中国计算机学会
China Computer Federation

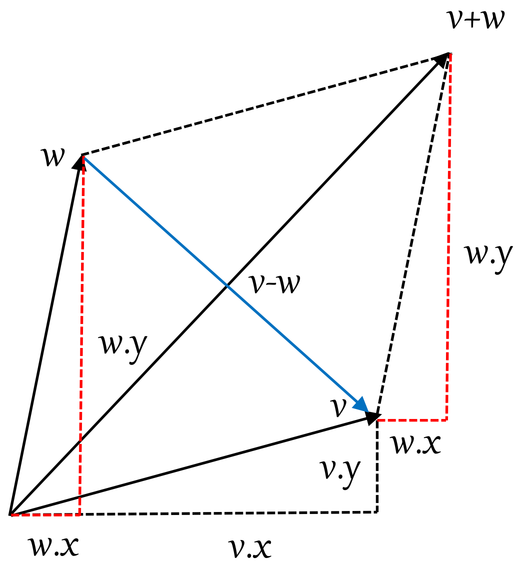
计算几何及其应用

中山纪念中学 宋新波

二维几何基础

□向量

- 有大小和方向的量。如速度、位移等物理量都是向量。
- 平面坐标系中，向量和点一样，用两个数x和y表示。
- 向量等于向量的起点到终点的位移。
- 也相当于把起点平移到原点后终点的坐标。
- 向量的加法满足平行四边形法则，减法同样如此。如图。
- 注意不能混淆点和向量。
- 如，点-点=向量，向量+向量=向量，点加向量=点，点+点没有意义。



中国计算机学会 China Computer Federation

常用定义

```
struct Point{
  double x,y;
  Point(double x=0,double y=0):x(x),y(y){ }
};
typedef Point3 Vector;
```

➢ 向量+向量=向量，点+向量=点

```
Vector operator + (Vector A , Vector B){return Vector(A.x+B.x,A.y+B.y);}
```

➢ 点-点=向量

```
Vector operator - (Point A , Point B){return Vector(A.x-B.x,A.y-B.y);}
```

➢ 向量*数=向量

```
Vector operator * (Vector A , double p){return Vector(A.x*p,A.y*p);}
```

➢ 向量/数=向量

➢ Vector operator / (Vector A , double p){return Vector(A.x/p,A.y/p);}

```
bool operator < (const Point& a , const Point& b)
{
    return a.x<b.x || a.x==b.x && a.y<b.y;
}
const double eps=1e-10;
int dcmp(double x)//三态函数，减少精度问题
{
    if(fabs(x)<eps)return 0;else return x<0?-1:1;
}
bool operator == (const Point& a , const Point& b){
    return dcmp(a.x-b.x)==0 && dcmp(a.y-b.y)==0;
}
```

基本证明及运算

口差角余弦公式$\cos(\boldsymbol{\alpha}-\boldsymbol{\beta})=\cos(\boldsymbol{\alpha})\cos(\boldsymbol{\beta})+\sin(\boldsymbol{\alpha})\sin(\boldsymbol{\beta})$

➢**证明：**

单位圆中∠AOX=$\boldsymbol{\alpha}$， ∠AOX=$\boldsymbol{\beta}$,BC⊥OA于C点，

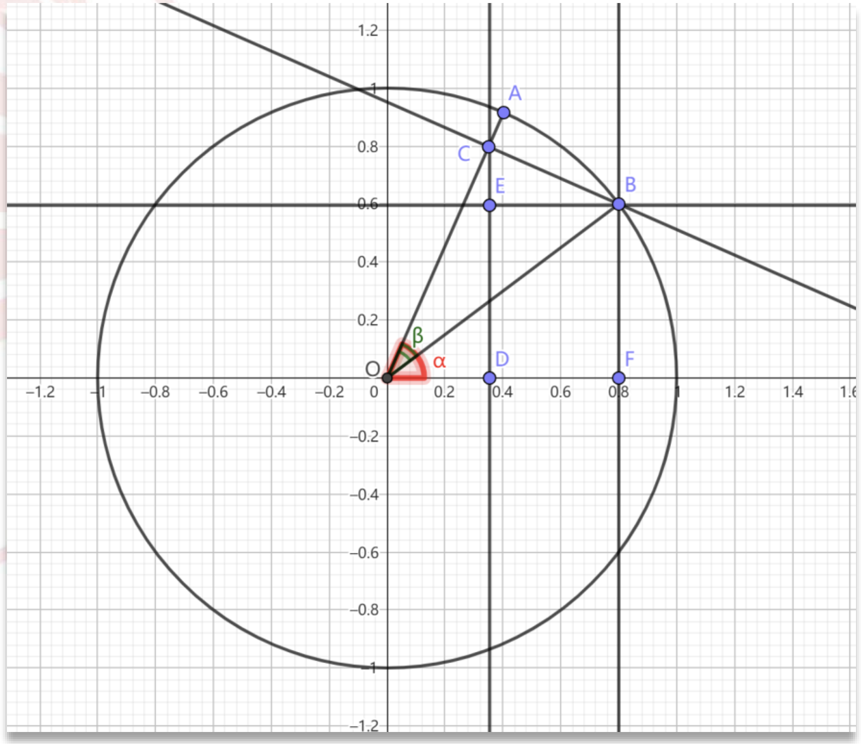
CD垂直于x轴于D点,BE⊥CD于E点,BF垂直于x轴于F点

则有∠BCD=$\boldsymbol{\alpha}$. OC= $\cos(\boldsymbol{\beta})$,OD=OC* $\cos(\boldsymbol{\alpha})$= $\cos(\boldsymbol{\alpha})\cos(\boldsymbol{\beta})$

OF= $\cos(\boldsymbol{\alpha}-\boldsymbol{\beta})$,BC= $\sin(\boldsymbol{\beta})$ ， DF=BE=BC* $\sin(\boldsymbol{\alpha})$ = $\sin(\boldsymbol{\alpha})\sin(\boldsymbol{\beta})$

OD=OF−DF= $\cos(\boldsymbol{\alpha}-\boldsymbol{\beta})$− $\sin(\boldsymbol{\alpha})\sin(\boldsymbol{\beta})$= $\cos(\boldsymbol{\alpha})\cos(\boldsymbol{\beta})$

所以： $\cos(\boldsymbol{\alpha}-\boldsymbol{\beta})=\cos(\boldsymbol{\alpha})\cos(\boldsymbol{\beta})+\sin(\boldsymbol{\alpha})\sin(\boldsymbol{\beta})$



基本证明及运算

■余弦定理$c^2 = a^2 + b^2 - 2abcos(\alpha)$

➢**证明：**

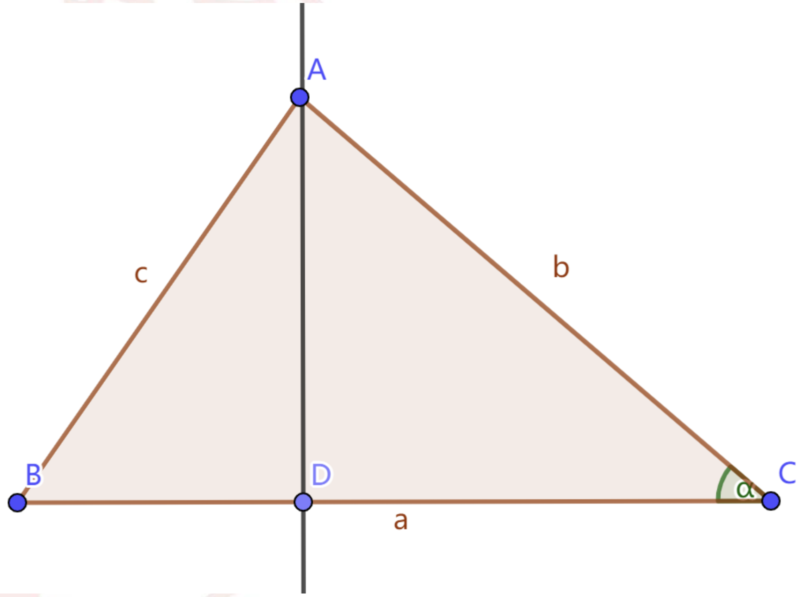
$AD = b * sin(\alpha)\,, CD = b * cos(\alpha)$

$BD = BC - CD = a - b * cos(\alpha)$

由勾股定理得：$AB^2 = AD^2 + BD^2$

即$c^2 = [bsin(\alpha)]^2 + [a - bcos(\alpha)]^2$

$c^2 = a^2 + b^2 - 2abcos(\alpha)$得证！



基本证明及运算

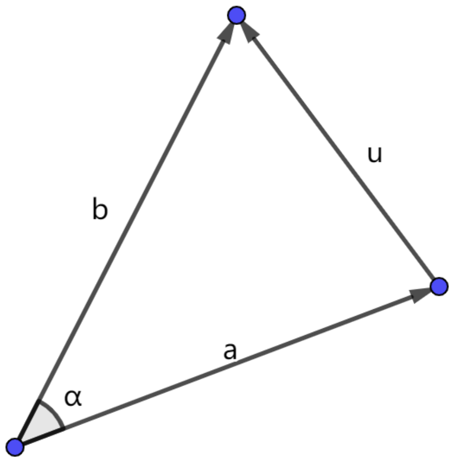
□向量的点积 $a \cdot b = |a|\,|b|cos(\alpha) = a.x * b.x + a.y * b.y$

(夹角α是指从a到b逆时针旋转的角)

➢证明：方法一

$|a| = \sqrt{(a.x)^2 + (a.y)^2}$， $|b| = \sqrt{(b.x)^2 + (b.y)^2}$

$|u| = |b - a| = \sqrt{(b.x - a.x)^2 + (b.y - a.y)^2}$



根据余弦定理有：

$a \cdot b = |a|\,|b|cos(\alpha) = (|a|^2 + |b|^2 - |u|^2)/2$

$= [(a.x)^2 + (a.y)^2 + (b.x)^2 + (b.y)^2 - (b.x - a.x)^2 - (b.y - a.y)^2]/2$

$= \ a.x * b.x + a.y * b.y$ 得证！

基本证明及运算

□向量的点积 $a \cdot b = |a|\,|b|cos(\alpha) = a.x * b.x + a.y * b.y$

(夹角α是指从a到b逆时针旋转的角)

➢证明：方法二，利用差角余弦公式。

把向量a,b的起点均移至原点，则有

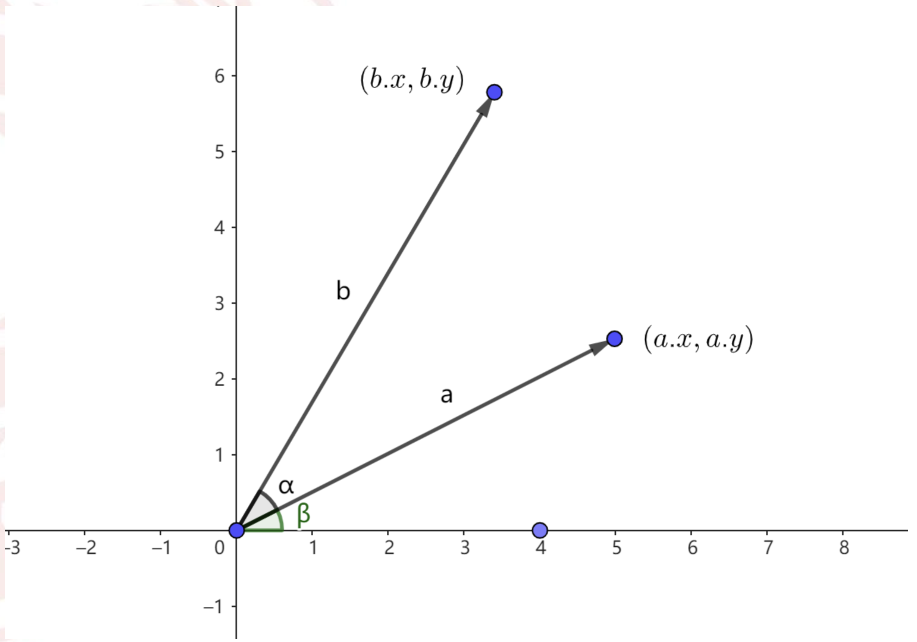
$a.x = |a|cos(\beta)$，$a.y = |a|sin(\beta)$

$b.x = |b|cos(\alpha + \beta), b.y = |b|sin(\alpha + \beta)$

$|a|\,|b|cos(\alpha) = |a|\,|b|cos(\alpha + \beta - \beta)$
$= |a|\,|b|cos(\alpha + \beta)cos(\beta) + |a|\,|b|sin(\alpha + \beta)sin(\beta)$

$= \ a.x*b.x + a.y * b.y$

得证！



➢余弦是偶函数，向量的点积满足交换律。

基本证明及运算

□向量的叉积 $a * b = |a|\ |b| sin(\alpha) = a.x * b.y - a.y * b.y$

(夹角α是指从a到b逆时针旋转的角)

➢证明：利用差角正弦公式。

把向量a,b的起点均移至原点，则有

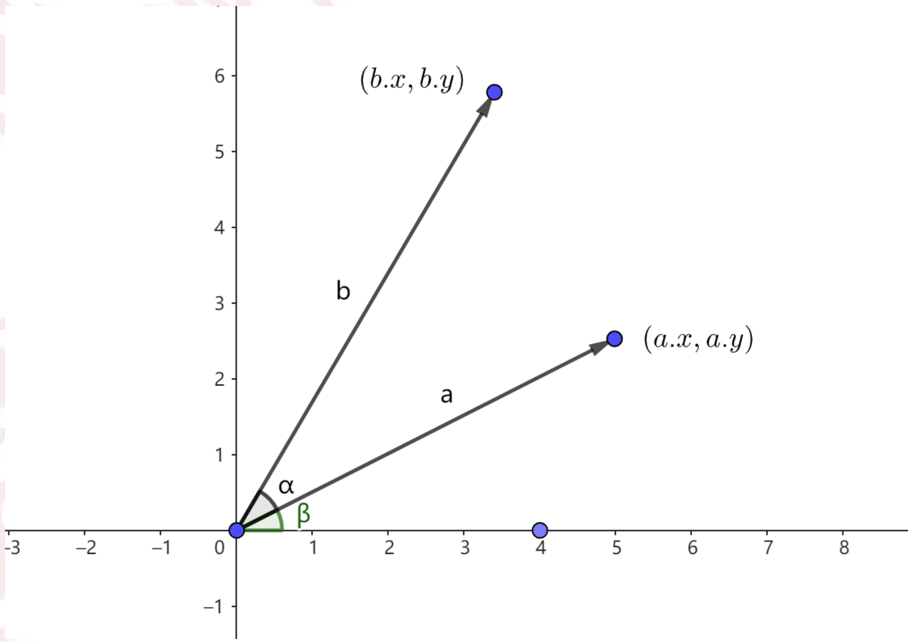
$a.x = |a|cos(\beta)$，$a.y = |a|sin(\beta)$

$b.x = |b|cos(\alpha+\beta), b.y = |b|sin(\alpha+\beta)$

$|a|\ |b|sin(\alpha) = |a|\ |b|sin(\alpha+\beta-\beta)$
$= |a|\ |b|sin(\alpha+\beta)cos(\beta) - |a|\ |b|cos(\alpha+\beta)sin(\beta)$

