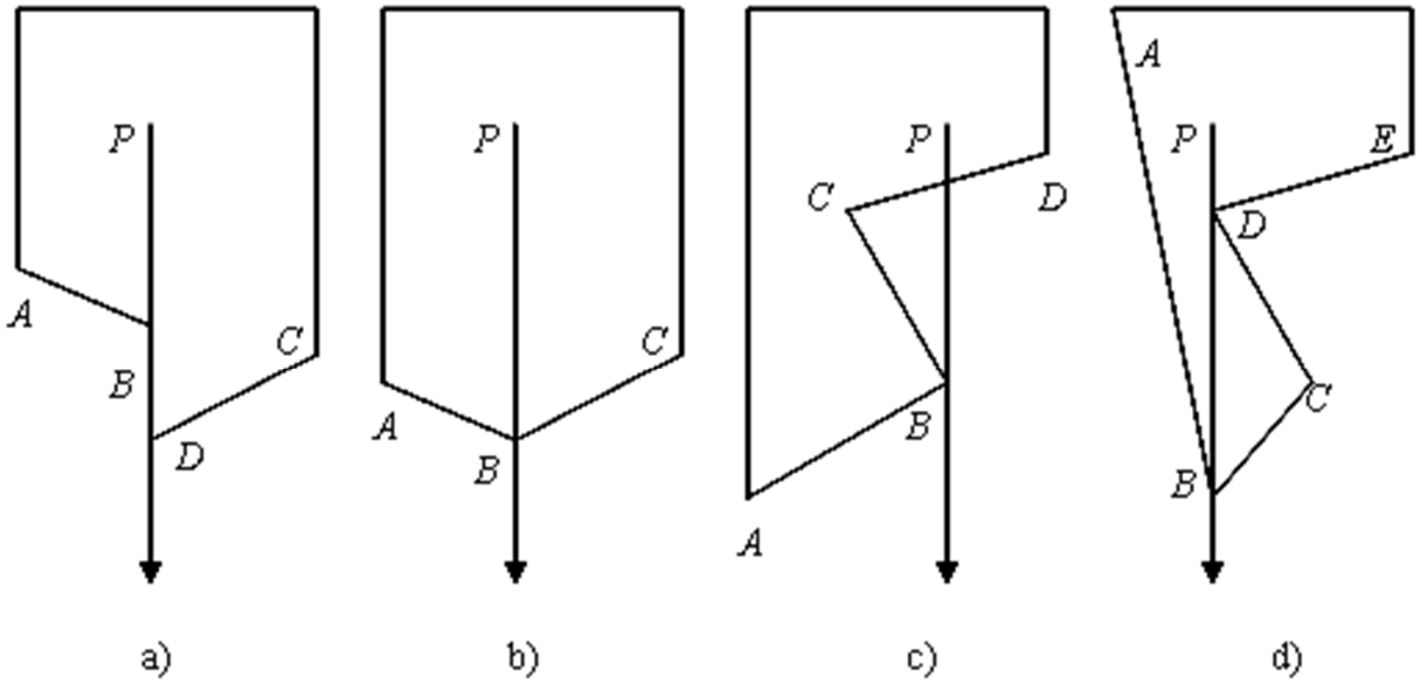




特殊情况处理：当由点P竖直向下的射线恰好通过多边形的顶点或某一边时,交点计数可采取简单的"左闭右开"法来处理

即:当点P的x坐标大于或等于多边形一边的两个顶点的x坐标时,相应交点不计算在内（左闭）

只要点P的x坐标小于多边形一边两端点的一个x坐标，交点就计数（右开）



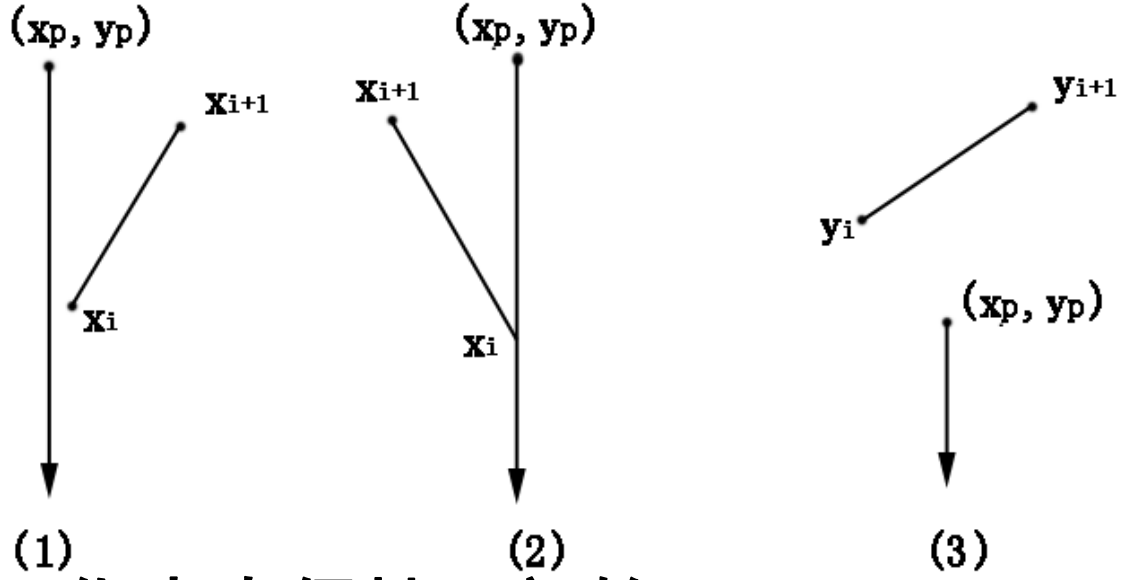
注意 ： 射线均为竖直向下的

为了减少计算量，可判断不可能相交的情况

（1）$x_p<x_i$并且$x_p<x_{i+1}$：

（2）$x_p\geq x_i$并且$x_p\geq x_{i+1}$；

（3）$y_p<y_i$并且$y_p<y_{i+1}$；



算法中用m代表奇偶性：初始 m=-1， $m(-1)^c$

C为偶数　m　负　　点在多边形外部

C为奇数　m　正　　点在多边形内部

简单多边形包含性检验的算法

```
int inner(POINT q[],int n,POINT p){ // q为多边形顶点数组 p为待测点
        int m,i,y;
        q[n]=q[0];
        m=-1; i=0;
        while (i<n)        {
                if ((p.x<q[i].x && p.x<q[i+1].x)|| (p.x>=q[i].x &&
                  p.x>=q[i+1].x)||(p.y<q[i].y && p.y<q[i+1].y)) ;
                else {
                        y=q[i].y+(p.x-q[i].x)*(q[i+1].y-q[i].y)/(q[i+1].x-q[i].x);
                        if (y==p.y) return 1;
                        if (y<p.y) m=m*(-1);
                }
                ++i;
        }
        if (m==-1) return 0; //点P在多边形外部
        else return 1; //点P在多边形内部
}
```

$$y = y_i + (x - x_i)\frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

二.凸多边形的包含算法

点P坐标为(x_p,y_p),设直线的端点P_1为(x_1,y_1),P_2为(x_2,y_2),

$$\overrightarrow{P_1P}\times\overrightarrow{P_1P_2}=\begin{vmatrix} x_p-x_1 & y_p-y_1 & 0 \\ x_2-x_1 & y_2-y_1 & 0 \\ i & j & k \end{vmatrix}=k[(y_2-y_1)x_p+(x_1-x_2)y_p+x_2y_1-x_1y_2]$$

k为指向纸面外的单位矢量

直线方向是由(x_1,y_1)到(x_2,y_2)的方向。

直线的方程记为$Ax+By+C=0$,则有:

$A=y_2-y_1, B=x_1-x_2, C=x_2y_1-x_1y_2$

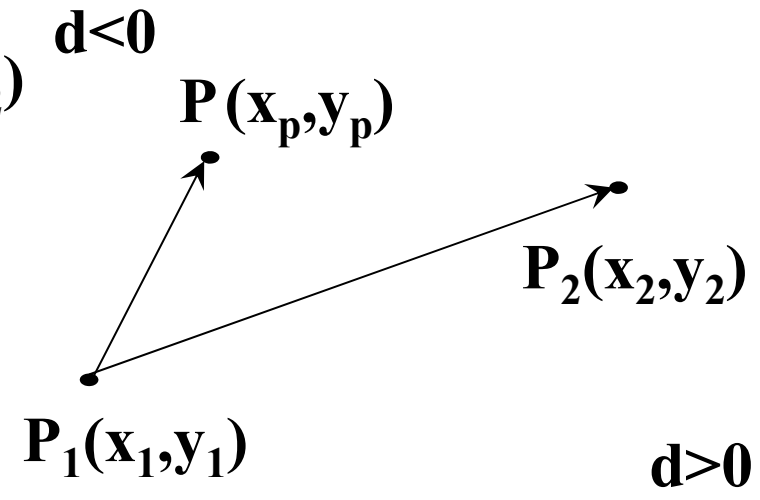
计算D:

$D=(y_2-y_1)x_p+(x_1-x_2)y_p+(x_2y_1-x_1y_2)$

若D<0,则点在直线左侧;

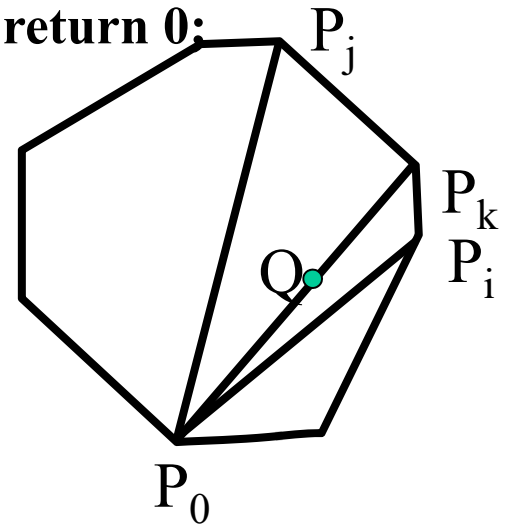
若D>0,则点在直线右侧;

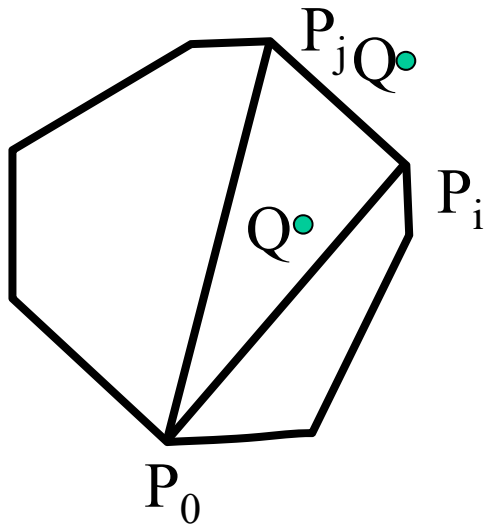
若D=0,则点在直线上。



“折半查找”

```
/* p凸多边形顶点数组 n多边形边数 q待测点 */
int  inner1(POINT p[],int n,POINT q){
        int i,j,k;
        i=1;j=n-1;
        while (j-i>1){
                k=(i+j)/2;//折半查找
                if (test(q,p,0,k)==0) return 1;//q在p[0]和p[k]线段内部
                else if (test(q,p,0,k)==1) return 0; //q在p[0]和p[k]线段延长线上
                else if (test(q,p,0,k)==2) i=k; //q在在p[0]和p[k]线段左侧
                else if (test(q,p,0,k)==3) j=k; //q在在p[0]和p[k]线段右侧
        }
        if (test3(q,p,0,i,j))  return 1;//q在p[0]p[i]p[j]三角形内
        else  return 0;
}
```