$= \ a.x * b.y - a.y * b.x$

得证！

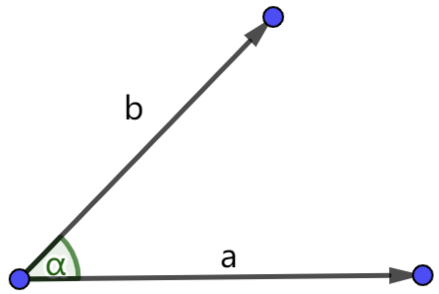


基本证明及运算

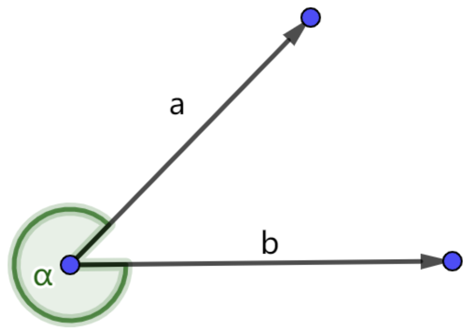
□向量的叉积 $a * b = |a|\,|b| sin(\alpha) = a.x * b.y - a.y * b.y$

(夹角α是指从a到b逆时针旋转的角)

- **根据以上结论有：**
  - ✓ 两个向量a和b的叉积等于a和b组成的三角形的有向面积的两倍
- **有向面积是指：**
  - ✓ 当两个向量共线时，叉积为0
  - ✓ 当夹角α在$(0,\pi)$内时，叉积为正
  - ✓ 当夹角α在$(\pi, 2\pi)$内时，叉积为负
- **所以有：**$a * b$**=**$-b * a$



$a * b > 0$



$a * b < 0$

基本证明及运算

- 向量的点积

```
double Dot (Vector A , Vector B){return A.x*B.x+A.y*B.y;}
```

- 向量的长度

```
double Length (Vector A){return sqrt(Dot(A,A);}
```

- 向量夹角的弧度，范围为$[0,\pi]$

```
double Angle (Vector A , Vector B){
    return acos(Dot(A,B)/Length(A)/Length(B);
}
```

- 向量的叉积

```
double Cross (Vector A , Vector B){return A.x*B.y-A.y*B.x;}
```

- 三点A,B,C形成的三角形有向面积的2倍

```
Double(Point A,Point B,Point C){return Cross(B-A,C-A);}
```

基本证明及运算

- 向量a逆时针旋转β弧度后得到向量b，则

$$b.x = |a|\cos(\alpha+\beta) = |a|\cos(\alpha)\cos(\beta) - |a|\sin(\alpha)sin(\beta) = a.x * \cos(\beta) - a.y * sin(\beta)$$

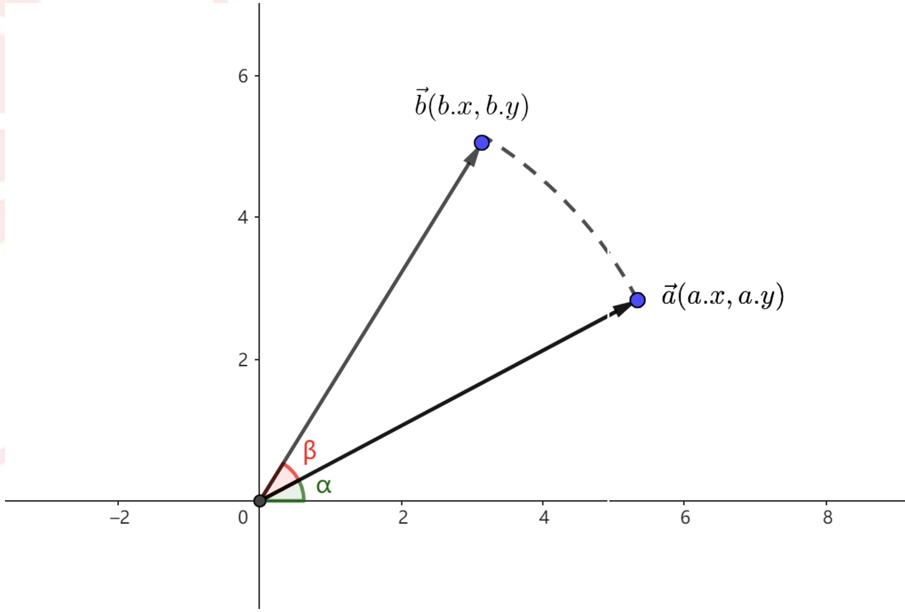
$$b.y = |a|sin(\alpha+\beta) = |a|sin(\alpha)\cos(\beta) + |a|\cos(\alpha)sin(\beta) = a.y * \cos(\beta) + a.x * sin(\beta)$$

- 向量旋转，rad是弧度

```
Vector Rotate(Vector A , double rad){
    return Vector(A.x*cos(rad)-A.y*sin(rad),A.y*cos(rad)+A.x*sin(rad);
}
```

- 计算向量的单位法向量（左转90°，再把长度归一化）

```
Vector Normal(Vector A){
    double L=Length(A);
    return Vector(-A.y/L,A.x/L);
}
```



点和直线

直线的参数表示

- 直线可以用直线上一点P_0和方向向量v表示。直线上所有点P满足$P=P_0+tv$，其中t称为参数。如已知直线上的两个不同点A和B，则方向向量为B-A，所以参数方程为$A+(B-A)t$。
- 直线的t没有范围限制，射线的$t>0$，线段的t在0-1之间。

直线交点

- 设直线分别为$P+tv$和$Q+tw$，交点在第一条直线的参数为t_1，在第二条直线上的参数为t_2。设向量$u=P-Q$，根据x和y坐标可以列出方程：

$$\begin{cases} P.x+t_1v.x=Q.x+t_2w.x \\ P.y+t_1v.y=Q.y+t_2w.y \end{cases}$$ 解方程得： $$\begin{cases} t_1=\dfrac{w.x*u.y-w.y*u.x}{v.x*w.y-v.y*w.x}=\dfrac{Cross(w,u)}{Cross(v,w)} \\ t_2=\dfrac{v.x*u.y-v.y*u.x}{v.x*w.y-v.y*w.x}=\dfrac{Cross(v,u)}{Cross(v,w)} \end{cases}$$

点和直线

□直线交点代码

//调用前确保两条直线P+tv和Q+tw有唯一交点。即当Cross(v,w)不等于0，即两天直线不平行

```
Point GetLineIntersection(Point P , Vector v , Point Q , Vector w)
{
    Vector u=P-Q;
    double t=Cross(w,u)/Cross(v,w);
    return P+v*t;
}
```

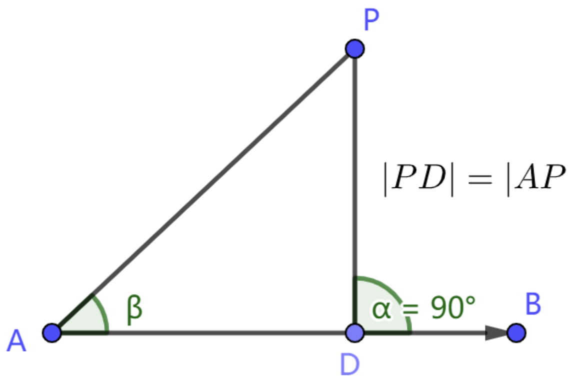
中国计算机学会
China Computer Federation

点和直线

□点到直线的距离

//可以用叉积算出平行四边形的面积，再除以底。如图。

```
double DistanceToLine(Point P，Point A , Point B)
{
    Vector v1=B-A,v2=P-A;
     return fabs(Cross(v1,v2)/Length(v1));
}
```

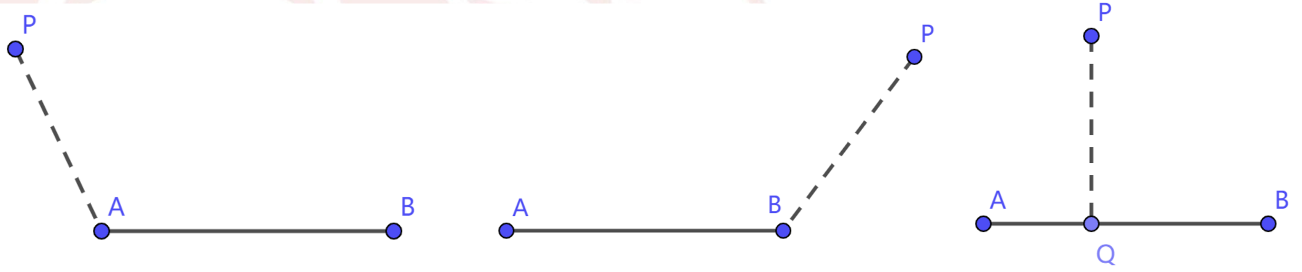


$$|PD| = |AP|sin(\beta) = \frac{fabs(Cross(B-A, P-A))}{|AB|}$$

点和直线

□点到线段的距离

```
double DistanceToSegment(Point P， Point A , Point B)
{
    if(A==B)return Length(P-A);
    Vector v1=B-A;
    Vector v2=P-A;
    Vector v3=P-B;
    if(dcmp(Dot(v1,v2))<0)return Length(v2);//情况1
    else if(dcmp(Dot(v1,v3))>0)return  Length(v3);//情况2
    else return fabs(Cross(v1,v2)/Length(v1));//情况3.就是点P到直线AB的距离
}
```



情况1：答案 $= PA$　情况2：答案 $= PB$　情况3：答案 $= PQ$

点和直线

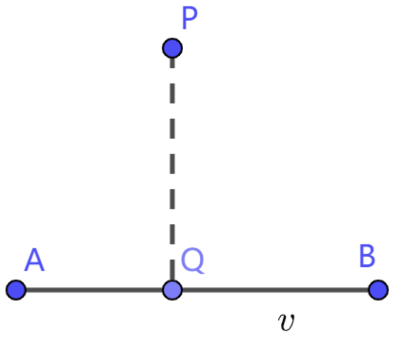
□点在直线上的投影

- ➢ 点积的分配律：Dot(A,B+C)=Dot(A,B)+Dot(A,C)
- ✓ 证明：B+C=(B.x+C.x,B.y+C.y)

Dot(A,B+C)=A.x*(B.x+C.x)+A.y*(B.y+C.y)

=(A.x*B.x+A.y*B.y)+(A.x*C.x+A.y*C.y)=Dot(A,B)+Dot(A,C)

- ➢ Q点坐标：$A+\overrightarrow{AB}*t$
- ➢ 根据$Dot(\overrightarrow{AB},\overrightarrow{PQ})$=Dot(v,P-A-v*t)=Dot(v,P-A)-t*Dot(v,v)=0，解得$t=\frac{Dot(v,P-A)}{Dot(v,v)}$



Q点坐标：$A+\vec{AB}*t$.利用$Dot(\vec{AB},\vec{PQ})=0$计算t.

```
Point GetLineProjection(Point P , Point A , Point B){
    Vector v=B-A;
    return A+v*Dot(v,P-A)/Dot(v,v);
}
```

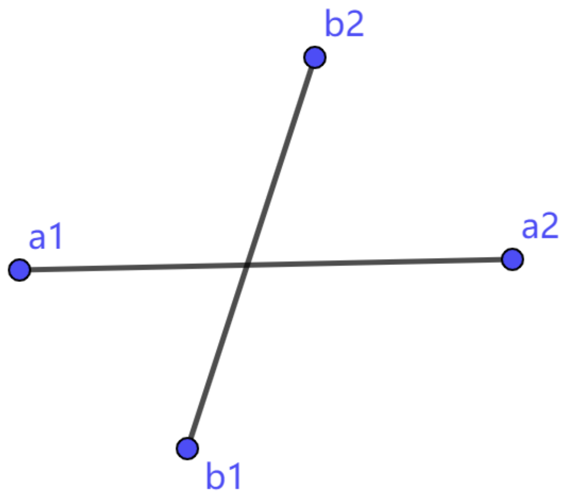
点和直线

□线段相交判定

➢定义“**规范相交**”为两条线段恰有一个公共点，且不在任何线段的端点。

➢“**规范相交**”的充要条件是：两条线段的两个端点都在另一条线段的两侧（利用叉积符号不同判断）

```
bool SegmentProperIntersection(Point a1 , Point a2 , Point b1 , Point b2){
   double c1=Cross(a2-a1,b1-a1),c2=Cross(a2-a1,b2-a1),
              c3=Cross(b2-b1,a1-b1),c4=Cross(b2-b1,a2-b1);
   return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}
```



多边形

□ 多边形

➢ 由在同一平面且不在同一直线上的三条或三条以上的线段首尾顺次连结且不相交所组成的封闭图形叫做多边形。

□ 凸多边形

➢ 指如果把一个多边形的所有边中，任意一条边向两方无限延长成为一直线时，其他各边都在此直线的同旁，那么这个多边形就叫做凸多边形。

➢ 其内角全不是优角

➢ 任意两个顶点间的线段位于多边形的内部或边上。

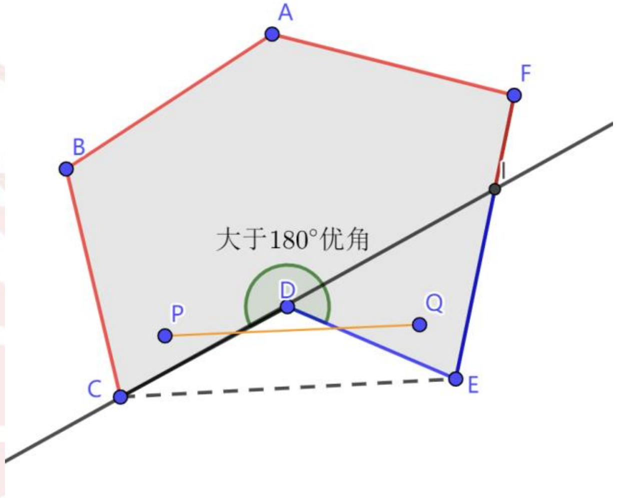
□ 凹多边形

➢ 指如果把一个多边形的所有边中，有一条边向两方无限延长成为一直线时，其他各边不都在此直线的同旁，那么这个多边形就叫做凹多边形。

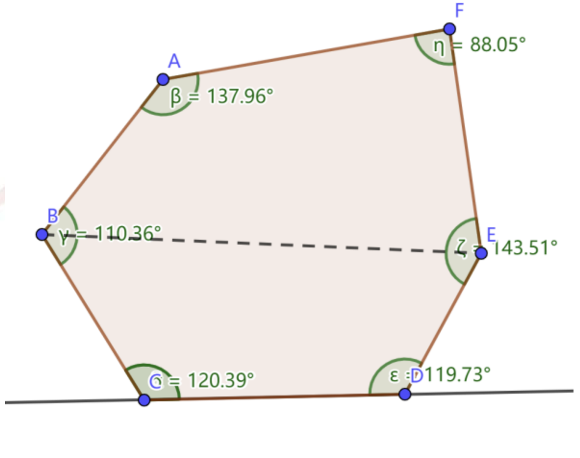
➢ 其内角中至少有一个优角

➢ 存在两个顶点间的线段位于多边形的外部

➢ 多边形内存在两个点，其连线不全部在多边形内部。



凹多边形

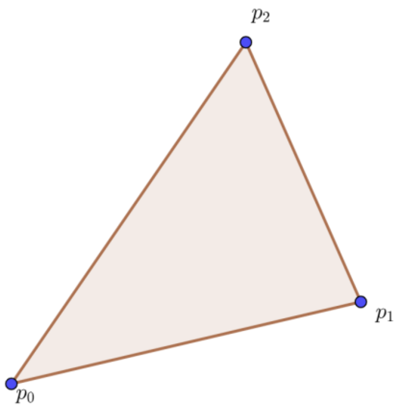


凸多边形

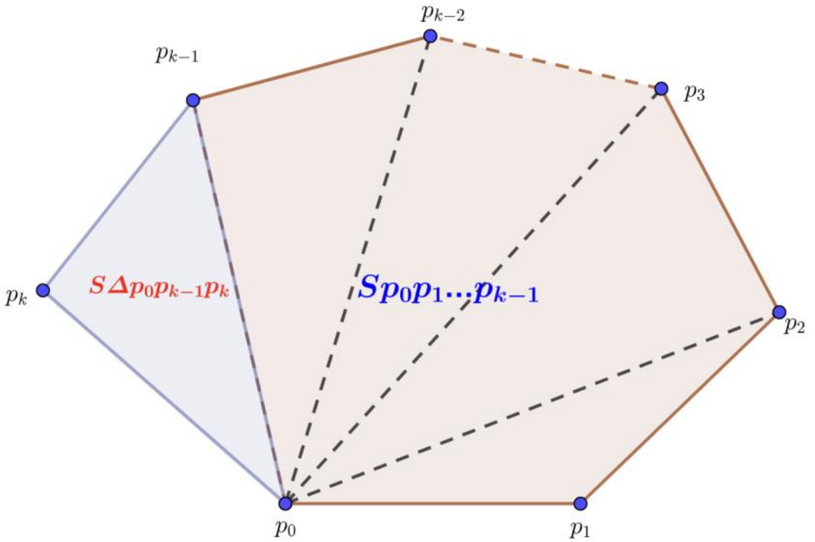
多边形

□凸多边形的有向面积

- ➢ 顶点已按顺序（顺时针或逆时针）给出，从第一个顶点出发把凸多边形分成n-2个三角形，然后把面积加起来。如右图。
- ➢ 证明：假设所有点按逆时针顺序给出。则有：
- ✓ 当n=3时成立。
- ✓ 设当n=k时成立。即通过此方法得到的是凸多边形$\boldsymbol{p_0}$ $\boldsymbol{p_1 \dots p_{k-1}}$的有向面积。当n=k+1时，由于是凸多边形，故点$\boldsymbol{p_k}$在向量$\overrightarrow{\boldsymbol{p_0 p_{k-1}}}$的左边，增加三角形$\boldsymbol{p_0 p_{k-1}\ p_k}$的面积后，总面积就是凸多边形$\boldsymbol{p_0\ p_1 \dots p_k}$的有向面积。结论对n=k+1也成立。
- ✓ 得证！



$n=3$时



$$S\Delta p_0p_{k-1}p_k + Sp_0p_1\dots p_{k-1} = Sp_0p_1\dots p_k$$

多边形

□ 凹多边形的有向面积

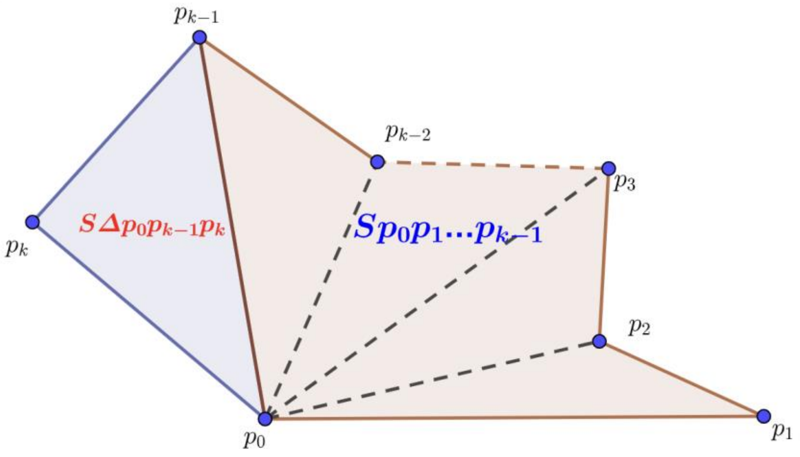
- ➢ 由于三角形面积是有向的，在外面的部分可以正负抵消，以上方法对凹多边形同样适用。
- ➢ 证明：假设点按逆时针顺序给出。则有：
- ✓ 当n=3时成立。
- ✓ 设当n=k时成立。即通过此方法得到的是凹多边形p_0 $p_1 \dots p_{k-1}$的有向面积。当n=k+1时，由于是凹多边形，分以下两种情况：

① $\mathrm{p_k}$在向量$\overrightarrow{p_0 p_{k-1}}$的左边，增加$S\Delta p_0 p_{k-1}\ p_k$后，总面积就是凹多边形$p_0\ p_1 \dots p_k$的有向面积。右图1.

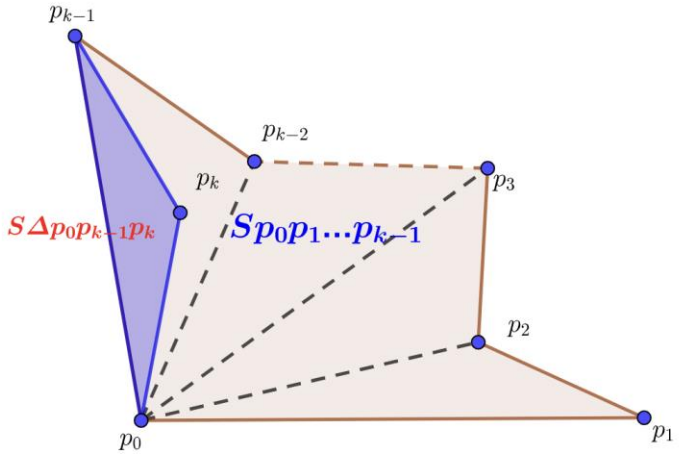
② $\mathrm{p_k}$在向量$\overrightarrow{p_0 p_{k-1}}$的右边，凹多边形$p_0\ p_1 \dots p_{k-1}$的面积相对凹多边形$p_0\ p_1 \dots p_k$多算了$S\Delta p_0 p_{k-1}\ p_k$，增加三角形$p_0 p_{k-1}\ p_k$的有向面积后，恰好把多算的部分抵消。右图2.

结论对n=k+1也成立。

- ✓ 得证！



$$S\Delta p_0 p_{k-1} p_k + S p_0 p_1 \dots p_{k-1} = S p_0 p_1 \dots p_k$$



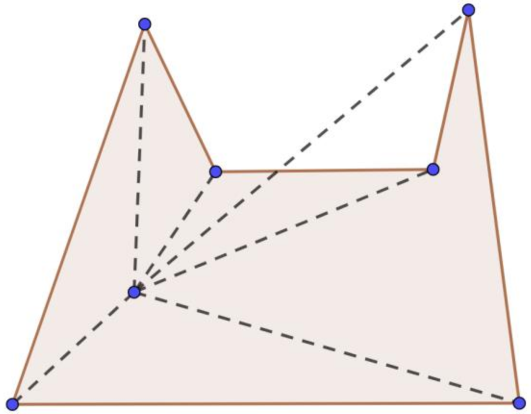
$$S\Delta p_0 p_{k-1} p_k - S p_0 p_1 \dots p_{k-1} = S p_0 p_1 \dots p_k$$

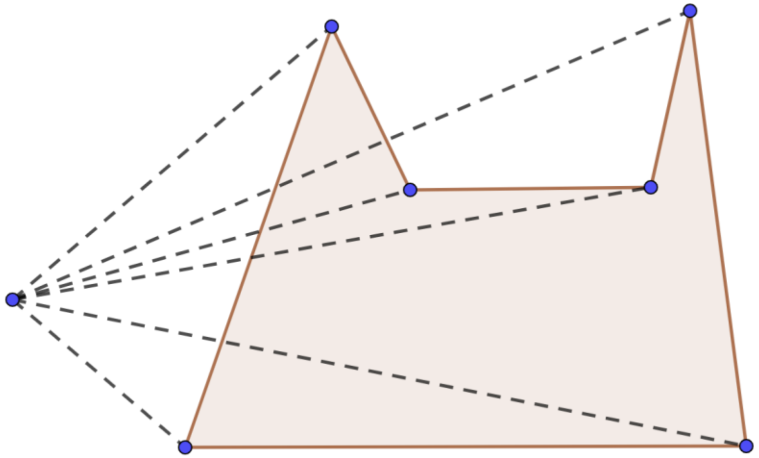
多边形

□ 多边形的有向面积代码

```
double PolygonArea(Point* p , int n){
    double area=0;
    for(int i=1;i<n-1;++i)
        area+=Cross(p[i]-p[0],p[i+1]-p[0]);
    return area/2;
}
```

- 可以从任意点出发进行划分

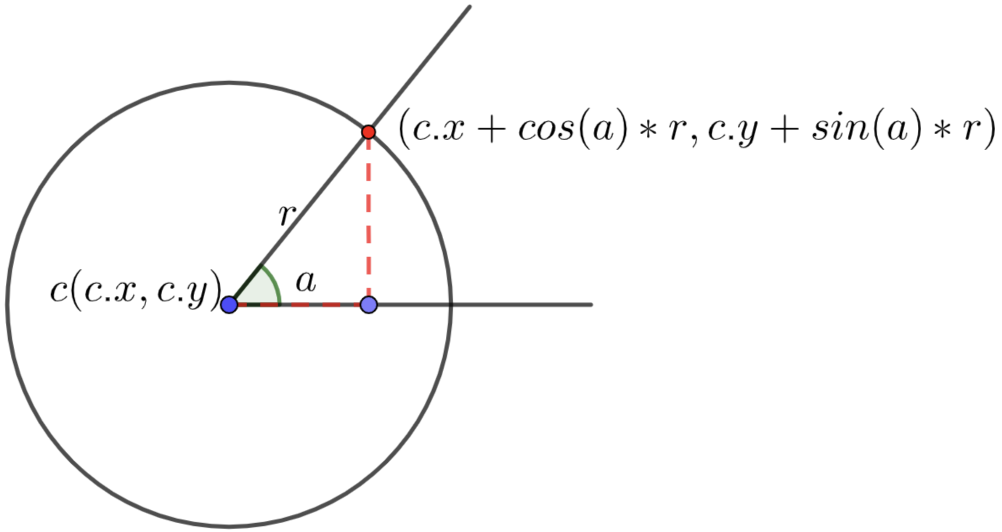




与圆有关的计算问题

□圆的定义

```
struct Circle{
    Point c;//圆心
    double r;//半径
    Circle(Point c , double r):c(c),r(r){}
    Point point(double a){//a表示圆心角的弧度
        return Point(c.x+cos(a)*r,c.y+sin(a)*r);
    }
}
```



直线与圆的交点

方法一：解方程组

- 假定直线为AB，圆的圆心为C，半径为r。设交点为P=A+t(B-A)，代入圆方程整理得到$(at+b)^2$+$(ct+d)^2$= r^2，进一步整理得一元二次方程$et^2+ft+g=0$。根据判别式可以区分无交点、一个交点和两个交点得情况。代码如下：

```
int  getLineCircleIntersection(Line L,Circle C,double& t1,double& t2,vector<point>& sol)
{
      double a=L.v.x , b=L.p.x-C.c.x , c=L.v.y , d=L.p.y-C.c.y;
      double e=a*a+c*c , f=2*(a*b+c*d) , g=b*b+d*d-C.r*C.r;
      double delta=f*f-4*e*g;  //判别式
      if (dcmp(delta)<0) return 0;//相离无交点
      if(dcmp(delta)==0){ //相切一个交点
            t1=t2=-f/(2*e);sol.push_back(L.point(t1));return 1;
      }
      t1=(-f-sqrt(delta))/(2*e);sol.push_back(L.point(t1));//相交
      t2=(-f+sqrt(delta))/(2*e);sol.push_back(L.point(t2));
      return 2;
}
```

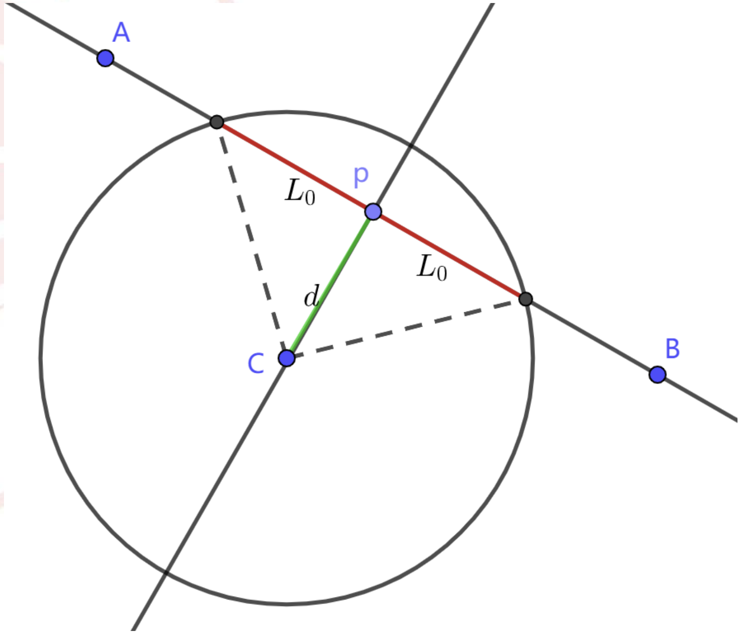
```
struct Line{
    Point p;
    Vector v;
    Line(Point p , Vector v):p(p),v(v){ }
    Point point(double t){
        return p+v*t;
    }
}
```

直线与圆的交点

▫方法二：几何法

➢ 假定直线为AB，圆的圆心为C。先求出C在AB上的投影p,再求向量AB对应的单位向量v，则两个交点分别为p-L0*v和p+L0*v,其中L0为p到交点的距离，可以由勾股定理算出。如图所示。代码如下：

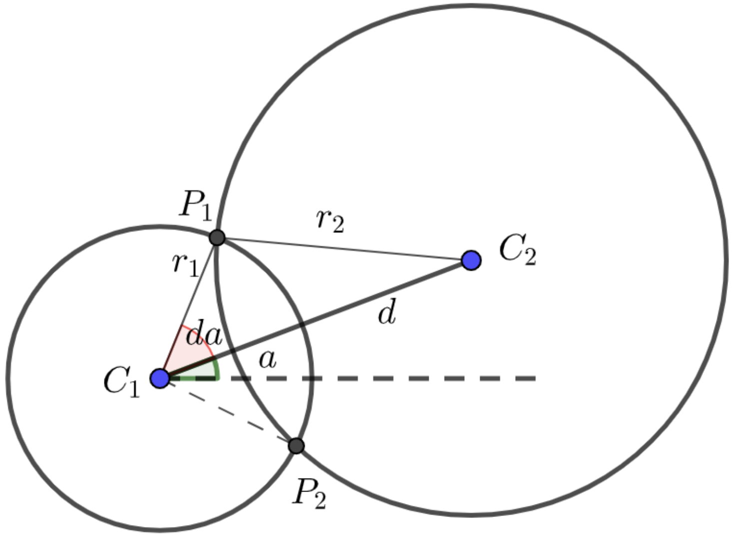
```
int  getLineCircleIntersection(Line L,Circle C,vector<point>& sol)
{
    double d=DistanceToLine(C.c,L.p,L.p+L.v);//圆心到直线的距离
    if(dcmp(d-C.r)>0)return 0;//相离无交点
    Point p=GetLineProjection(C.c,L.p,L.p+L.v);
    if(dcmp(d-C.r)==0){sol.push_back(p1);return 1;} //相切，1个交点
    double L0=sqrt(C.r*C.r-d*d);//投影到交点的距离
    Vector V=L.v/Length(L.v);//直线AB的单位向量
    sol.push_back(p+L0*V); sol.push_back(p-L0*V); //相交，2个交点
    return 2;
}
```