三.凸多边形重叠计算

两个凸多边形的重叠问题就是对两个凸多边形求相交部分的问题。约定凸多边形指它的边界和内部,凸多边形仍用顶点坐标的逆时针方向序列确定。

设给出的两个凸多边形P和Q的顶点序列分别是$P_1,P_2,\ldots,P_L$和$Q_1,Q_2,\ldots,Q_m$。

假设P的边界上不包含Q的顶点,Q的边界也不包含P的顶点。有两个问题需要解决:

1.如何有次序地求出各边的所有交点,

2.如何排列求出的交点和原凸多边形的顶点,

形成新的凸多边形的顶点序列。

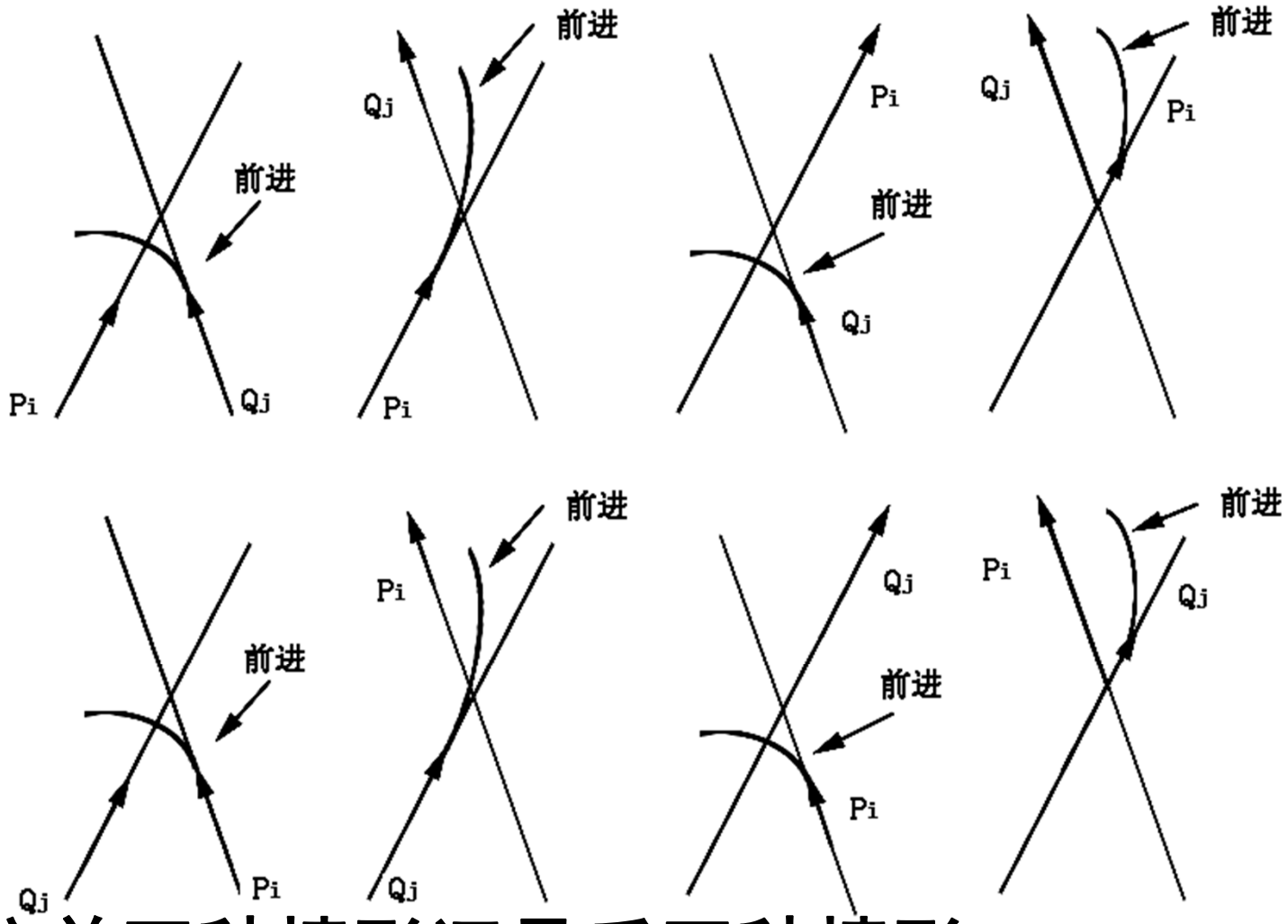
1.求交点:

为了有次序地求出交点,可以在两个多边形边上交替地前进,原则是在哪个多边形的边上可能有交点就等待,在另一个多边形的边上前进。初始从对边P_0P_1与Q_0Q_1的求交开始,注意所有求交是线段的求交。这里规定了$P_0=P_L$,$Q_0=Q_m$。

设已经算得$P_{i-1}P_i$和$Q_{j-1}Q_j$的交点，依可能性判定接下去计算的是P_iP_{i+1}与$Q_{j-1}Q_j$的交点还是$P_{i-1}P_i$和Q_jQ_{j+1}的交点

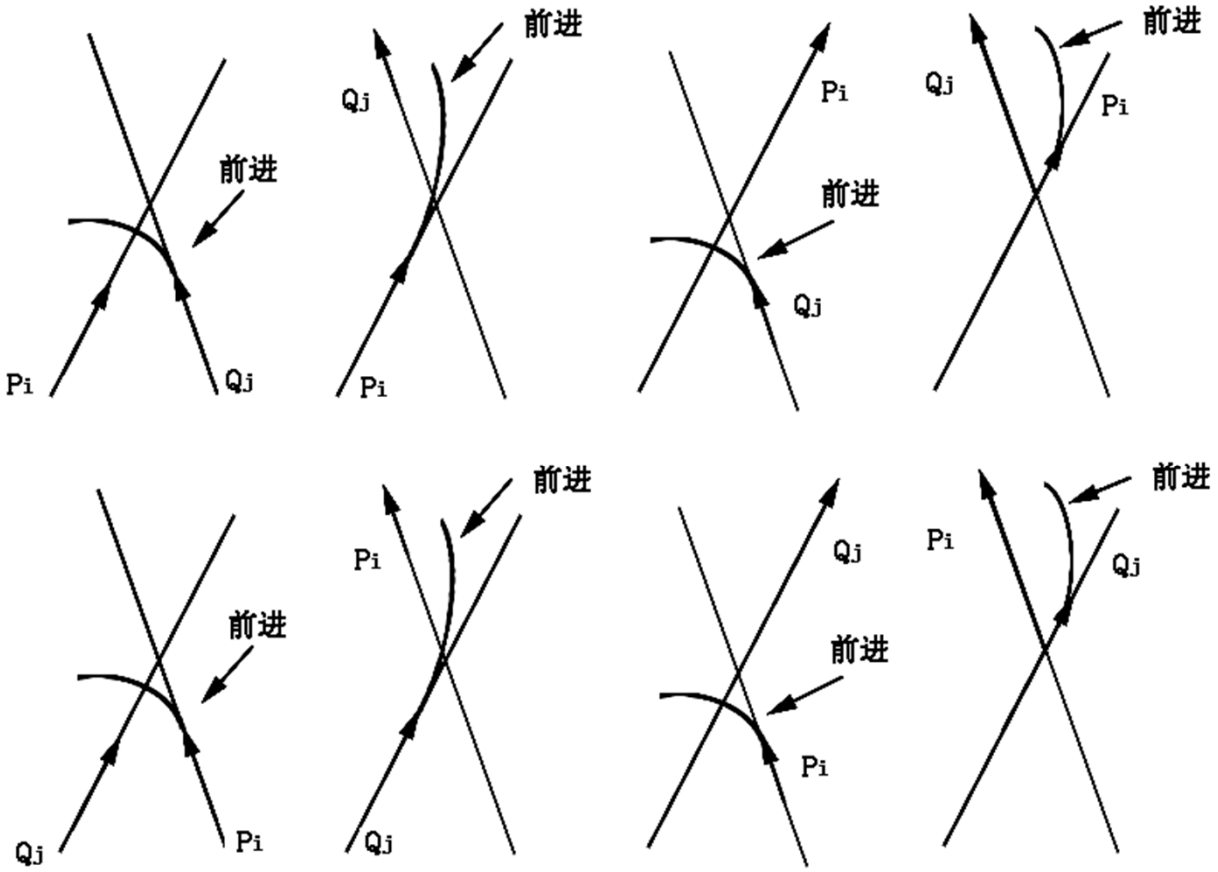
此外还需考虑P_i 和 Q_j之间的夹角是否大于180度

初始从对边P_0P_1与Q_0Q_1的求交开始,注意所有求交是线段的求交。



区分前四种情形还是后四种情形

求矢量积$P_{i-1}P_i \times Q_{j-1}Q_j$的z分量,若该分量大于等于0,是前四种情形,小于0是后四种情形。



情形	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
P_i在$Q_{j-1}Q_j$	左	左	右	右	右	左	右	左
Q_j在$P_{i-1}P_i$	右	左	右	左	左	左	右	右
前进的多边形	Q	P	Q	P	P	Q	P	Q

情形	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
P_i在$Q_{j-1}Q_j$	左	左	右	右	右	左	右	左
Q_j在$P_{i-1}P_i$	右	左	右	左	左	左	右	右
前进的多边形	Q	P	Q	P	P	Q	P	Q