两个圆的交点

- 假定圆心分别为C_1和C_2，半径为r_1和r_2，圆心距为d，根据余弦定理可以算出C_1C_2到C_1P_1的角da，根据向量C_1C_2的极角a，加减da就可以得到C_1P_1和C_1P_2的极角。有了极角就可以计算出P_1和P_2的坐标。如图所示。代码如下：

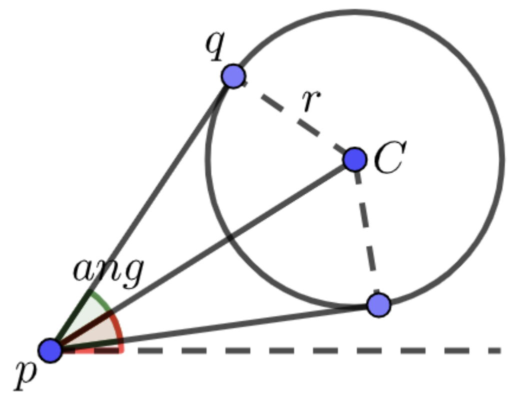
```
double angle(Vector v) {return atan2(v.y,v.x);}//返回向量v的极角，范围(-π,π]
int get CircleCircleIntersection(Circle C1,Circle C2,vector<Point>& sol){
    double d=Length(C1.c-C2.c);
    if(dcmp(d)==0){
        if(dcmp(C1.r-C2.r)==0)return -1;//两圆重合
        return 0;
    }
    if(dcmp(C1.r+C2.r-d)<0 || dcmp(fabs(C1.r-C2.r)-d)>0)return 0;//分离或内含
    double a=angle(C2.c-C1.C),da=acos((C1.r*C1.r+d*d-C2.r*C2.r)/(2*C1.r*d));
    Point p1=C1.point(a+da),p2=C1.point(a-da);
    sol.push_back(p1);
    if(p1==p2)return 1;
    sol.push_back(p2);return 2;
}
```



点到圆的切线

- 过定点作圆的切线。先求出pq与pc的夹角ang，则向量pc的极角加减ang就是两条切线的极角。注意切线不存在和只有一条的情况。

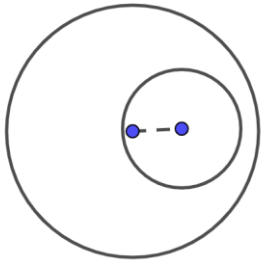
```
int  getTangents(Point p , Circle C , Vector* v){//v[i]是第i条切线的向量。返回切线条数。
      Vector u=C.c-p;
      double dis=Length(u);
      if(dis<C.r)return 0;
      else if(dcmp(dis-C.r)==0){v[0]=Rotate(u,PI/2);return 1;}
       double ang=asin(r/dis);
        v[0]=Rotate(u,ang);
        v[1]=Rotate(u,-ang);
        return 2;
}
```



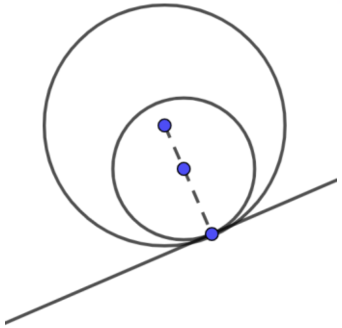
两个圆的共切线

□根据两圆的圆心距从小到大排列，共有6种情况

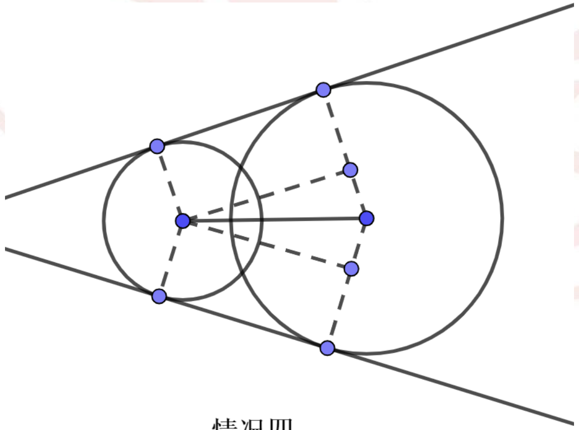
- 情况一：两圆完全重合。有无数条公切线。
- 情况二：两圆内含。没有公共点，没有公切线。
- 情况三：两圆内切。有1条外公切线。
- 情况四：两圆相交。有2条外公切线。
- 情况五：两圆外切。有3条外公切线，其中一条内公切线，两条外公切线。
- 情况六：两圆相离。有4条外公切线，其中两条内公切线，两条外公切线。



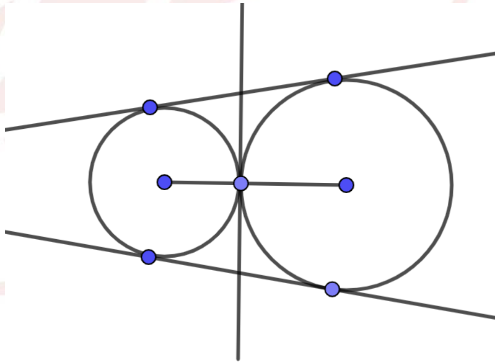
情况二



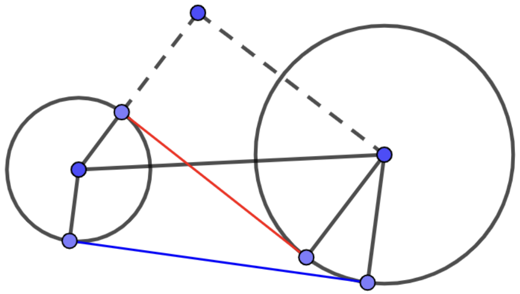
情况三



情况四



情况五

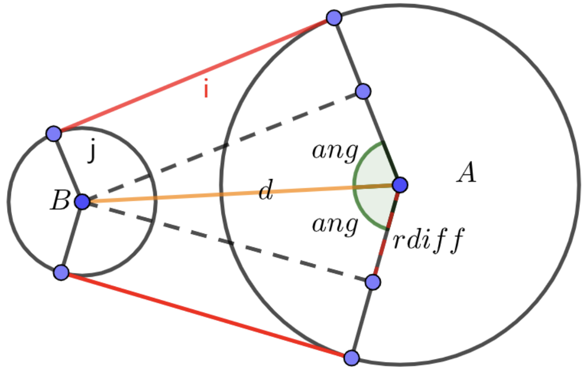


情况六

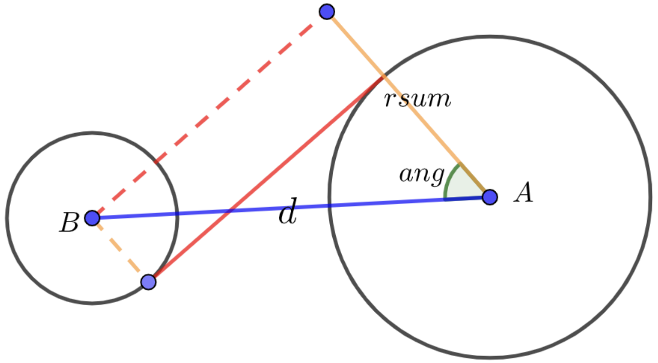
两个圆的共切线

□ 代码如下：

```
//返回切线条数。-1表示无穷条切线。a[i]和b[i]是第i条切线在圆A和圆B上的切点
int  getTangents(Circle A,Circle B,Point* a,Point* b){
   int cnt=0;
   if(A.r<B.r){swap(A,B);swap(a,b);}
   double d=Length(A.c-B.c),diff=A.r-B.r,rsum=A.r+B.r;
   if(d==0 && A.r==B.r)return -1;//重合。无数条切线。
   if(d<diff)return 0;//内含
   double base=angle(A.c-B.c);
   if(d==diff) {//内切。1条切线。
        a[cnt]=A.point(base);b[cnt++]=B.point(base);return 1;
   }
```



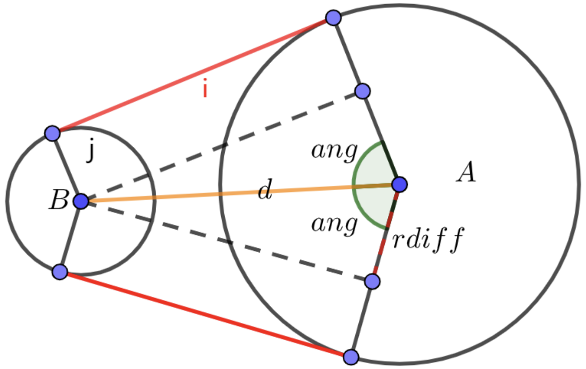
外公切线情况



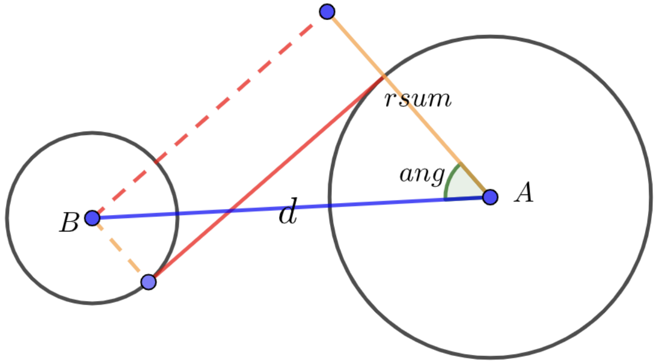
内切线情况（只画出一侧）

两个圆的共切线

```
 //续上
double ang=acos((A.r-B.r)/d); //有外公切线
a[cnt]=A.point(base+ang);b[cnt++]=B.point(base+ang);
a[cnt]=A.point(base-ang);b[cnt++]=B.point(base-ang);
if(d==rsum) //一条内公切线
 {a[cnt]=A.point(base);b[cnt++]=B.point(base+PI);}
else if(d>rsum){//两条内公切线
     double ang=acos(rsum/d);
     a[cnt]=A.point(base+ang);
     b[cnt++]=B.point(base+ang+PI);
     a[cnt]=A.point(base-ang);
     b[cnt++]=B.point(base-ang+PI);
 }
 return cnt;
}
```



外公切线情况



内切线情况（只画出一侧）

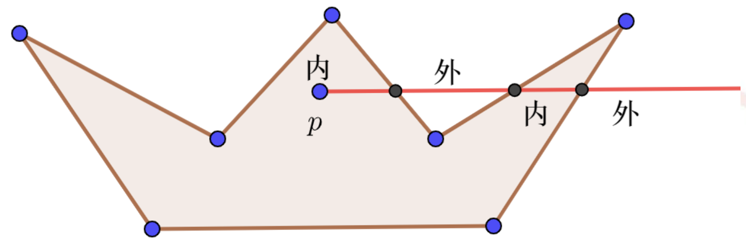
二维几何常用算法

点在多边形内判定

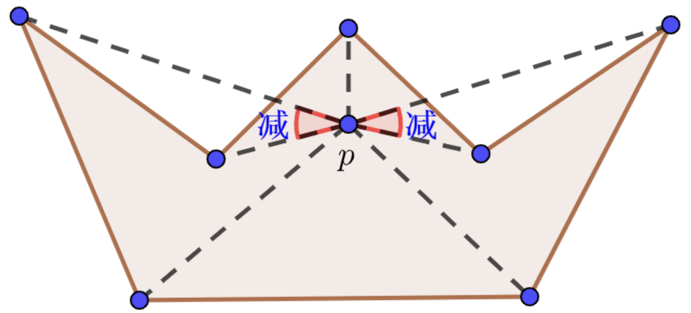
- 一个顶点按逆时针顺序排列的多边形，给定一个点，判断点是否在多边形内。
- 如果是凸多边形，只需判断该点是否在所有边的左边即可。对于所有多边形，主要有两种方法：

✓ **射线法**：从判定点出发，任意引一条射线。如果和边界相交奇数次，说明点在多边形内；如果相交偶数次，说明点在多边形外。因为每相交一次，就切换内外关系（如下图）。注意射线如果在端点处和多边形相交，或者穿过一条完整的边，则需要重新引一条射线，或者通过一些条件判断。

✓ **转角法**：统计多边形每条边相对该点转角之和。如果是**360°**，说明在多边形内；如果是**0°**，说明在多边形外；如果是**180°**，说明在多边形边界上。



射线法示意图

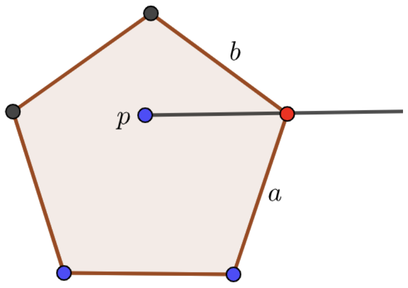


转角法示意图（图中红色角是要减去的）

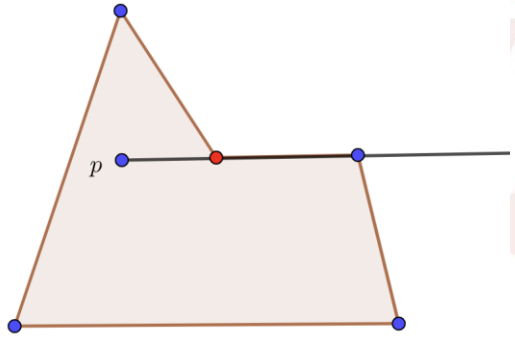
二维几何常用算法

❑点在多边形内判定

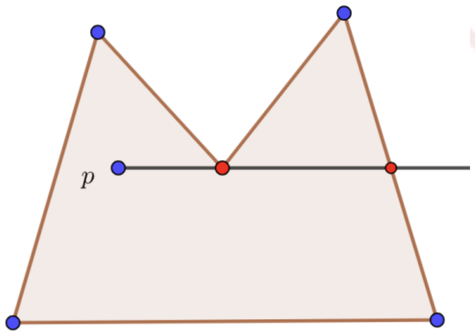
➢ 射线法两个需要特殊处理的情况（红色被计入）。



情况一（只能算一个交点）



情况二（被射线覆盖的边不计入交点数）



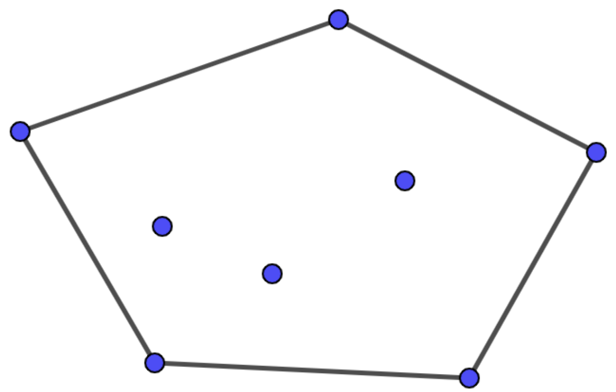
情况三（左边红色点算2次或0次）

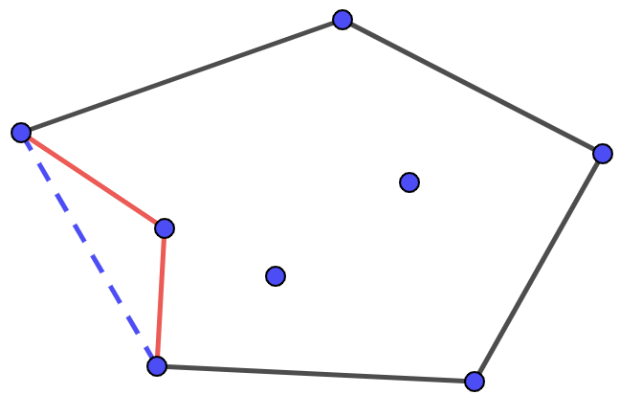
```
bool OnSegment(Point p,Point a,Point b)
{
    return Cross(b-a,p-a)==0 && dcmp(Dot(a-p,b-p))<0;
}
//点要有顺序，顺时针或逆时针皆可。返回1表示在多边形内，0表示在多边形外，-1表示在多边形上
int  InPolygon(Point p,Point poly[])
{
    int wn=0;
    for(int i=0;i<n;++i){
        if(OnSegment(p,poly[i],poly[(i+1)%n]))return -1;
        int k=dcmp(Cross(poly[(i+1)%n]-poly[i],p-poly[i]));
        int d1=dcmp(poly[i].y-p.y);
        int d2=dcmp(poly[(i+1)%n].y-p.y);
        if(k>0 && d1<=0 && d2>0) wn++;
        if(k<0 && d1>0 && d2<=0) wn++;
    }
    return wn%2;
}
```