区分前四种情形还是后四种情形

求矢量积$P_{i-1}P_i \times Q_{j-1}Q_j$的z分量,若该分量大于等于0,是前四种情形,小于0是后四种情形。

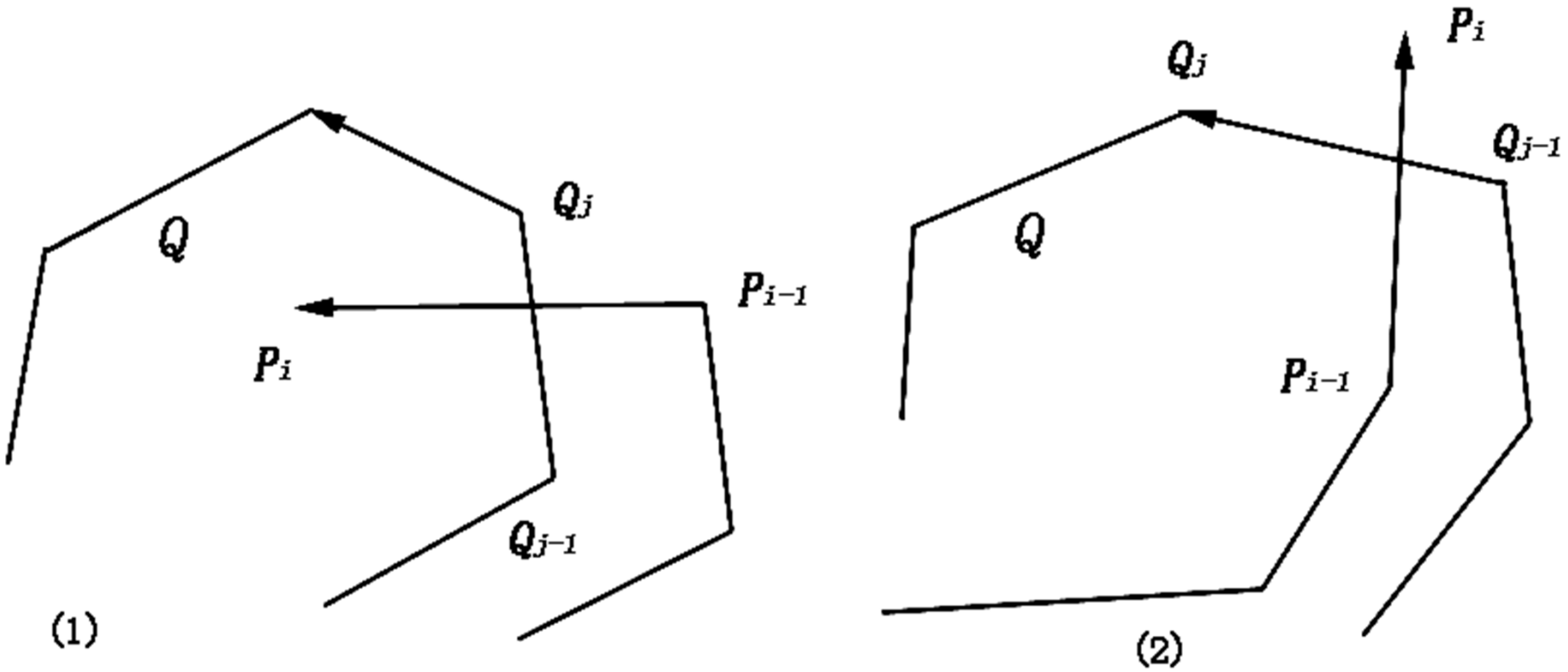
```
/* P、Q为多边形顶点数组，l、m为顶点个数*/
Void Advance(PONIT P[],int l,POINT Q[],int m){
    int s;
    s=vector3(P,Q,i,j);//s置成PiPi-1与QjQj-1的叉乘的z值
    if (s>=0){
        if((left(P,i,Q,j)&& left(Q,j,P,i))||(right(P,i,Q,j)&& left(Q,j,P,i))){
            if (i<l) i++;
            else i=1;
        }else {
            if(j<m) j++;
            else j=1;
        }
    }else{
        if((right(P,i,Q,j)&& left(Q,j,P,i))||(right(P,i,Q,j)&& right(Q,j,P,i))){
            if (i<l)i++;
            else i=1;
        }else{
            if(j<m) j++;
            else j=1;
        }
    }
}
```

情形	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
P_i在$Q_{j-1}Q_j$	左	左	右	右	右	左	右	左
Q_j在$P_{i-1}P_i$	右	左	右	左	左	左	右	右
前进的多边形	Q	P	Q	P	P	Q	P	Q

为了正确排列求出的交点并加入原两个凸多边形部分顶点以形成相交的凸多边形,可以在每求出一个交点时进行一次输出。

求出的第一个交点可只做一下记录,如果在以后交替前进求交点的过程中再次求出与第一次求得相同的交点,就知道整个求交过程已经结束了。

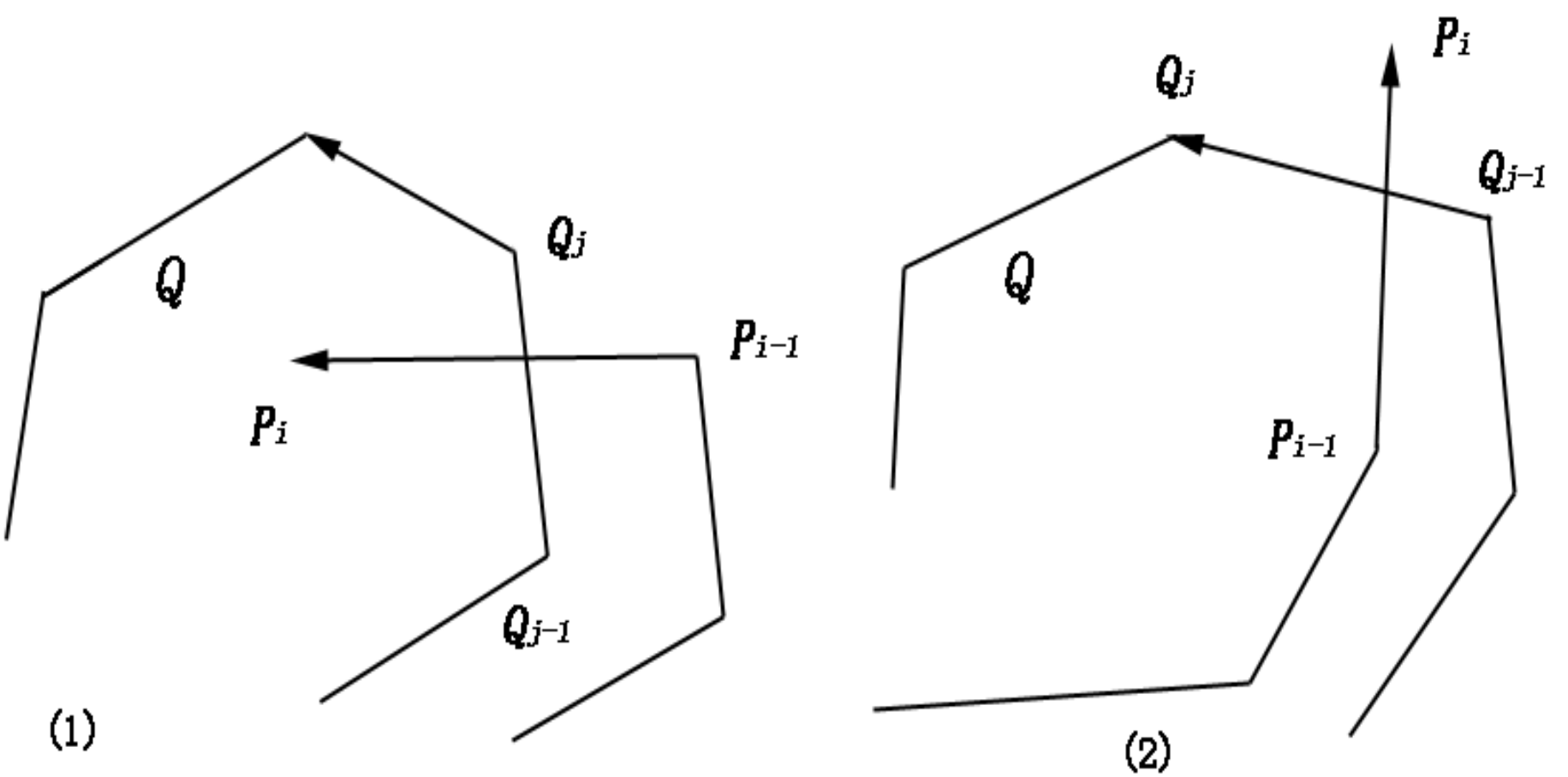
求得一个不是第一个的其它任何一个交点时,为形成交得凸多边形顶点序列,要区分边$P_{i-1}P_i$是进入多边形Q,还是走出Q两种情况。



$P_{i-1}P_i$正进入多边形Q，此时应输出本次求出交点前，上次求得交点后的多边形Q上的各顶点，然后再输出本次交点。

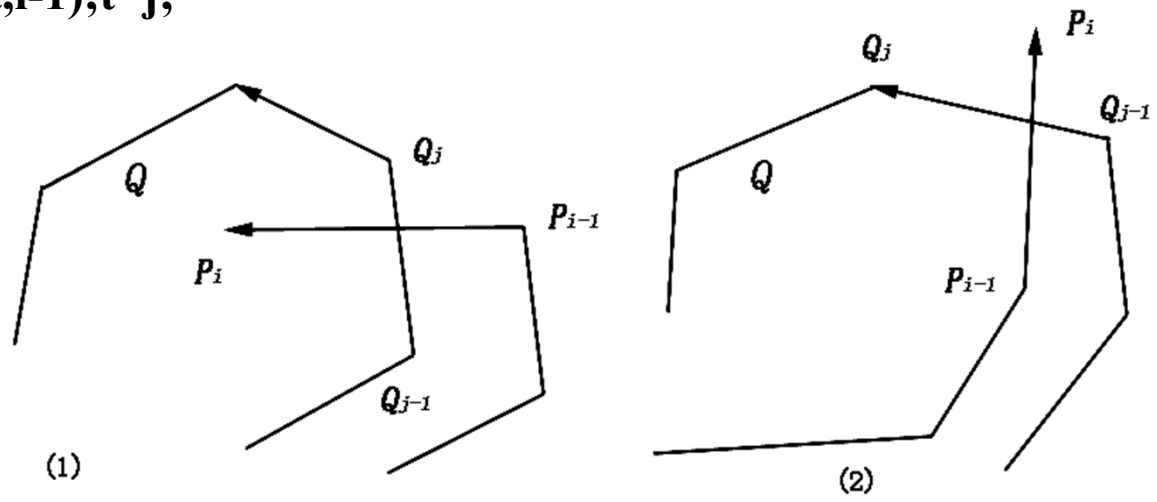
$P_{i-1}P_i$是走出Q，这时应输出本次求出交点前，上次求得交点后的多边形P上的各顶点，再输出本次交点。