凸包

□凸包的定义

- 对于给定集合X，所有包含X的凸集的交集S被称为X的凸包。
- 凸包是把给定点包围在内部的、面积最小、周长最小的凸多边形。
- 可以理解为用一个橡皮筋包含住所有给定点的形态。





两边之和大于第三边

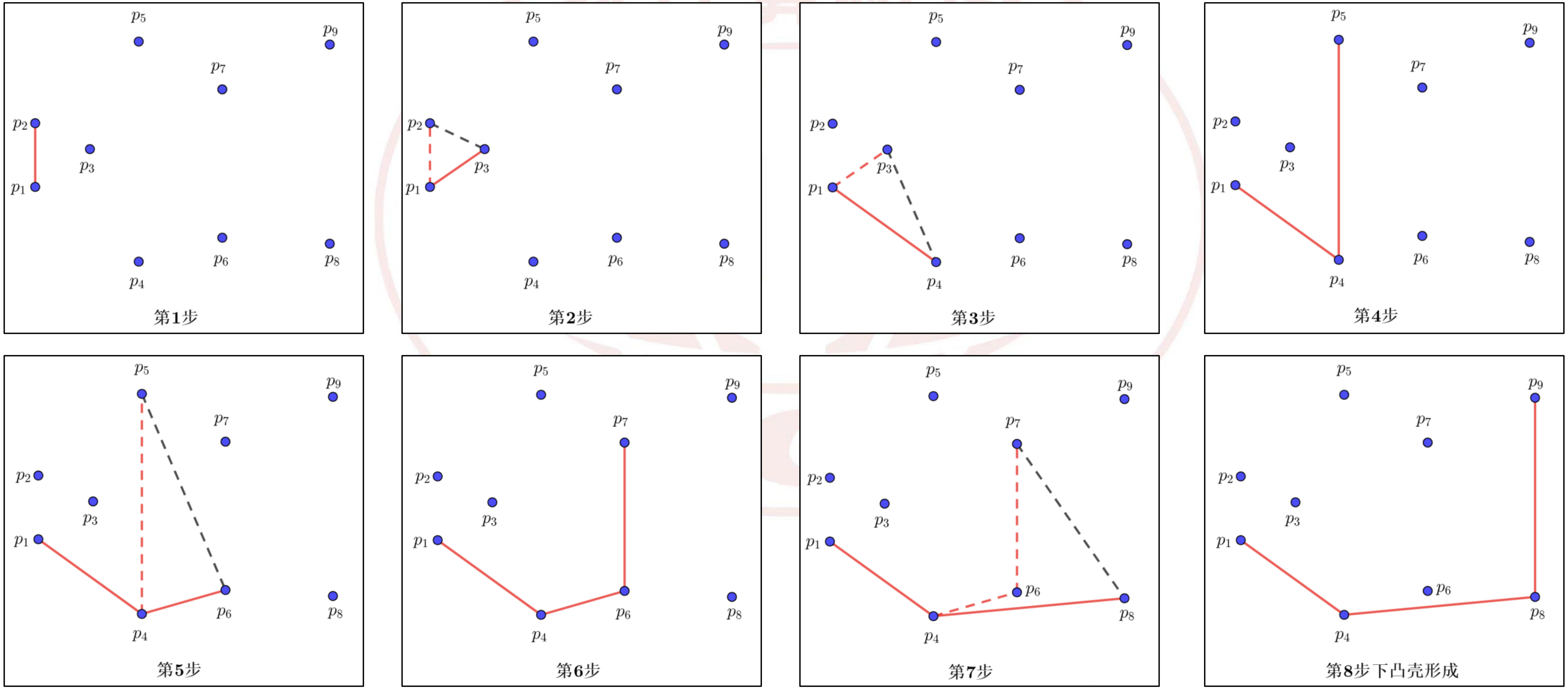
凸包

□Andrew算法

- 把所有点按照x从小到大排序（如果x相同，按照y从小到大排序），删除重复点后得到序列$p_1,p_2,\cdots p_n$
- 凸多边形满足从任意一点出发逆时针走，轨迹总是“左拐”的，一旦出现右拐，就说明这一段不在凸包上。因此我们可以用一个单调栈来维护上下凸壳。我们首先 升序枚举维护出下凸壳，然后降序维护出上凸壳。
- 升序枚举首先把p_1和p_2放进栈中。从p_3开始，当新点在凸包“前进”方向的左边时则进栈，否则依次删除最近加入凸包的点，直到新点在左边或栈中只有一个元素时再加进栈。维护结束形成下凸壳。
- 同样再从p_n开始降序维护出上凸壳。合并起来就是完整的凸包。

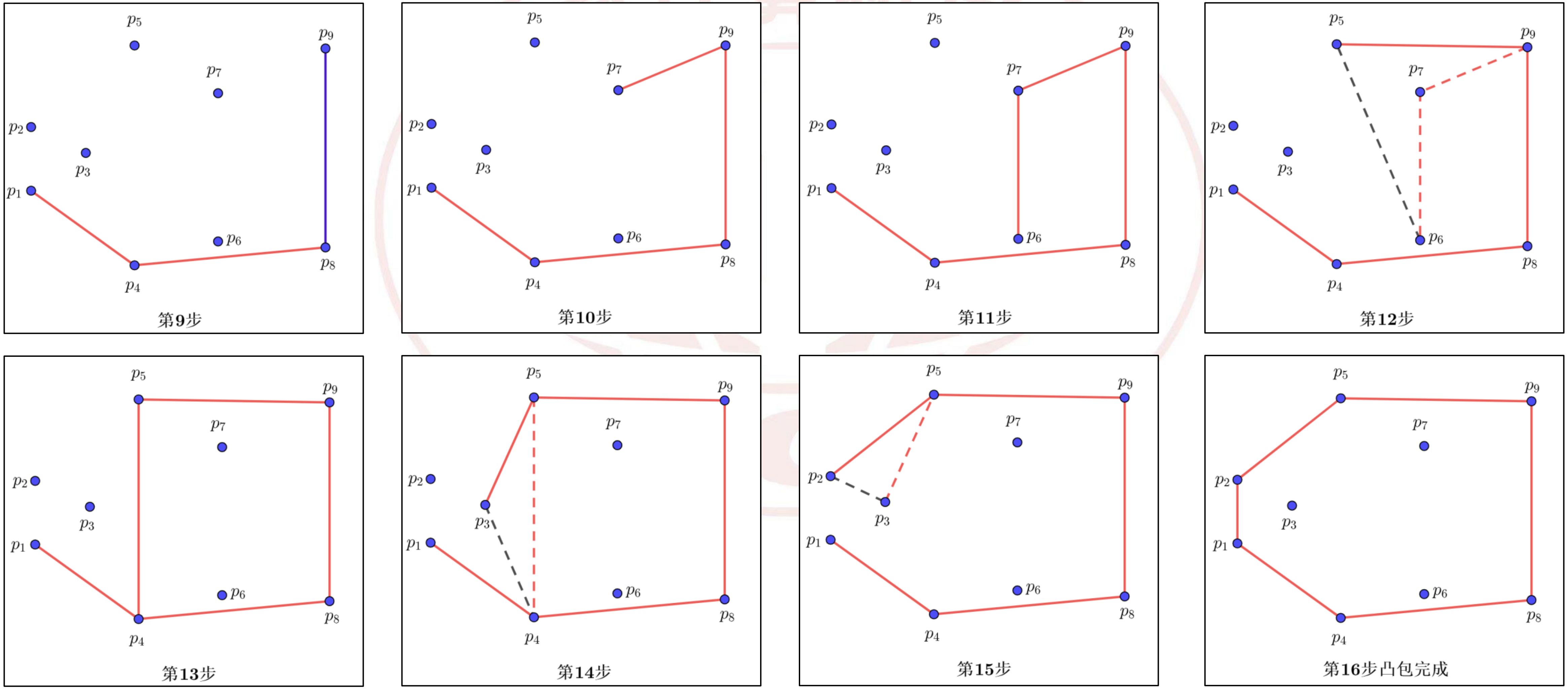
凸包

□Andrew算法演示



凸包

□Andrew算法演示



凸包

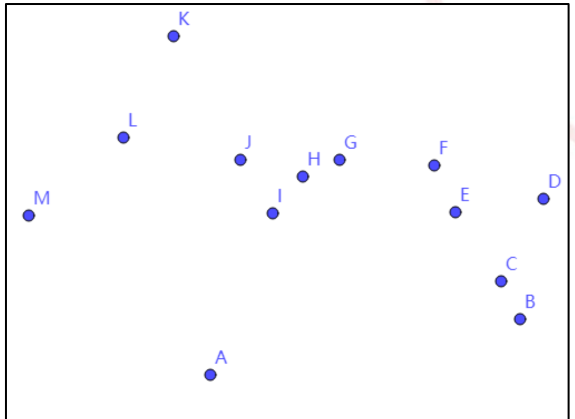
□ **Andrew算法代码。时间复杂度为$O(nlgn)$。**输入点数组p，点的个数为n，输出点数组ch表示凸包。函数返回凸包顶点数。

```
int  ConvexHull(Point* p,int n,Point* ch){
     sort(p,p+n);//先比较x坐标，再比较y坐标
      int m=0;
      for(int i=0;i<n;++i){//维护下凸壳
           while(m>1 && Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0)m--;ch[m++]=p[i];
      }
      int k=m;
      for(int i=n-2;i>=0;--i) {//维护上凸壳
           while(m>k && Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0)m--;ch[m++]=p[i];
      }
      if(n>1)m--;//p[0]多算一次
      return m;
}
```

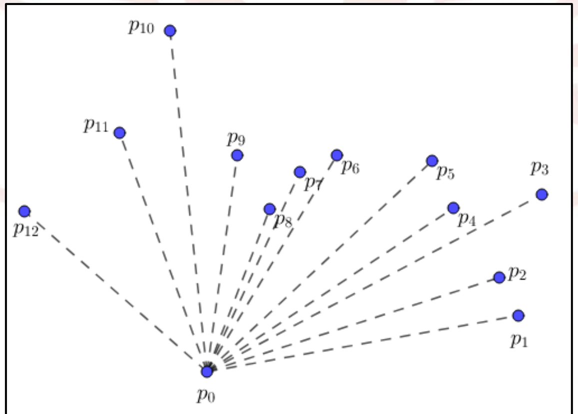
凸包

□Graham扫描法

- **寻找起点**：找到给定点集中纵坐标最小的点作为起点，如果最小纵坐标的点有多个，选择其中横坐标最小的点。如图中的p_0。
- **极角排序**：以p_0为极点，p_0向右方向为极轴，计算其余各点相对于p_0的极角，并按极角从小到大对其余各点排序，当极角相同时，距离p_0较近的点排在前面。
- **维护单调栈**：p_0, p_1入栈，对于$p_i (2 \le i \le n-1)$，如p_i不在当前栈顶两元素形成前进方向的“左拐”方向，则依次删除栈顶元素，直到满足p_i在当前栈顶两元素前进方向的左侧，或栈中只有一个元素为止，最后再把p_i入栈。最终留在栈中的点按逆时针顺序构成凸包。