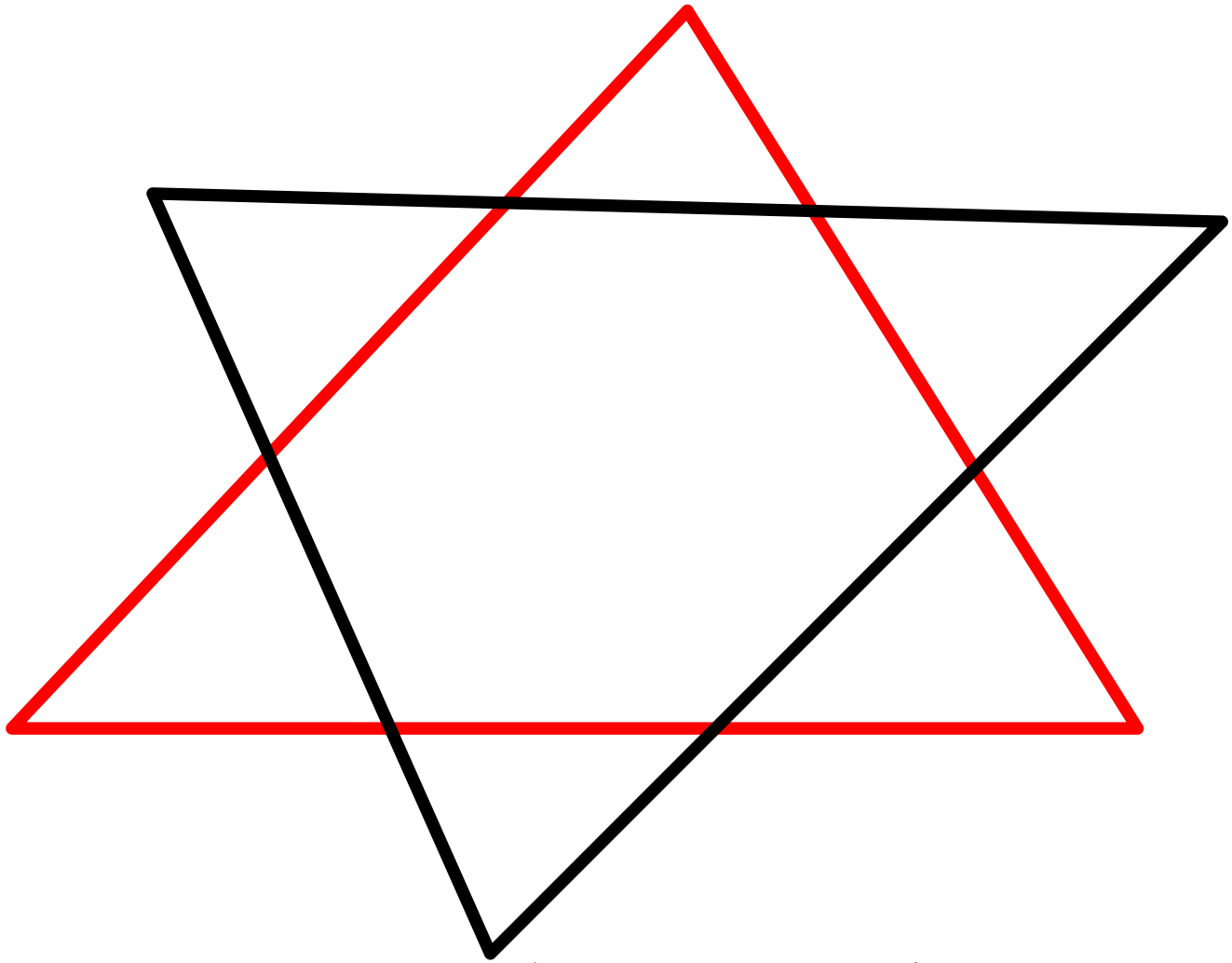
区分这两种情况，可通过检查P_i在直线$Q_{j-1}Q_j$的左侧还是右侧来确定。



$P_{i-1}P_i$正进入多边形Q,此时应输出本次求出交点前，上次求得交点后的多边形Q上的各顶点，然后再输出本次交点，因为这些点是交得凸多边形的顶点。

```
/* P、Q为多边形顶点数组，l、m为顶点个数*/
void Output(PONIT P[],int l,POINT Q[],int m){
        POINT r0;int t;
        if(本过程第一次调用){
                r0=intersect(P,i,Q,j);//r0置成Pi-1Pi与Qj-1Qj之交点
                if (left(P,i,Q,j)) //Pi在Qj-1Qj左
                        t=i;
                else
                        t=j;
        }else {
                if (left(P,i,Q,j)) {//若Pi在Qj-1Qj左，输出Q中t至j-1顶点，输出Pi-1Pi与
Qj-1Qj之交点
                        out(Q,t,j-1);t=i;
                }else {//输出P中t至i-1顶点，输出Pi-1Pi与Qj-1Qj之交点
                        out(P,t,i-1);t=j;
                }
        }
}
```





P和Q，P中的每一条边与P相交最多两个交点，故Q最多与P交2m个交点，同理，P最多与Q交2l个交点，故2(l+m)步是足够的,max(2l,2m)即可

两个凸多边形求交的完整算法:

```
/* P、Q为多边形顶点数组，l、m为多边形边数 */
void  Convex_polygon_intersection(POINT P[],int l,POINT Q[],int m)
{
    int i,j,k,P0,Q0;
    i=1;j=1;k=1;P0=P[l];Q0=Q[m];//初始化
    while (k<=2*(l+m)&& (求得的交点不是第一个交点R0))//交替前进求交
    {
        if (Pi-1Pi与Qj-1Qj相交) Output(P,l,Q,m);//若相交，则调用Output
        Advance(P, l, Q, m);
        k++;
    }
    if(找到过交点) {
        Output(P,l,Q,m);
        return ;
    }else{
        if(inner(Q,m,P[1])) printf("P在Q中");
        else if (inner(P,l,Q[1])) printf("Q在P中");
        else printf("P与Q分离");
    }
}
```

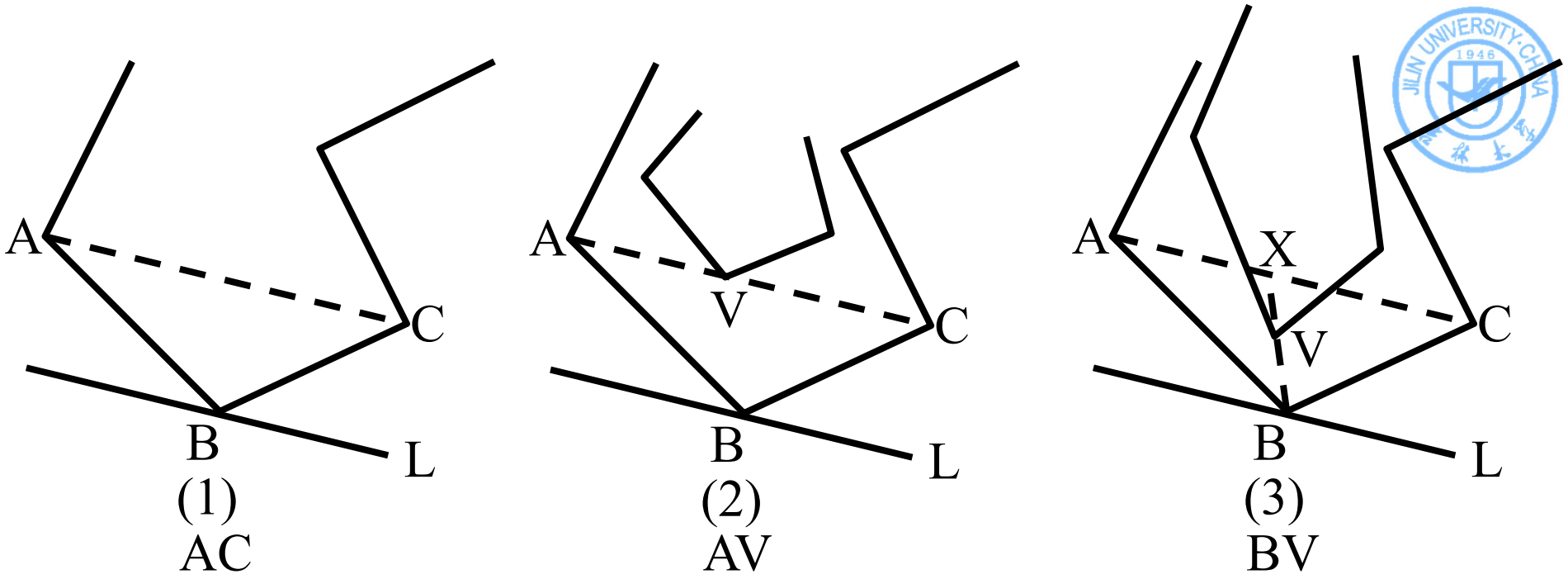
第五节　简单多边形的三角剖分

简单多边形做三角剖分,是要求选出完全在内部又互不相交的一组对角线，把整个多边形划分成一些三角形。

对角线是不相邻顶点间的连线,选出的对角线的集合称为是简单多边形的三角剖分。对任意一个简单多边形,其三角剖分不唯一。

事实1 简单多边形必有一条对角线完全在其内部。

事实2 简单多边形上必有连续的三个顶点A,B,C,使对角线AC完全在其内部。



必有完全在内部的对角线

简单多边形三角剖分的算法:考查连续三个顶点A,B,C,若AC完全在多边形内部,则可输出△ABC为一个剖分后形成的三角形,删除点B后再对少了一个顶点的多边形继续进行。

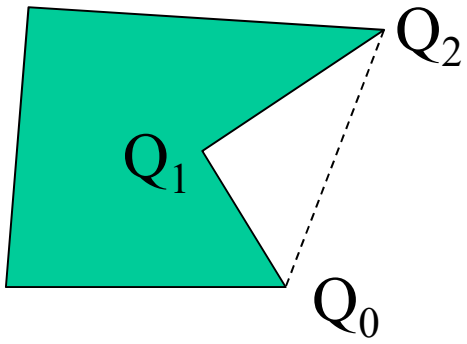
简单多边形的顶点序列为P_0,P_1,P_{n-1},那么算法可描述如下:

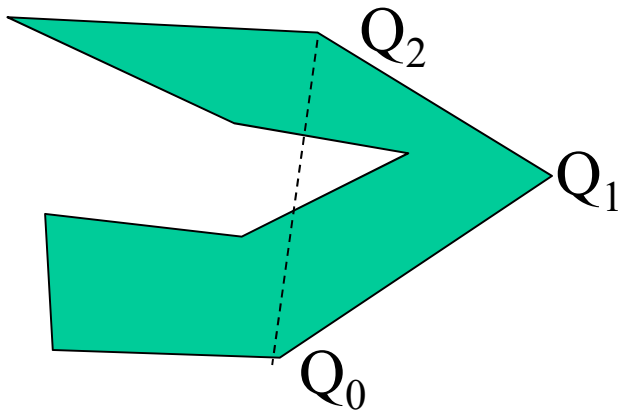
```
/* P为指向多边形链表的首指针 n为多边形顶点数  */
void Simple_polygon_triangulation(POINT *P,int n)
{
    Q0=P;m=n;
    while(m>3)
    {
        Q1=next(Q0);Q2=next(Q1);
        if(Test(Q0,Q1,Q2) {
            Output(Q0);//输出三角形Q0Q1Q2;
            m--;
            delete(Q1);//删除顶点Q1
        }
        else
            Q0=Q1;
    }
    Output(Q0);//输出Q0开始的剩下的三角形;
}
```

函数Test是对△$Q_0Q_1Q_2$进行检查，分两步实现：

第一步检查Q_0,Q_1,Q_2是否是一个在Q_1的左转，若是右转,则Q_0Q_2在多边形外部,可以输出假而结束。

第二步检查可对原多边形中除去Q_0,Q_1,Q_2这三点的其它点，对每一点都考查它对三角形的包含性,若有一点被包含则就可以输出假而结束,只有其它点都在三角形外部时才能输出真而结束。