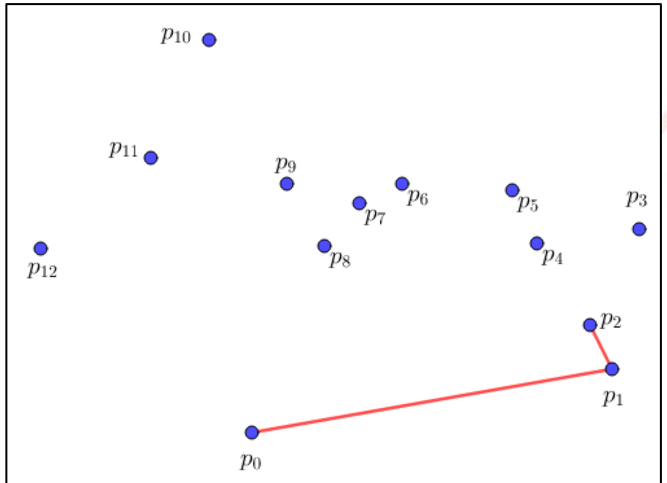
原点集



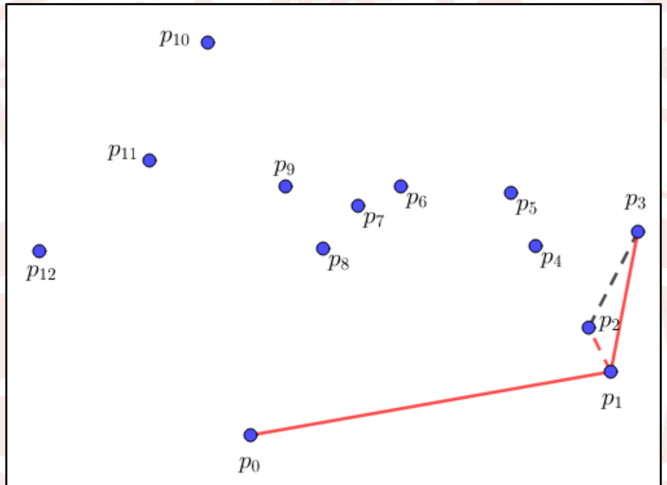
极角排序后

凸包

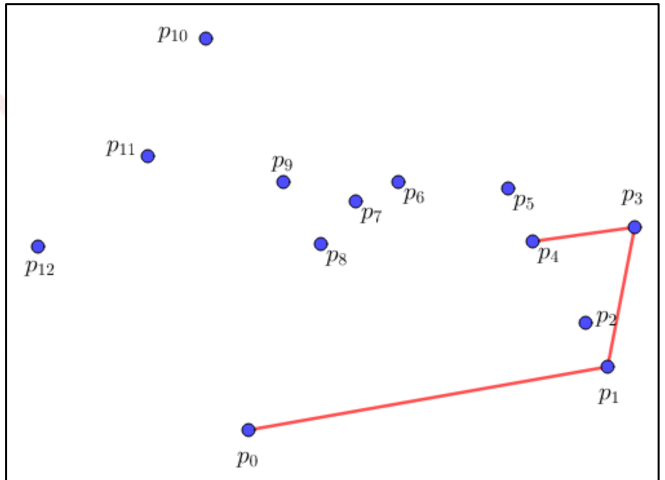
□Graham算法演示



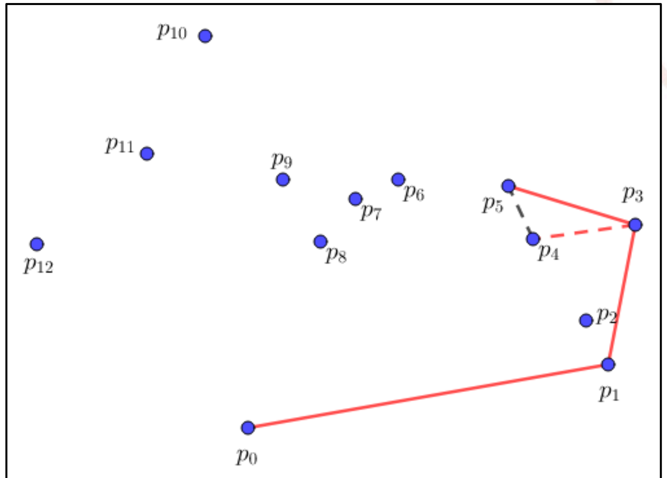
第1步



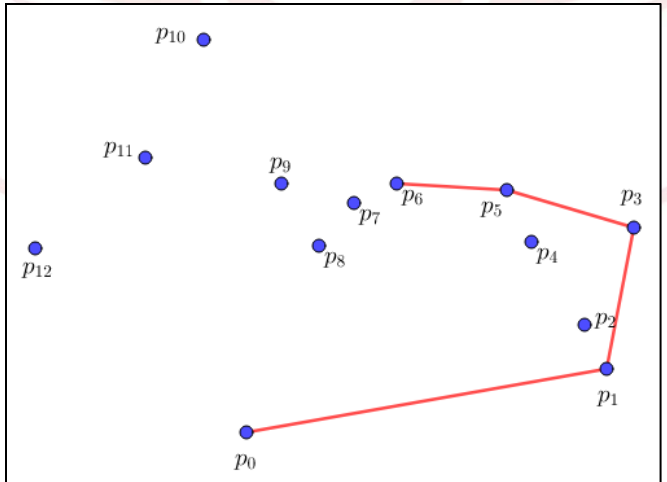
第2步



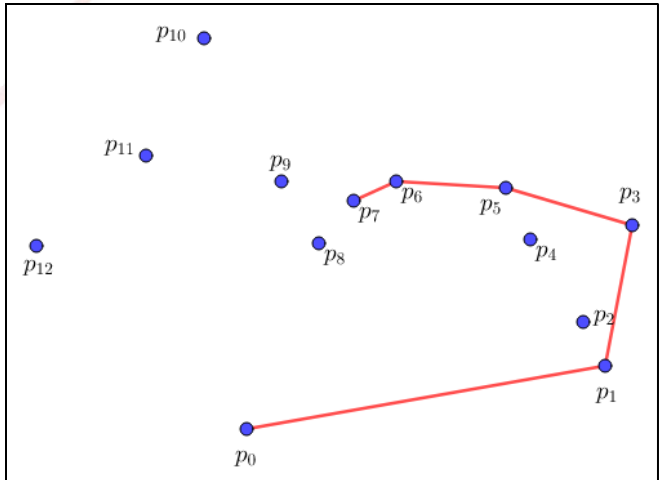
第3步



第4步



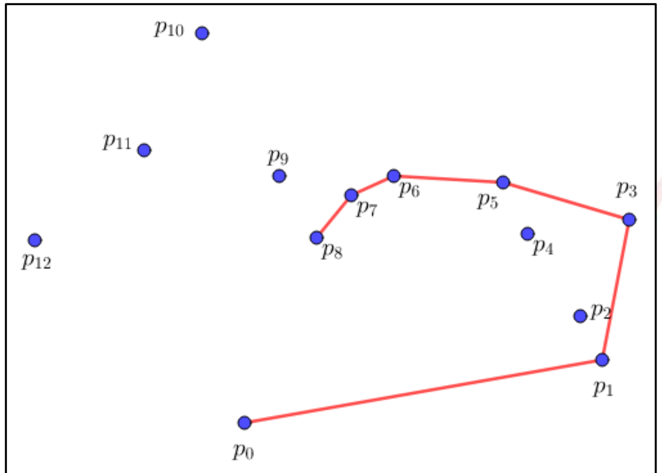
第5步



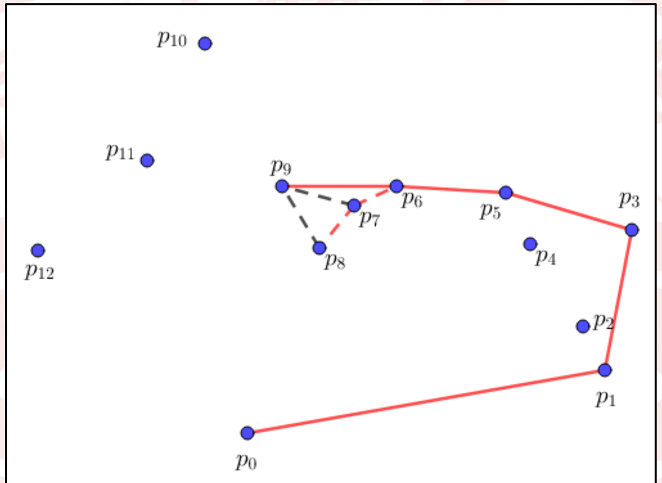
第6步

凸包

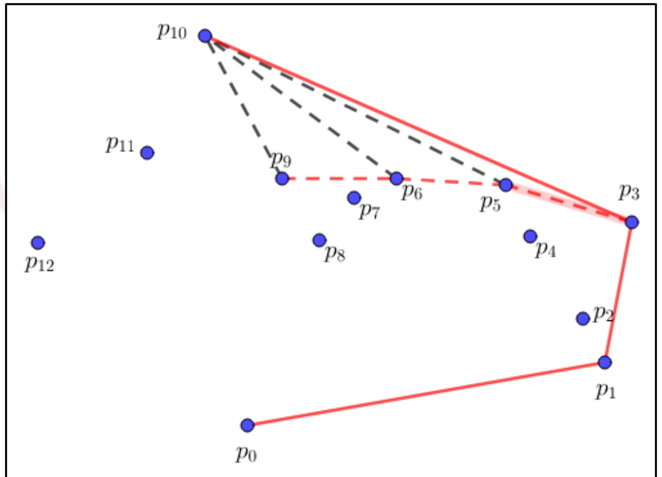
□Graham算法演示



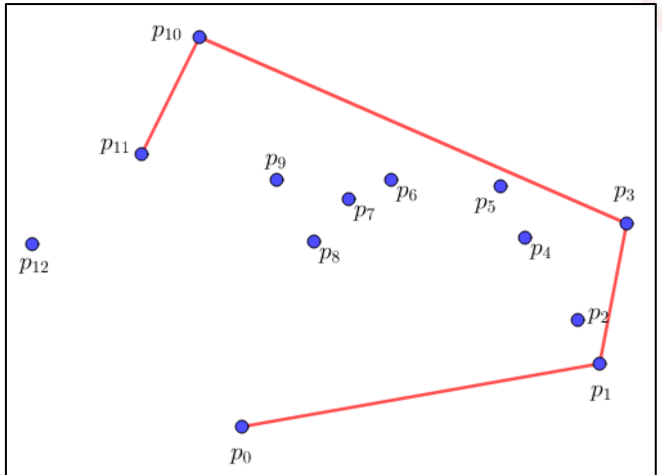
第7步



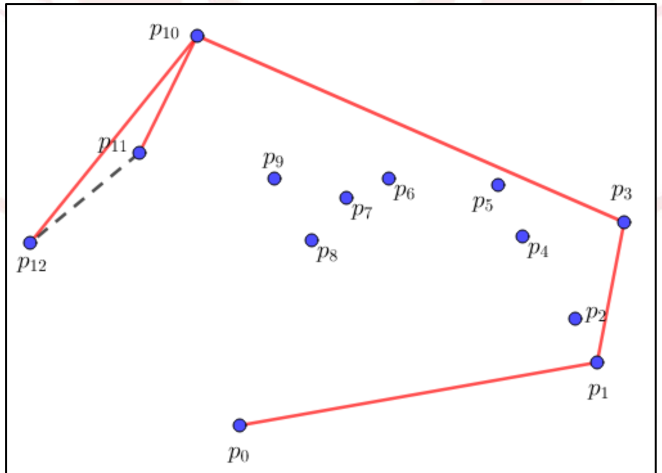
第8步



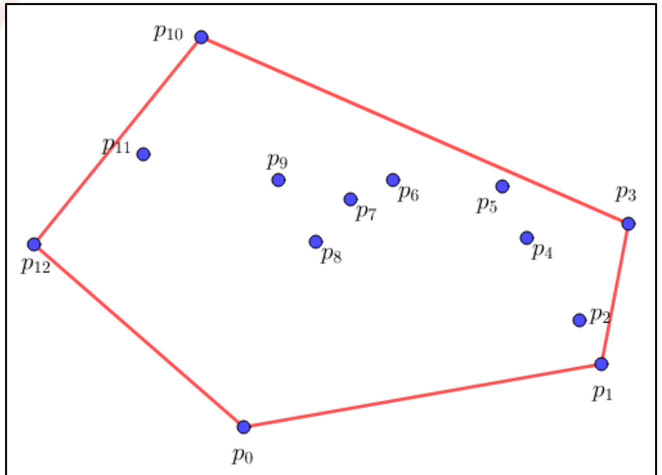
第9步



第10步



第11步



第12步

凸包

□Graham算法代码。时间复杂度为$O(nlgn)$。

```
int  ConvexHull(Point* p,int n,Point* ch){
    //选择起点
    for(int i=1;i<n;++i){ if(p[i].y<p[0].y|| p[i].y==p[0].y && p[i].x<p[0].x) swap(p[0],p[i]);}
    sort(p+1,p+n,cmp);//按极角排序，可以利用叉积
    int m=2;
    ch[0]=p[0];ch[1]=p[1];
    for(int i=2;i<n;++i){
        while(m>1 && Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])<=0)m--;ch[m++]=p[i];
    }
    return m;
}
```

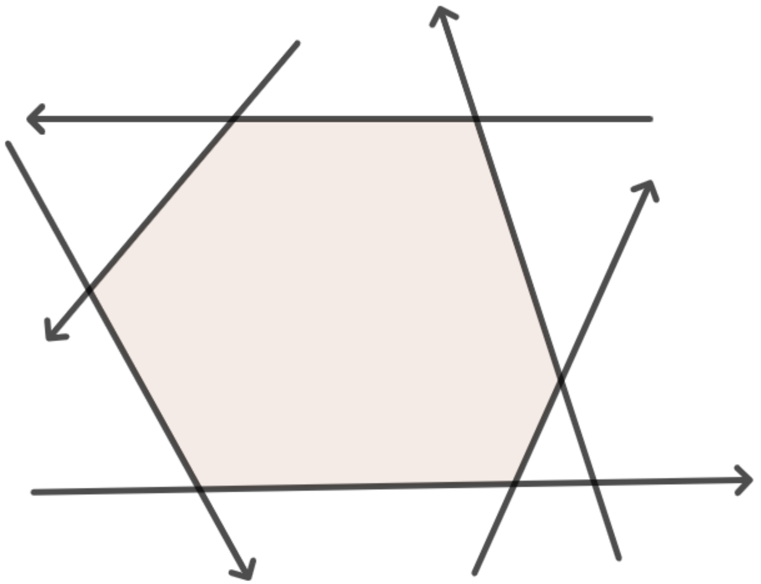
半平面交

□ 半平面交问题定义

➢ 半平面交问题就是给出若干个半平面，求它们的公共部分。如图。半平面是一个点集，可以认为是一条直线和直线的左侧构成的点集。当包含直线时，称为闭半平面；不包含直线，称为开半平面。

➢ 每个半平面用一条有向直线表示，它的左侧就是它所代表的半平面。有向直线的定义如下：

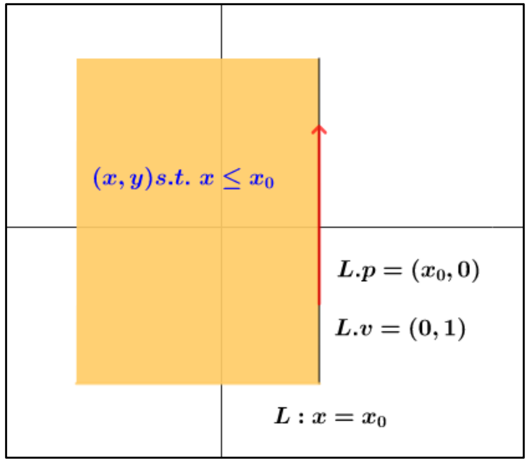
```
struct Line{
    Point p;//有向直线上的一点
    Vector v;//有向直线的方向向量
    double ang;//极角
    Line(Point p , Vector v):p(p),v(v){ang=atan2(v.y,v.x); }
    bool operator <(const Line& L)const{//排序用的比较运算符
        return ang<L.ang;
    }
}
```