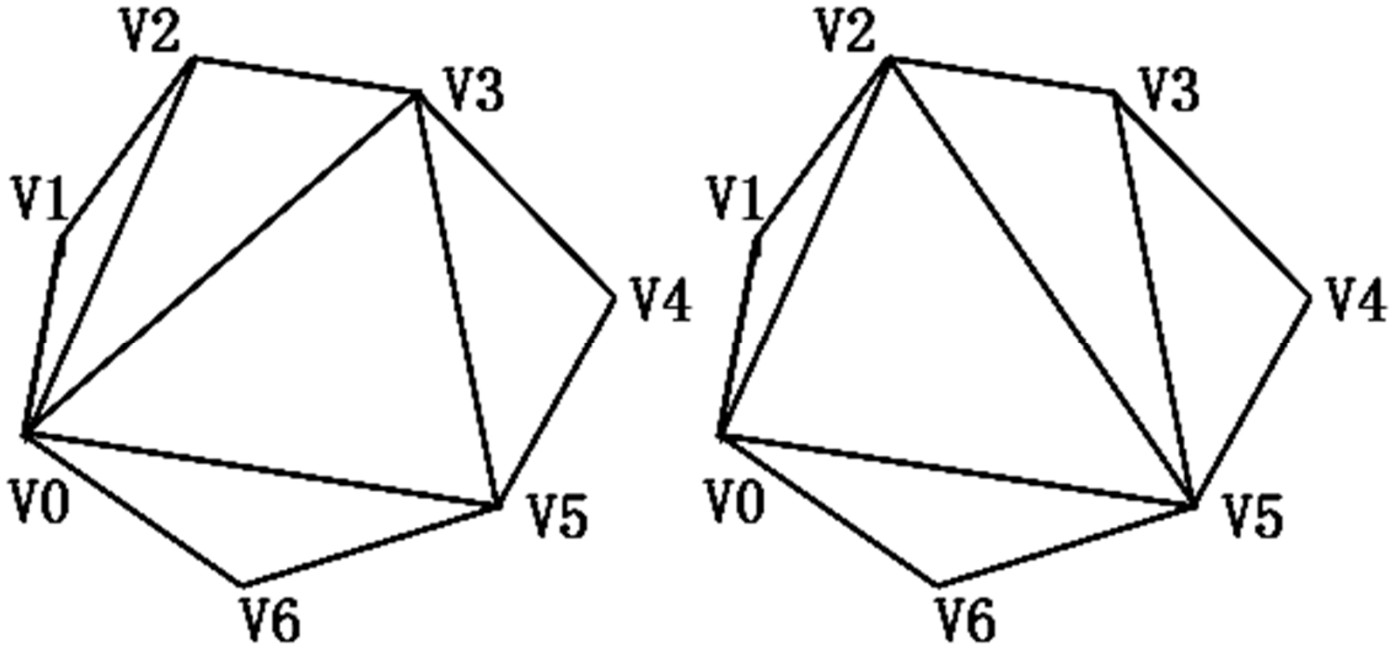




最小权三角剖分或最小三角剖分

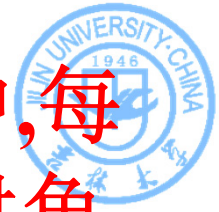
如果一个三角剖分中选取的对角线的总长度最小。

任意的凸多边形最小三角剖分（简化问题）



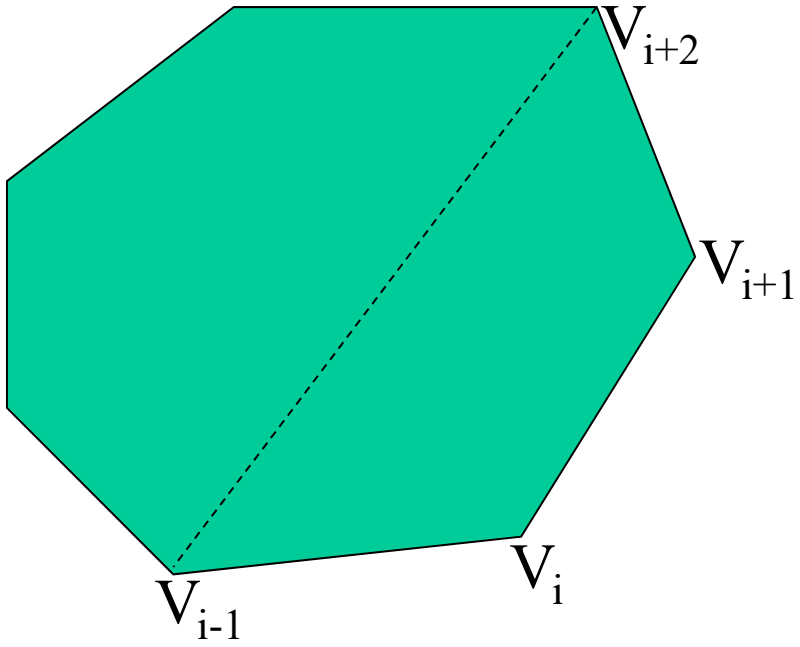
(1) (2)

同一个凸多边形的两种三角剖分



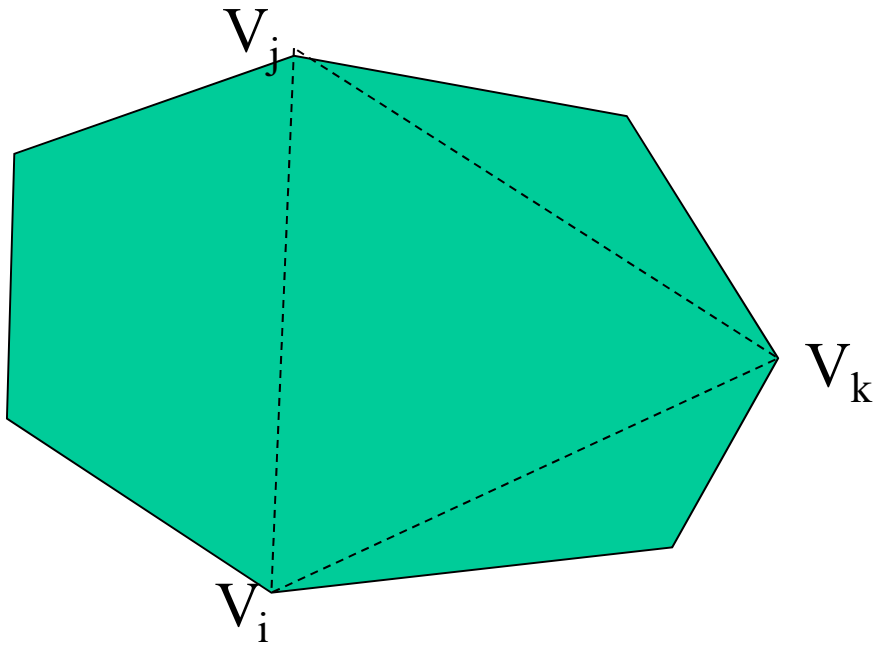
事实3 在n边形(n≥3)的任意一种三角剖分中,每一对相邻顶点中至少有一个顶点是某条对角线的端点。

因为若相邻顶点V_i,V_{i+1}都不是某条对角线的端点,则区域(V_{i-1},V_i,V_{i+1},V_{i+2})中没有对角线,因而也就没有被三角剖分。



事实4 如果V_iV_j是三角剖分的一条对角线,则一定存在某顶点V_k,使得V_iV_k和V_kV_j是多边形的边或对角线。

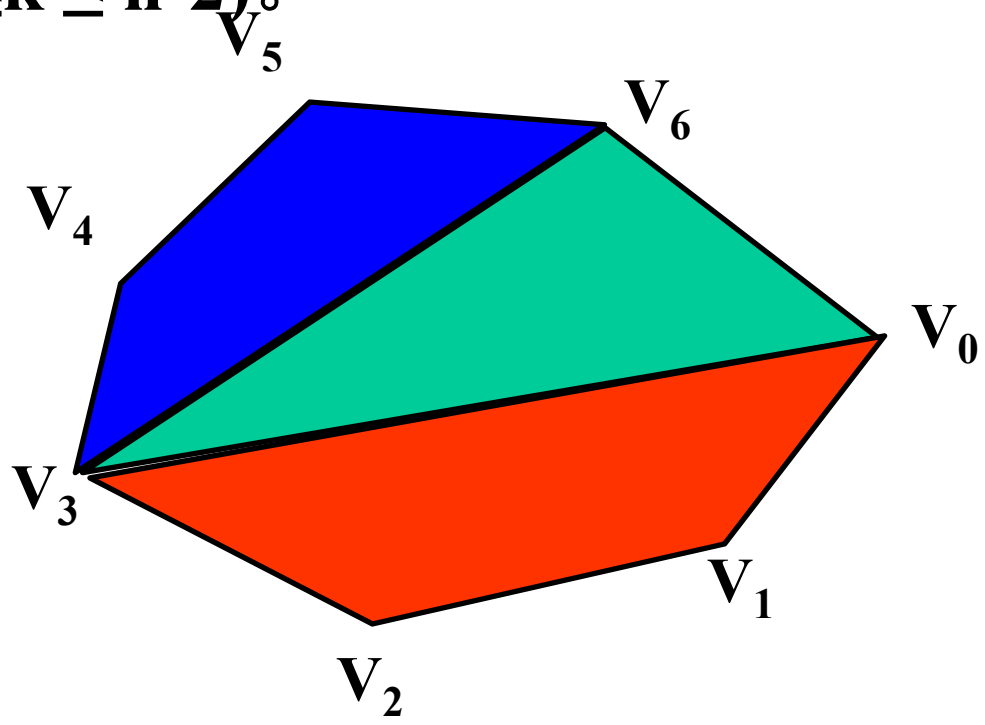
因为若不然,一定存在以V_iV_j为边界的某个区域没有被三角剖分。



- 顶点序列$V_0, V_1, \ldots, V_{n-1}$确定的凸n边形的最小三角剖分

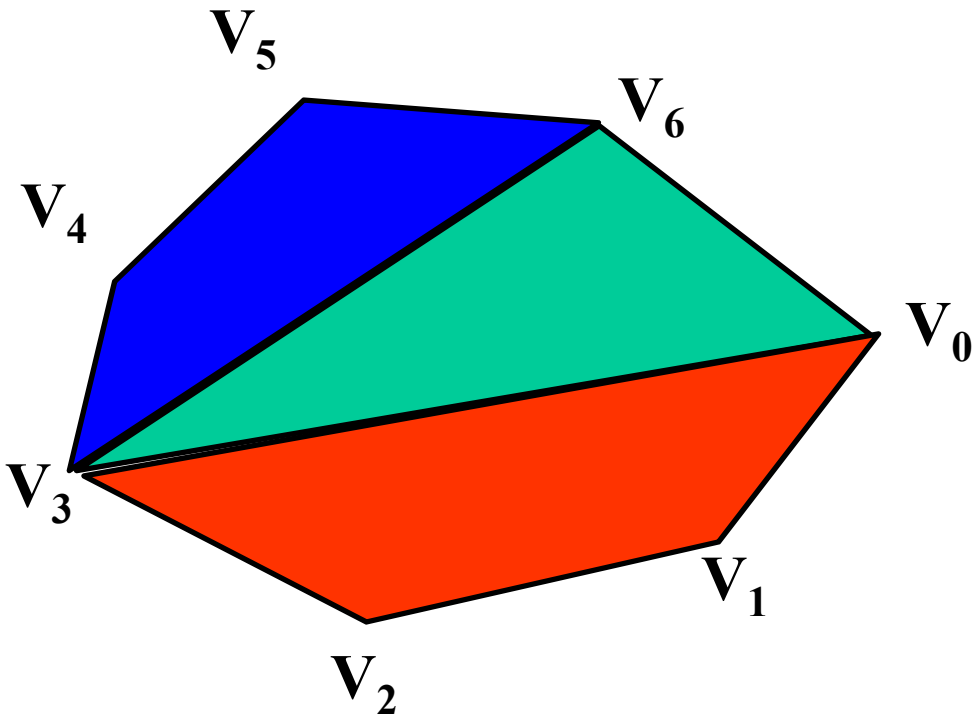
挑选两个相邻顶点V_0和V_{n-1},由事实3知道在最小三角剖分中,必有另一顶点V_k,或者使V_0V_k是对角线,或者使$V_{n-1}V_k$是对角线。