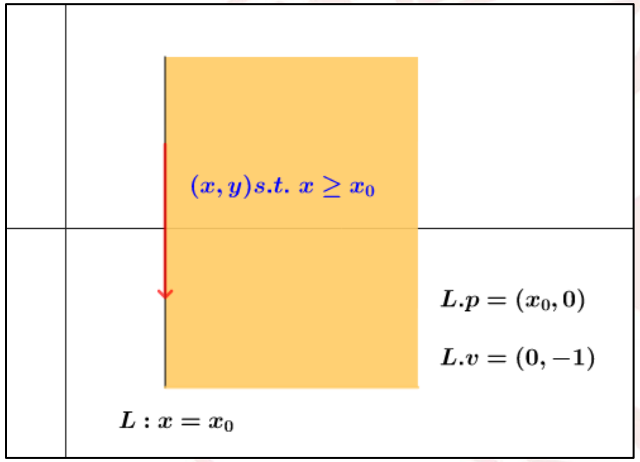


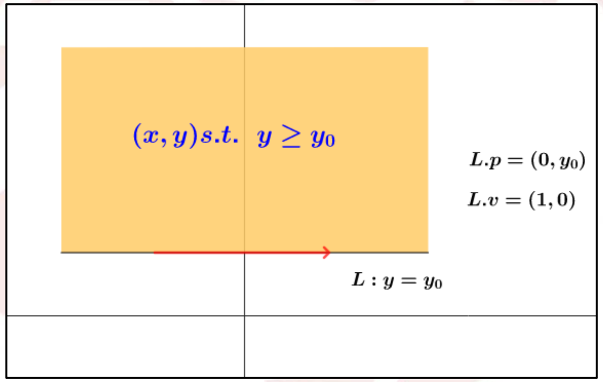
半平面交

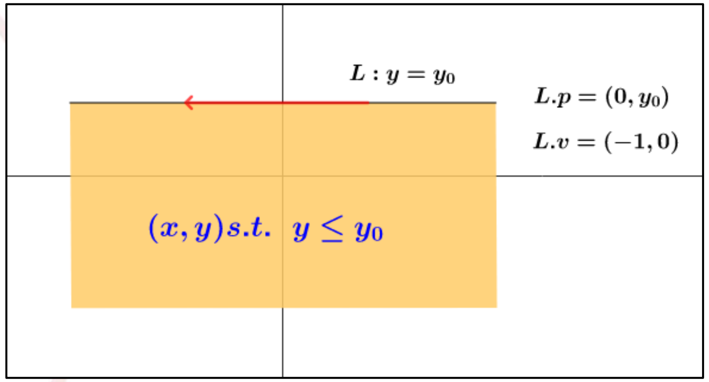
半平面交问题定义

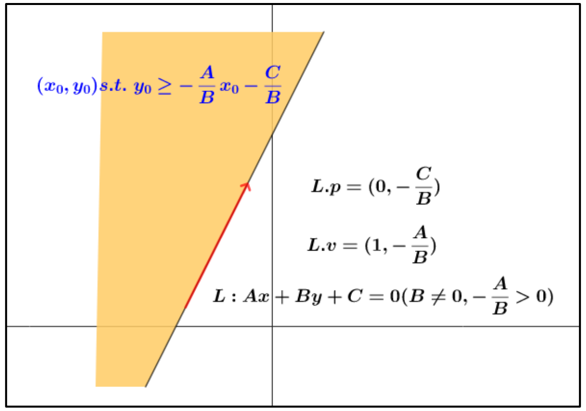
➢ 直线L的解析式为Ax+By+C=0。注意要保证有向直线（起点和方向向量决定）的左侧要跟表示的点集保持一致！

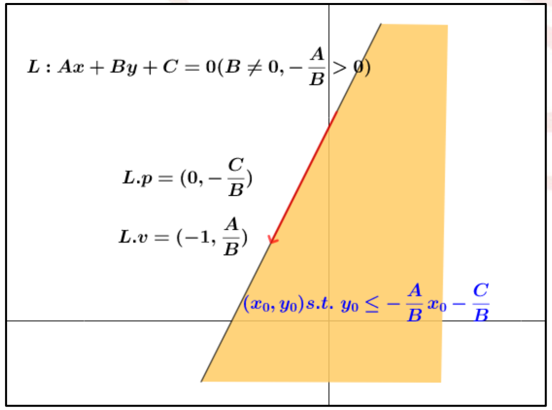


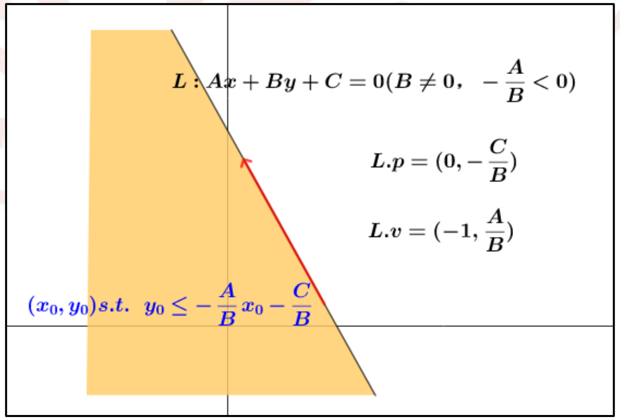


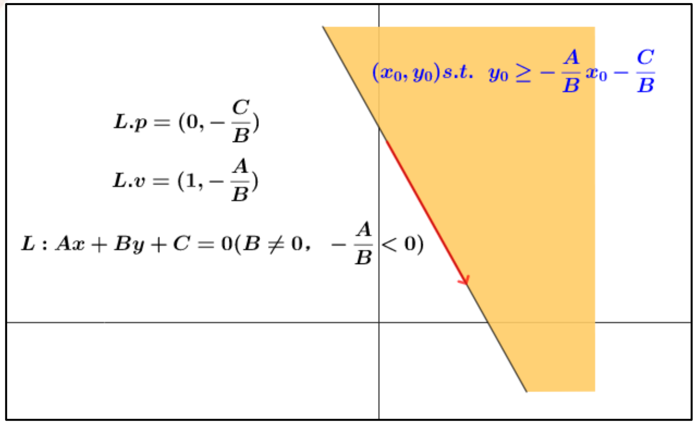












中国计算机学会
China Computer Federation

半平面交

□半平面交问题定义

➢ 半平面也可能是以Ax+By+C⩾0的形式给出。根据以上分析，其转化为有向直线的方式如下：

① **当A=B=0**:当C⩾0，恒成立，忽略；当C<0,返回无解。

② **当B=0时**：转化为以下两种情况

I. $x \geqslant x_0$,则有向直线L男足：L.p={x_0,0},L.v={0,-1}

II. 当$x \leqslant x_0$时，则有向直线L满足：L.p={x_0,0},L.v={0,1}

③ **当B≠0时**：转化为以下两种情况

I. y⩾ax+b,则有向直线L满足：L.p={0,b},L.v={1,a}

II. y⩽ax+b,则有向直线L满足：L.p={0,b},L.v={-1,-a}

半平面交

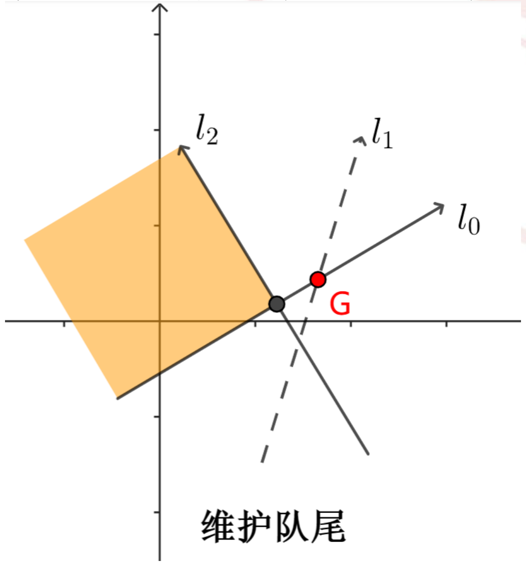
半平面交算法

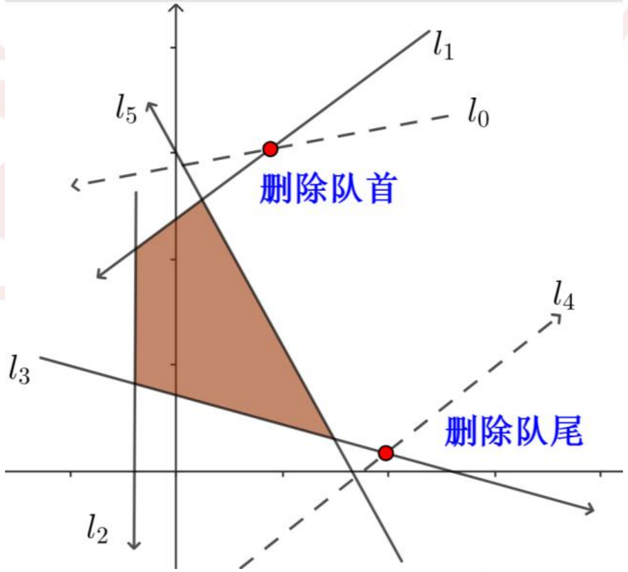
极角排序

按照向量的极角从小到大排序，得到新的边（向量）集。

维护单调队列

因为半平面交是一个凸多边形，所以需要维护一个凸壳。按照极角顺序依次加入边，后加入的边可能会影响最开始加入的或最后加入的边（此时凸壳连通），需要用单调队列维护队首和队尾的元素。我们遍历排好序的向量，并维护一个交点数组。当队列中元素超过 1 个时，他们之间就会产生交点。对于当前向量，如果队尾交点在这条向量表示的半平面交的右侧，那么上一条边就没有意义了，要进行删除。对于队首交点进行同样的处理。解释如下：





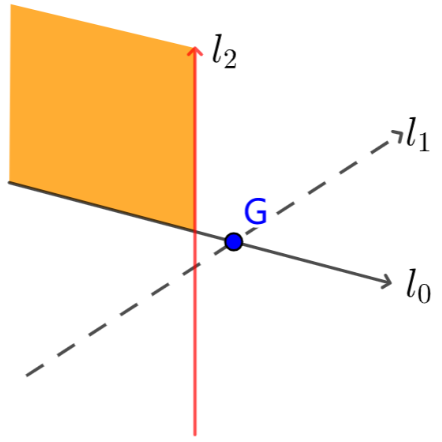
CCF 中国计算机学会 China Computer Federation

NOI

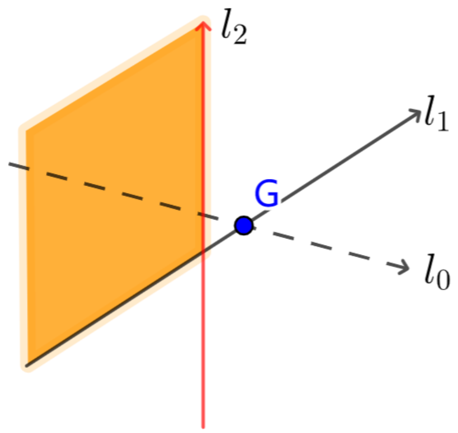
半平面交

两点注意事项

- ①必须先处理队尾，再处理队首。原因如下图

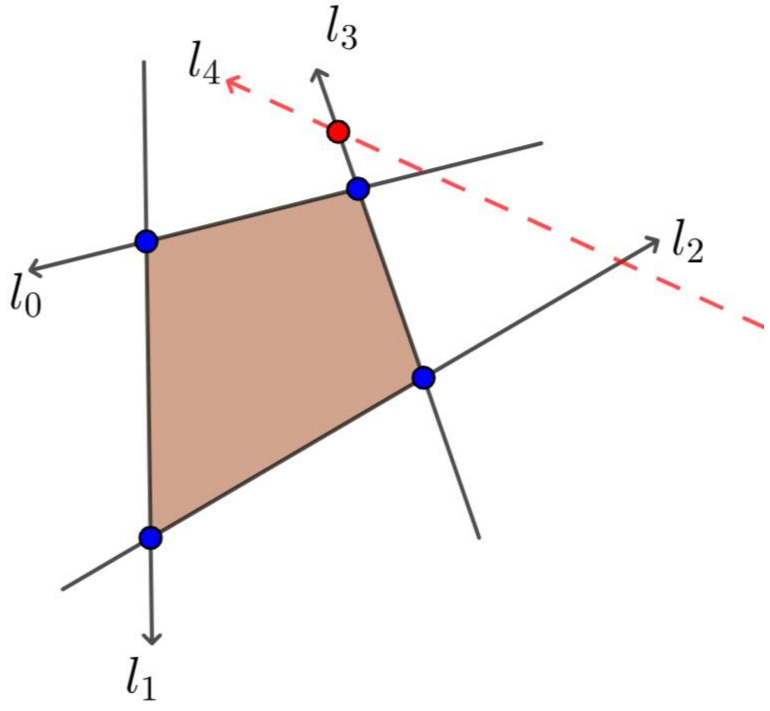


先处理队尾



先处理队首

- ②处理结束后，要用队首的向量排除一下队尾多余的向量。因为队首的向量会被后面的约束，而队尾的向量不会。



半平面交

- 程序代码。时间复杂度为$O(nlgn)$。

```
int  HalfplaneIntersection(Line* L,int n,Point* poly){
     sort(L,L+n,cmp);//按极角排序，极角相同外侧优先
     Point *p=new Point[n];//p[i]为q[i]和q[i+1]的交点
     Line *q=new Line[n];
     int head=0,tail=0,cnt=0;
     //极角相同时只保留内侧的向量
     for(int i=0;i<n-1;++i){
         if(dcmp(L[i].ang-L[i+1].ang)==0)continue;
         L[cnt++]=L[i];
      }
    L[cnt++]=L[n-1];
    q[0]=L[0];
    for(int i=1;i<cnt;++i){
       while(head<tail && !Onleft(L[i],p[tail-1]))tail--;//维护队尾
       while(head<tail && !Onleft(L[i],p[head]))head++;//维护队首
        q[++tail]=L[i];
        if(head<tail)
            p[tail-1]=GetLineIntersection(q[tail-1].p,q[tail-1].v,q[tail].p,q[tail].v);
     }
    while(head<tail && !Onleft(q[head],p[tail-1]))tail--;
    if(tail-head<=1)return 0;
    p[tail]=GetLineIntersection(q[tail].p,q[tail].v,q[head].p,q[head].v);
    int m=0;
    for(int i=head;i<=tail;++i)poly[m++]=p[i];
    return m;
}
```

三维几何基础

□常用定义

```
struct Point3{
  double x,y,z;
  Point3(double x=0,double y=0,double z=0):x(x),y(y),z(z){ }
} ;
typedef Point3 Vector3;
Vector3 operator + (Vector3 A , Vector3 B){return Vector3(A.x+B.x,A.y+B.y,A.z+B.z);}
Vector3 operator - (Point3 A , Point3 B) {return Vector3(A.x-B.x,A.y-B.y,A.z-B.z);}
Vector3 operator * (Vector3 A , double p){return Vector3(A.x*p,A.y*p,A.z*p);}
Vector3 operator / (Vector3 A , double p){return Vector3(A.x/p,A.y/p,A.z/p);}
```

三维几何基础

□直线

➢直线仍然可以用参数方程（点和向量）来表示。

□平面的表示

➢通常用点法式$(\boldsymbol{p_0},\boldsymbol{n})$来描述一个平面。其中$\boldsymbol{p_0}$是平面上的一个点，向量$\boldsymbol{n}$是平面的法向量。向量$\boldsymbol{n}$垂直于平面上的所有直线。