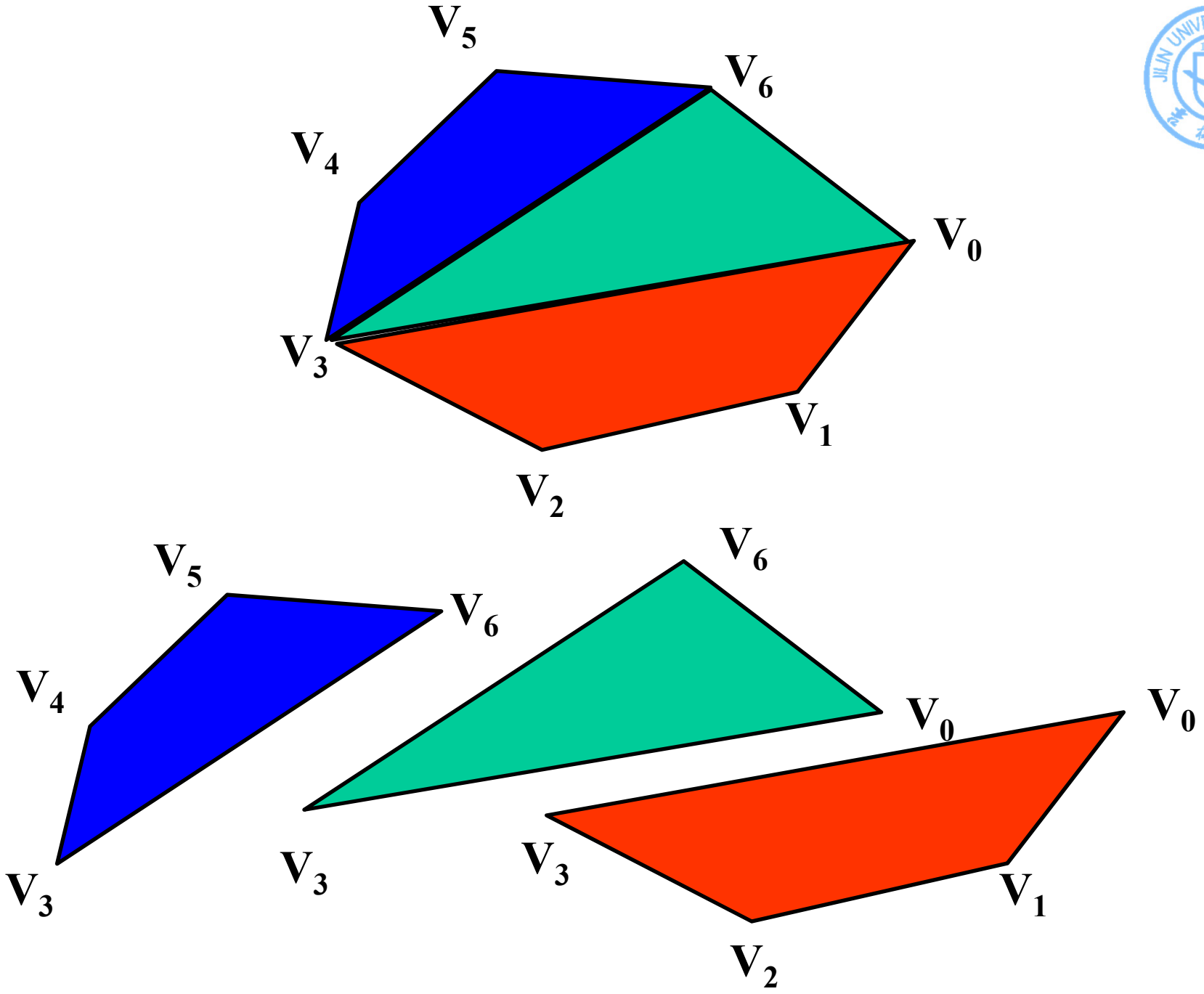
对有n(如n=7)个顶点的多边形，V_k的选取方法有n-2种($1 \le k \le n-2$)。



对于每个可能的V_k用对角线V_0V_k(或V_6V_k)把原多边形剖分成两个较小的多边形,这样原问题就被分成为两个子问题。

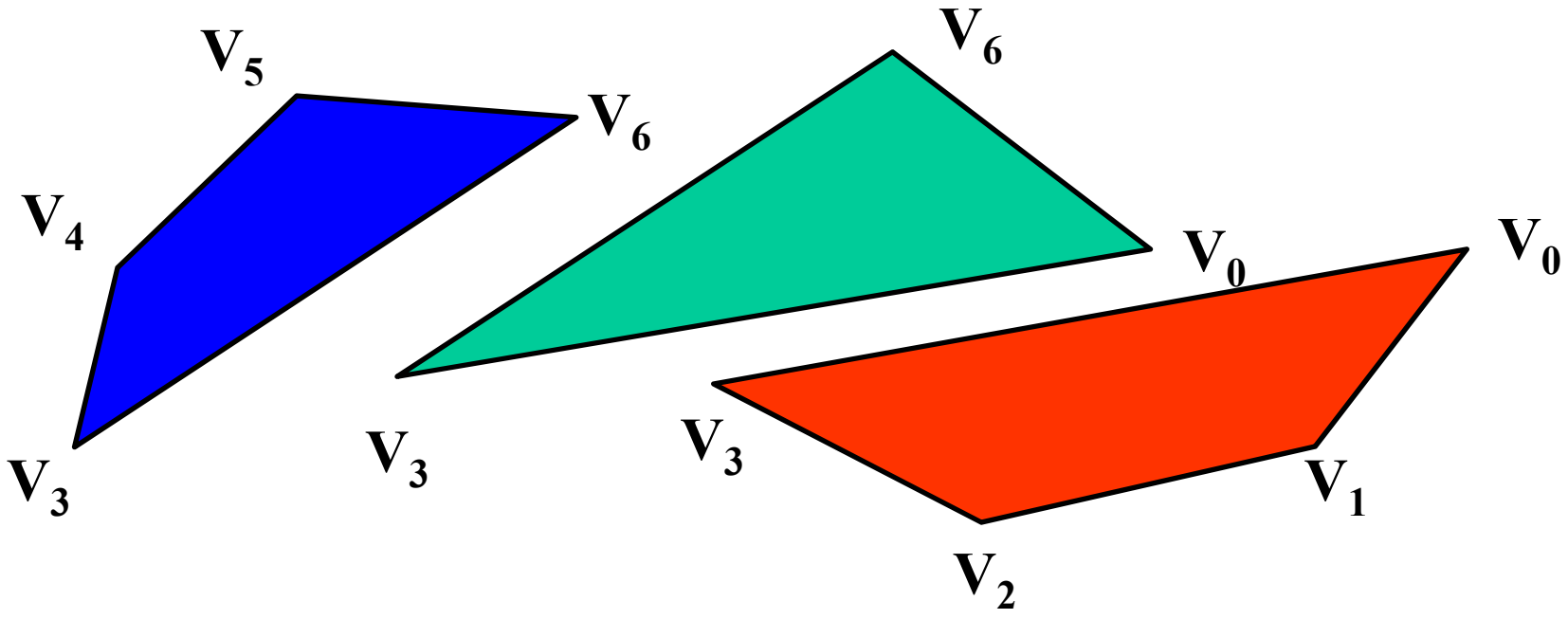
往下需要寻找分成的两个较小凸多边形的最小三角剖分。（递归）




V5
V6
V4
V0
V3
V1
V2
V5
V6
V4
V3
V6
V0
V3
V0
V3
V1
V2

选择剖分方法,使得每次剖分后所得的子问题只涉及一条原多边形的对角线。

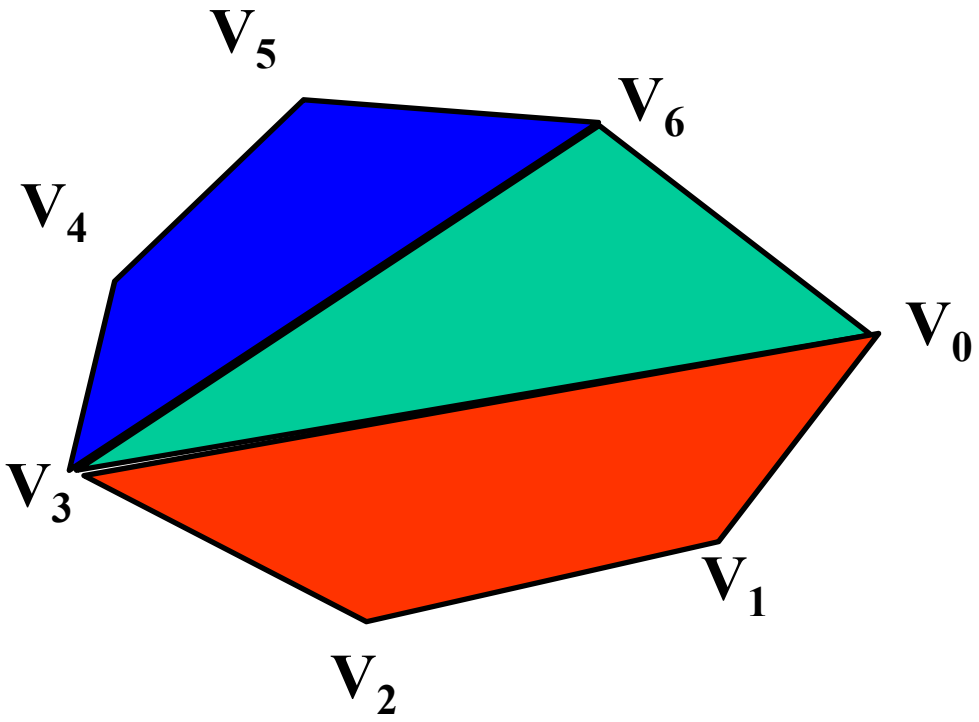
由事实4知在最小剖分中的对角线一定与另外一点构成三角形。



引人记号S_{is},表示一个子多边形$V_i,V_{i+1},\cdots,V_{i+s-1}$的最小三角剖分问题。子多边形由$V_i$开始的S个顶点按顺时针向排列围成。（图中$S_{04}$,S_{34}）

解S_{is},必须考虑如下三种情况:

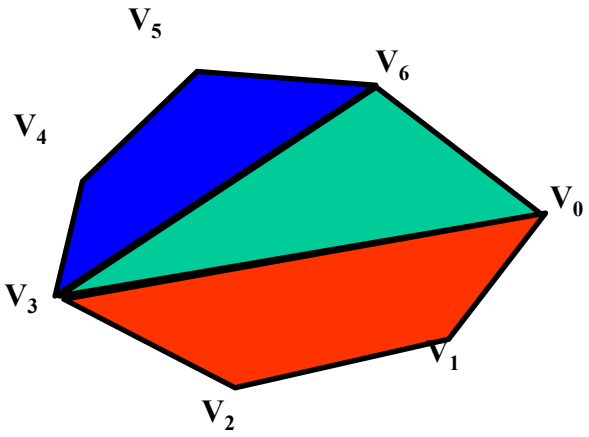
对$1\le k\le S-2$,选择V_{i+k}(例如k=3,s=7)这时构成一个三角形$V_iV_{i+k}V_{i+S-1}$,得到两个子问题$S_{i,k+1}$和$S_{i+k,S-k}$。

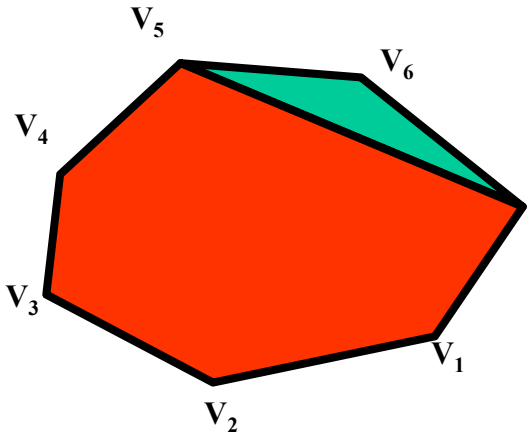


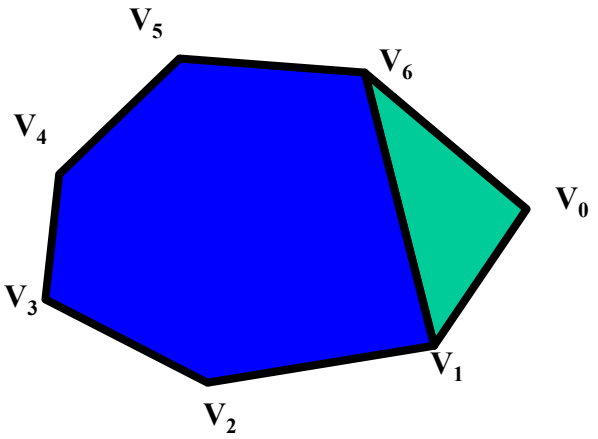
对$1 \le k \le S-2$,选择V_{i+k}(例如k=4)这时构成一个三角形$V_iV_{i+k}V_{i+S-1}$,得到两个子问题$S_{i,k+1}$和$S_{i+k,S-k}$。

1.当k=S-2时，选择顶点V_{i+S-2},这时构成一个三角形$V_iV_{i+S-2}V_{i+S-1}$,得到一个子问题$S_{i,S-1}$

2.当k=1时，选择顶点V_{i+1},这时构成一个三角形$V_iV_{i+1}V_{i+S-1}$,得到一个子问题$S_{i+1,S-1}$。







C_{iS}记为子问题S_{iS}的解

C_{iS}的公式如下:

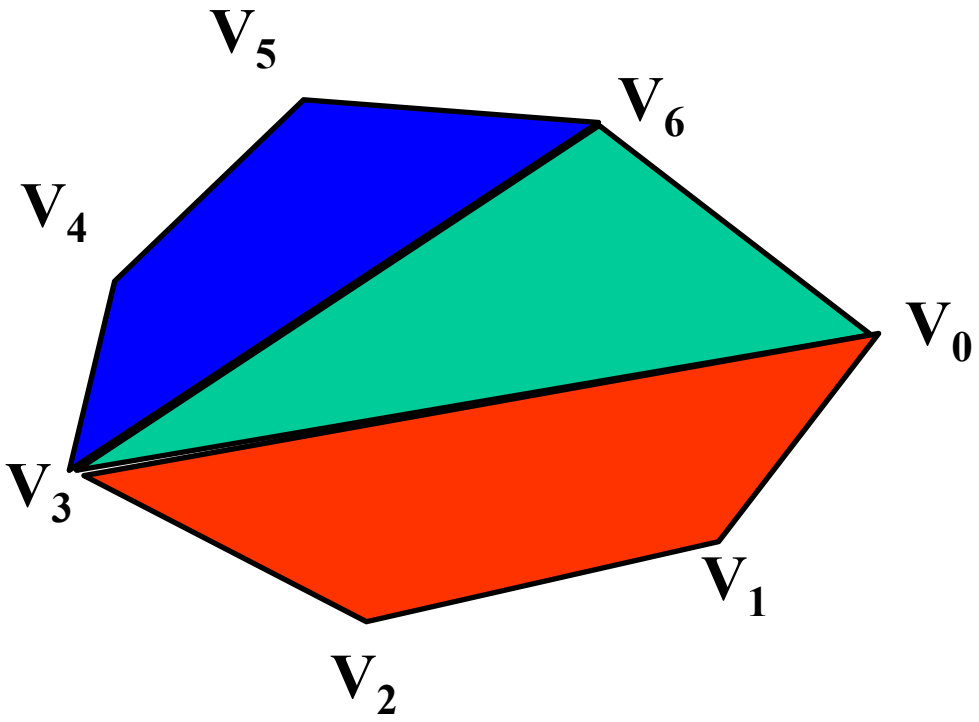
$$C_{iS}=\min[C_{i,k+1}+C_{i+k,S-k}+D(V_iV_{i+k})+D(V_{i+k}V_{i+S-1})]$$

$$1\le k\le S-2$$

若V_pV_q是对角线,则$D(V_pV_q)$是它的长度;

若V_pV_q是原多边形的边,则$D(V_pV_q)=0$;

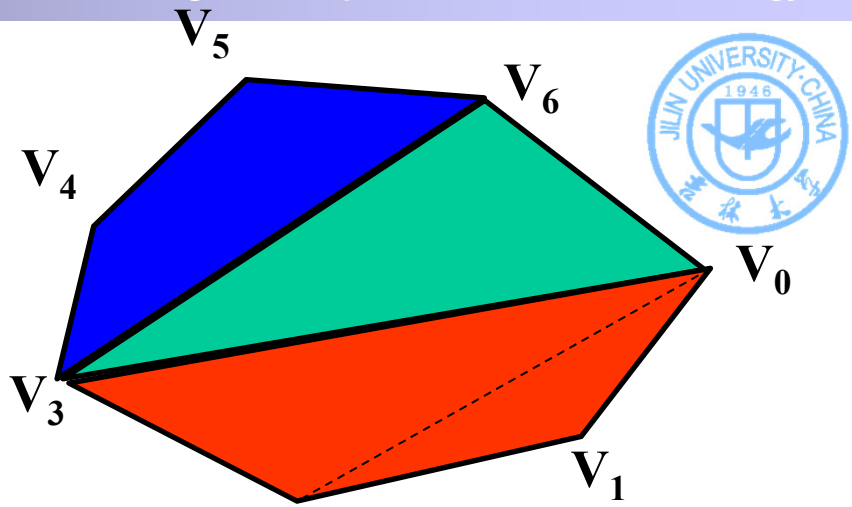
若$S<4$,则$C_{iS}=0$。这因为C_{iS}是最小三角剖分中所有对角线的总长度,原多边形的边不是对角线,当S<4时,无对角线。



C_{04}的计算：i=0,s=4　1<=k<=2

k=1　$C_{02}+C_{13}+D(V_0V_1)+D(V_1V_3)$

k=2　$C_{03}+C_{22}+D(V_0V_2)+D(V_2V_3)$



C_{07}

C_{06}　C_{16}

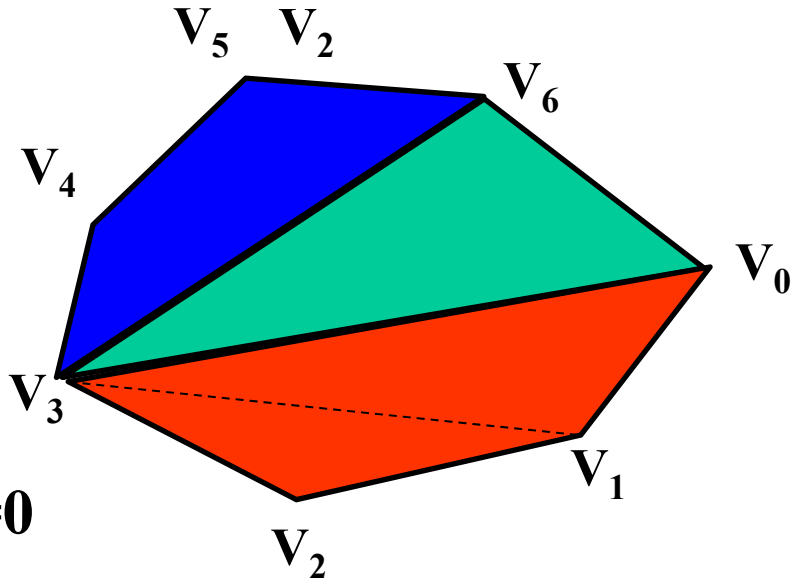
C_{05}　C_{15}　C_{25}

C_{04}　C_{14}　C_{24}　C_{34}

$C_{03}=0$　$C_{13}=0$　$C_{23}=0$　$C_{33}=0$　$C_{43}=0$

$C_{02}=0$　$C_{12}=0$　$C_{22}=0$　$C_{32}=0$　$C_{42}=0$　$C_{52}=0$

$C_{01}=0$　$C_{11}=0$　$C_{21}=0$　$C_{31}=0$　$C_{41}=0$　$C_{51}=0$　$C_{61}=0$



C_{25}的计算：i=2,s=5　1<=k<=3

k=1　$C_{22}+C_{34}+D(V_2V_3)+D(V_3V_6)$

k=2　$C_{23}+C_{43}+D(V_2V_4)+D(V_4V_6)$

k=3　$C_{24}+C_{52}+D(V_2V_5)+D(V_5V_6)$

C_{07}

C_{06}　C_{16}

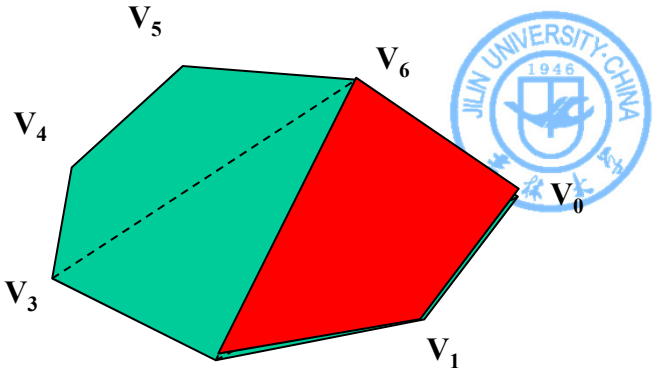
C_{05}　C_{15}　C_{25}

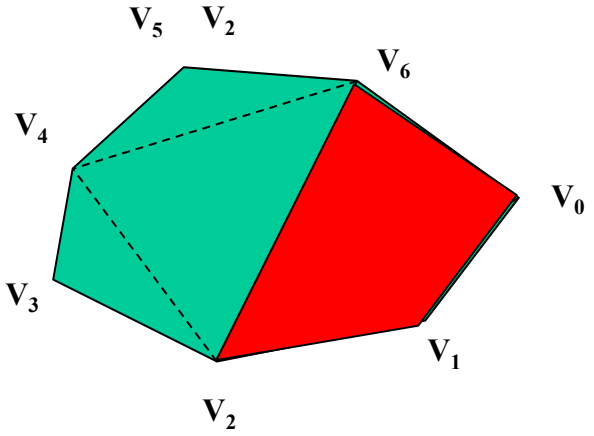
C_{04}　C_{14}　C_{24}　C_{34}

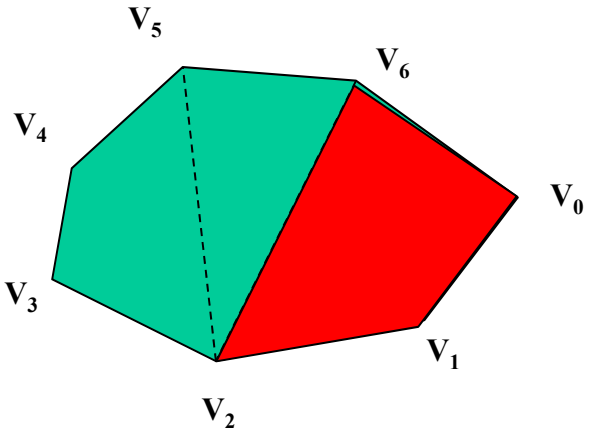
$C_{03}=0$　$C_{13}=0$　$C_{23}=0$　$C_{33}=0$　$C_{43}=0$

$C_{02}=0$　$C_{12}=0$　$C_{22}=0$　$C_{32}=0$　$C_{42}=0$　$C_{52}=0$

$C_{01}=0$　$C_{11}=0$　$C_{21}=0$　$C_{31}=0$　$C_{41}=0$　$C_{51}=0$　$C_{61}=0$







$$C_{is} = \min\{C_{i,k+1} + C_{i+k,s-k} + D(V_i V_{i+k}) + D(V_{i+k} V_{i+s-1})\}$$

C_{07}

$i = 0, s = 7, 1 \le k \le 5$

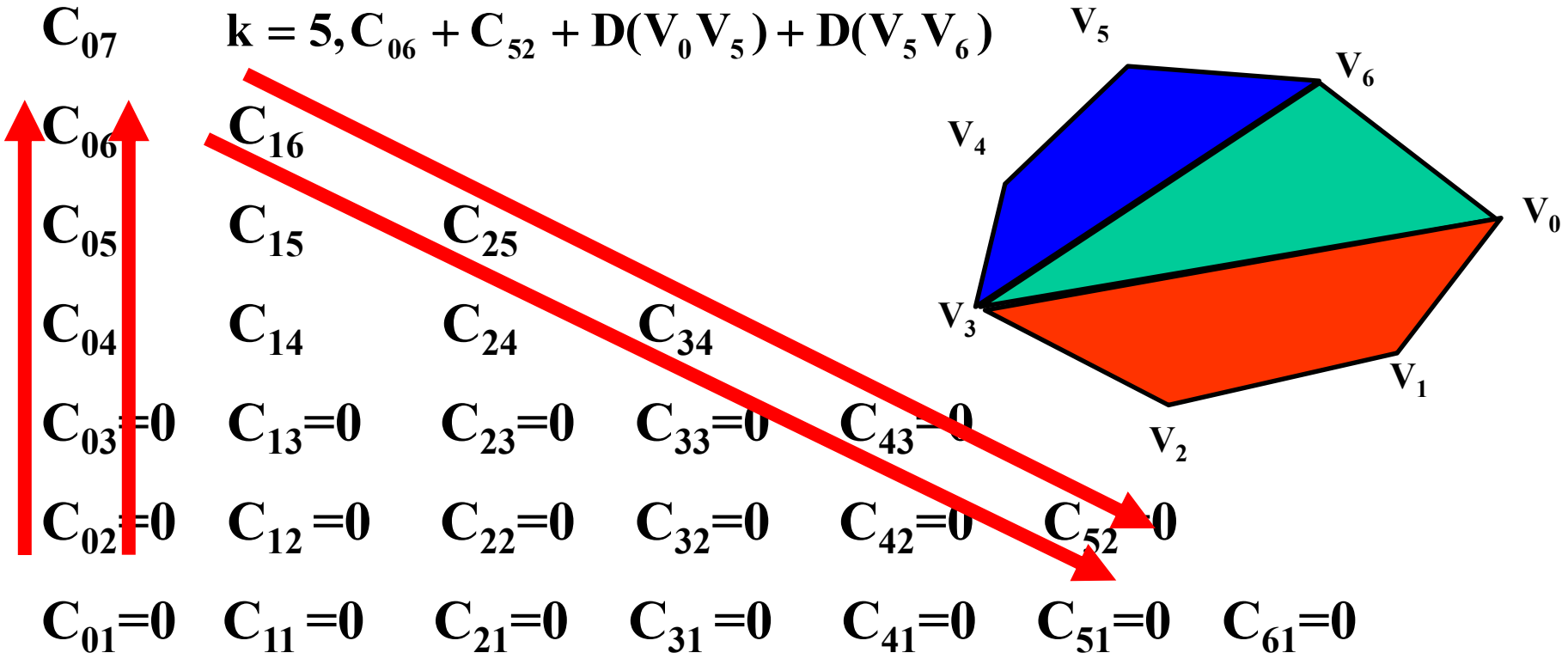
$k = 1, C_{02} + C_{16} + D(V_0 V_1) + D(V_1 V_6)$

$k = 2, C_{03} + C_{25} + D(V_0 V_2) + D(V_2 V_6)$

$k = 3, C_{04} + C_{34} + D(V_0 V_3) + D(V_3 V_6)$

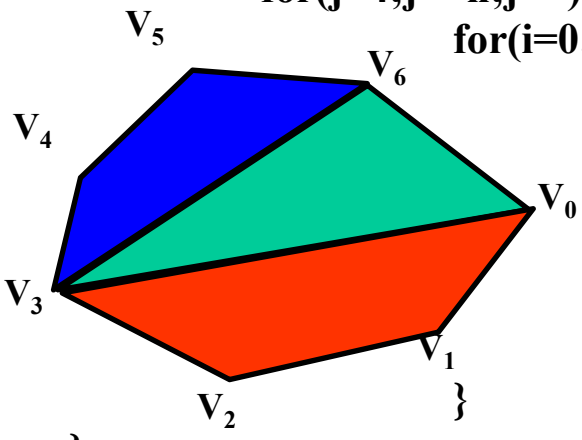
$k = 4, C_{05} + C_{43} + D(V_0 V_4) + D(V_4 V_6)$

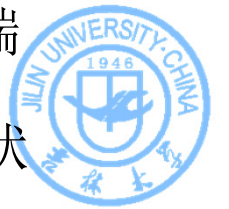
$k = 5, C_{06} + C_{52} + D(V_0 V_5) + D(V_5 V_6)$



最小三角剖分问题的求解适合采用动态规划方法。

```
void Min_Polygon_Triangulation(Point p[][2],int m ){ /* P存储多边形顶点坐标 m为顶点数 */
        double d[m][m],cc,ccc;//d数组存储对角线长度
        double c[m+1][m+1];
        int i,j,k;int n=m;
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        if(i!=j&&(abs(i-j)!=1)&&(abs(i-j)!=n-1))
                                d[i][j]=sqrt((p[i][0]-p[j][0])*(p[i][0]-p[j][0])
                                        +(p[i][1]-p[j][1])*(p[i][1]-p[j][1]));
                        else d[i][j]=0.0;       //计算对角线长度
        for(j=0;j<=3;j++)
                for(i=0;i<=n-j;i++)
                        c[i][j]=0.0;//j小于3的C_ij赋值成0
        for(j=4;j<=n;j++)       //计算j大于3到n-1的C_ij
                for(i=0;i<=n-j;i++)   {
                        c[i][j]=maxdouble;    //赋值最大双精度数
                        for(k=1;k<=j-2;k++) {
                                cc=c[i][(k+1)%n]+c[(i+k)%n][j-k]
                                        +d[i][(i+k)%n]+d[(i+k)%n][(i+j-1)%n];
                                if(c[i][j]>cc) c[i][j]=cc;
                        }
                }
}
```





- 1.〔事件点进度表E初始化〕将输入待求交点的n条线段的2n个端点按x,y字典式排序后存放于表E中;
- 2.〔准备收集交点〕A←null;{A是一集合,初为空,准备存入上一状态下的L;}
- 3.〔平面扫描〕若表E不为空,则进行3.1~3.3循环。直到表E为空时算法结束。
- 3.1〔取出当前事件点〕P←MIN(E); E←E- P;
- 3.2〔当前事件点处理〕考查当前事件点P,分三种情况:
- (1) 若P是边S的左端点,则做：
- 更新L;
- 如果A==L则无交点，否则有交点， 返回true;
- INSERT(S,L);
- (2) 若P是边S的右端点,则做：
- 更新L;
- 如果A==L则无交点，否则有交点，返回true;
- DELETE(S,L);
- 3.3 A←L
- 4返回false.

- $A_1=y_2-y_1, B_1=x_1-x_2, C_1=x_2y_1-x_1y_2, D_1=A_1x_1+B_1y_1+C_1$, $D_2=A_1x_2+B_1y_2+C_1$,
- $A_2=y_4-y_3, B_2=x_3-x_4, C_2=x_4y_3-x_3y_4, D_3=A_2x_3+B_2y_3+C_3$, $D_4=A_2x_4+B_2y_4+C_4$,
- 如果$D_1 \bullet D_2 \leq 0$且$D_3 \bullet D_4 \leq 0$，则两条线段相交。