➢平面上的任意点$\boldsymbol{p}$满足$\mathbf{Dot(n, p}-\boldsymbol{p_0})$=0。设点p的坐标为(x,y,z)，$\boldsymbol{p_0}$的坐标为$(\boldsymbol{x_0},\boldsymbol{y_0},\boldsymbol{z_0})$,向量n的坐标表示为(A,B,C)。则有：

$$\boldsymbol{A*(x-x_0)+B*(y-y_0)+C*(z-z_0)=0}$$

令D=$-\boldsymbol{Ax_0-By_0-Cz_0}$,得平面一般式$\boldsymbol{Ax+By+Cz+D=0}$

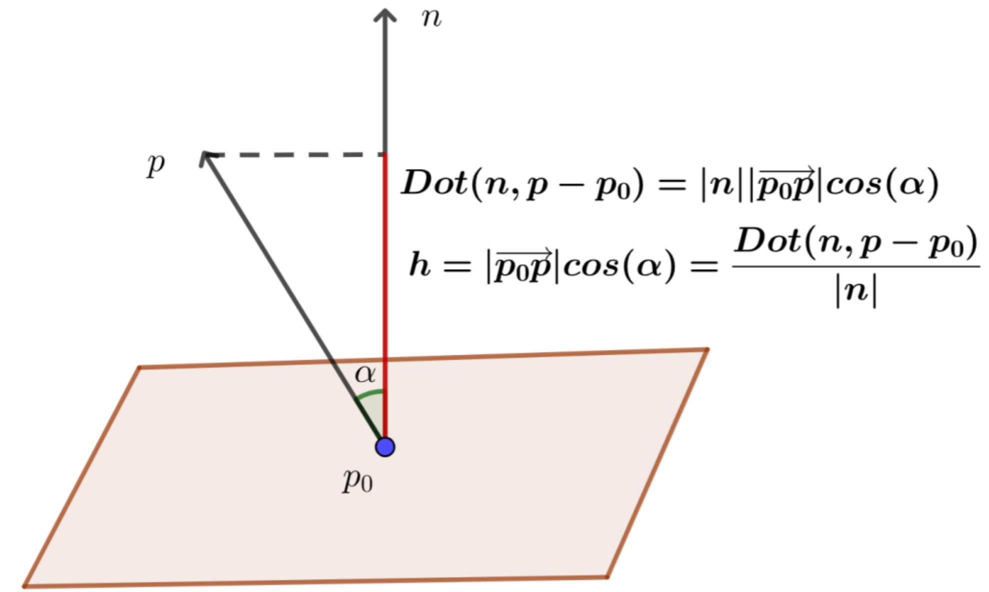
三维点积

□三维点积的定义和二维类似，也能用点积计算向量的长度和夹角，余弦定理可以证明。

```
double Dot(Vector3 A,Vector3 B){return A.x*B.x+A.y*B.y+A.z*B.z;}
double Length(Vector3 A){return sqrt(Dot(A,A));}
double Angle(Vector3 A,Vector3 B){return acos(Dot(A,B)/Len
```

□点到平面的距离

```
double DistanceToPlane(Point3 p,Point3 p0,Vector3 n)
{
    return fabs(Dot(n,p-p0))/Length(n);
}
```

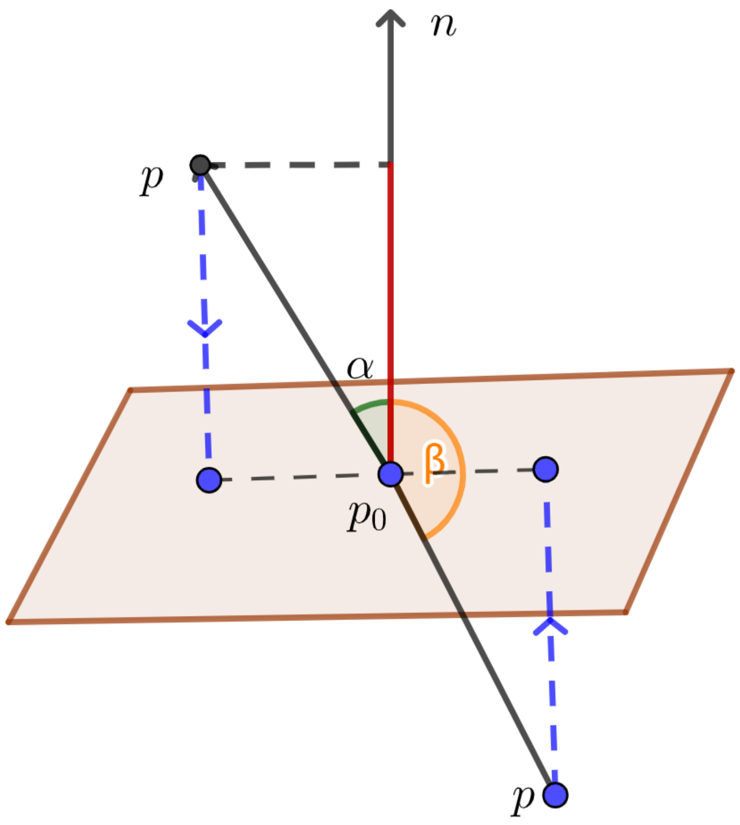


点到平面的距离

三维点积

□点到平面的投影

```
Point3 GetPlaneProjection(Point3 p,Point3 p0,Vector3 n)//其中n为单位法向量
{
    return p-Dot(p-p0,n)*n;
}
```



三维点积

□直线和平面的交点

- 过点$\boldsymbol{p_1}$和$\boldsymbol{p_2}$的直线参数方程为$p = p_1 + t(p_2 - p_1)$
- p是平面上一点，还满足$Dot(n, p - p_0) = 0$. 联立得：

$$Dot(n, \boldsymbol{p_1} - p_0 + \mathbf{t}(\boldsymbol{p_2} - \boldsymbol{p_1})) = 0 \quad t = \frac{Dot(n, p_0 - p_1)}{Dot(n, p_2 - p_1)}$$

```
Point3 LinePlaneIntersection(Point3 p1,Point3 p2,Point3 p0,Vector3 n)
{
    Vector3 v=p2-p1;
    double t=Dot(n,p0-p1)/Dot(n,v);//Dot(n,v)不能等于0
    return p1+v*t;
}
```

三维叉积

□三维叉积

➢三维叉积和二维叉积不一样，三维叉积是一个向量，而不再是一个带符号的数。

➢设$\boldsymbol{v_1}$=$(\boldsymbol{x_1},\boldsymbol{y_1},\boldsymbol{z_1})$,$\boldsymbol{v_2}$=$(\boldsymbol{x_2},\boldsymbol{y_2},\boldsymbol{z_2})$,则：

$$\boldsymbol{v_1} * \boldsymbol{v_2} = \det\begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ \boldsymbol{x_1} & \boldsymbol{y_1} & \boldsymbol{z_1} \\ \boldsymbol{x_2} & \boldsymbol{y_2} & \boldsymbol{z_2} \end{bmatrix} = (\boldsymbol{y_1 z_2} - \boldsymbol{y_2 z_1})\boldsymbol{i} + (\boldsymbol{x_2 z_1} - \boldsymbol{x_1 z_2})\boldsymbol{j} + (\boldsymbol{x_1 y_2} - \boldsymbol{x_2 y_1})\boldsymbol{k}$$

$$=(\boldsymbol{y_1 z_2} - \boldsymbol{y_2 z_1}, \boldsymbol{x_2 z_1} - \boldsymbol{x_1 z_2}, \boldsymbol{x_1 y_2} - \boldsymbol{x_2 y_1})$$

□三维叉积的几何意义

➢三维叉积同时垂直于$\boldsymbol{v_1}, \boldsymbol{v_2}$,方向遵守右手定则。

➢三维叉积向量的大小是向量$\boldsymbol{v_1}, \boldsymbol{v_2}$共起点时构成的平行四边形的面积。

中国计算机学会 China Computer Federation

三维叉积

□三维叉积几何意义证明

- Dot($v_1 * v_2$,v_1)=$(y_1z_2-y_2z_1, x_2z_1-x_1z_2, x_1y_2-x_2y_1)\cdot(x_1,y_1,z_1)$=$x_1(y_1z_2-y_2z_1)+y_1(x_2z_1-x_1z_2)+z_1(x_1y_2-x_2y_1)$=$x_1y_1z_2-x_1y_2z_1+x_2y_1z_1-x_1y_1z_2+x_1y_2z_1-x_2y_1z_1=0$
- Dot($v_1 * v_2$,v_2)=$(y_1z_2-y_2z_1, x_2z_1-x_1z_2, x_1y_2-x_2y_1)\cdot(x_2,y_2,z_2)$=$x_2(y_1z_2-y_2z_1)+y_2(x_2z_1-x_1z_2)+z_2(x_1y_2-x_2y_1)$=$x_2y_1z_2-x_2y_2z_1+x_2y_2z_1-x_1y_2z_2+x_1y_2z_2-x_2y_1z_2=0$
- 证明$|v_1 * v_2|=|v_1||v_2|sin(\alpha)$,其中$\alpha$为向量$v_1$和$v_2$之间的夹角，范围为$[0,\pi]$

✓ $|v_1 * v_2|^2=(y_1z_2-y_2z_1)^2+(x_2z_1-x_1z_2)^2+(x_1y_2-x_2y_1)^2$

✓ $(|v_1||v_2|sin(\alpha))^2=|v_1|^2|v_2|^2-(|v_1||v_2|cos(\alpha))^2=(x_1{}^2+y_1{}^2+z_1{}^2)(x_2{}^2+y_2{}^2+z_2{}^2)$-$(x_1x_2+y_1y_2+z_1z_2)^2=(y_1z_2-y_2z_1)^2+(x_2z_1-x_1z_2)^2+(x_1y_2-x_2y_1)^2$

得证！

三维叉积

➢三维叉积

```
Vector3 Cross(Vector3 A,Vector3 B)
{
    return Vector3(A.y*B.z-A.z*B.y,B.x*A.z-A.x*B.z,A.x*B.y-A.y*B.x);
}
```

➢三点形成平行四边形的面积

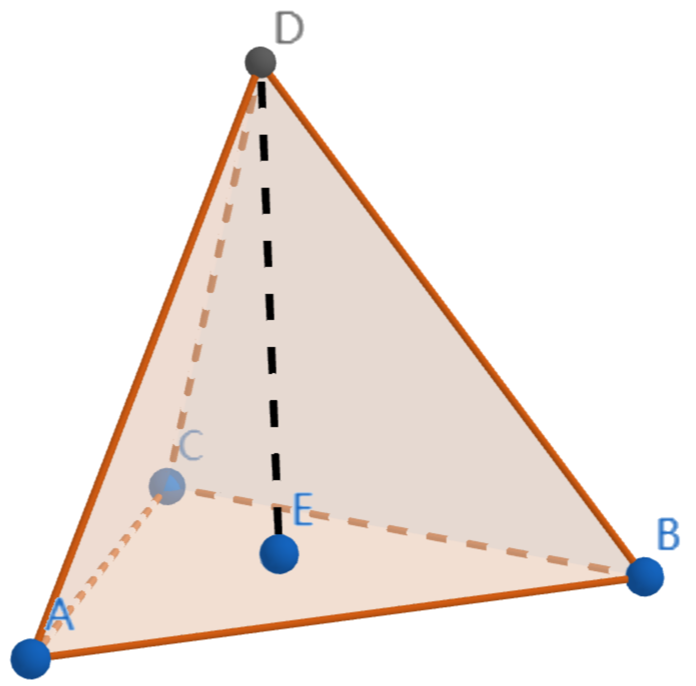
```
double Area2(Point3 A,Point3 B,Point3 C)
{
    return Length(Cross(B-A,C-A));
}
```

三维叉积

▫四面体的体积

➢ 四面体带符号体积$V=\frac{1}{3}$S·h=$\frac{1}{6}(\overrightarrow{AB}*\overrightarrow{AC})\cdot h=\frac{1}{6}((\overrightarrow{AB}*\overrightarrow{AC})\cdot\overrightarrow{AD})$

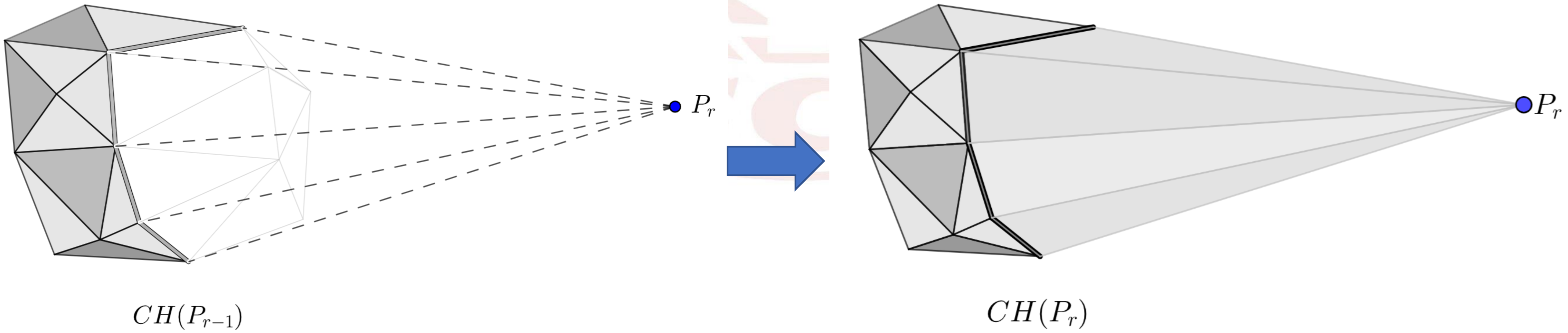
```
double Volume6(Point3 A,Point3 B,Point3 C,Point3 D)
{
    return Dot(D-A,Cross(B-A,C-A));
}
```



三维凸包

□增量法

- ➢首先对每个点进行微小扰动，避免出现四点共面的情况。随便加入三个点，初始化凸包。
- ➢对于一个已知凸包，新增一个点P，将P视作一个点光源，向凸包做射线。易知，光线的可见面和不可见面一定是由若干条棱隔开的。
- ➢将光的可见面删去，并新增由其分割棱与P构成的平面。
- ➢重复此过程即可。



三维凸包

□代码

```
void Convex_3D(){
    f[cnt++]={0,1,2};f[cnt++]={2,1,0};//初始化最初凸包
    for(int i=3;i<n;++i){
        int cc=0;
        for(int j=0;j<cnt;++j){
            int v=see(f[j],A[i]);//判断点A[i]能否看到面f[j],如果能看到则删除，看不见则保留，同时记录面f[j]三条边的可见情况
            if(!v)C[cc++]=f[j]; for(int k=0;k<3;++k)vis[f[j].v[k]][f[j].v[(k+1)%3]]=v;
        }
        for(int j=0;j<cnt;++j)
            for(int k=0;k<3;++k){
                int x=f[j].v[k],y=f[j].v[(k+1)%3];if(vis[x][y] && !vis[y][x])C[cc++]={x,y,i};//若x-y是分割棱，则添加新的面{x,y,i}
            }
        for(int j=0;j<cc;++j)f[j]=C[j];cnt=cc;
    }
}
```

三维凸包

□时间复杂度分析

➢**几何体欧拉公式：**V-E+F=2。其中V表示顶点数，E表示边数，F表示面数。

➢证明：设想这个多面体是先有一个面，然后将其他各面一个接一个地添装上去的。因为一共有F个面因此要添(F-1)个面。考察第1个面，设它是n边形，有n个顶点，n条边，这时E=V，即棱数等于顶点数。

添上第2个面后，因为一条棱与原来的棱重合，而且有两个顶点和第1个面的两个顶点重合，所以增加的棱数比增加的顶点数多1，因此，这时E=V+1。

以后每增添一个面，总是增加的棱数比增加的顶点数多1，例如

增添两个面后，有关系E=V+2；

增添三个面后，有关系E=V+3……

增添(F-2)个面后，有关系E=V+ (F-2).

最后增添一个面后，就成为多面体，这时棱数和顶点数都没有增加.因此，关系式仍为E=V+ (F-2).即 V-E+F=2

中国计算机学会
China Computer Federation

三维凸包

□时间复杂度分析

- **几何体欧拉公式：** V-E+F=2。其中V表示顶点数，E表示边数，F表示面数。
- 另有每个面有3条边，且每条边属于两个三角形，因此有3F=2E
- 设V=n，则计算得E=3n-6，F=2n-4
- 增量构造法的复杂度是面数×点数，所以时间复杂度为$O(n^2)$。

例1. POJ2284 好看的一笔画

- 平面上有一个包含n个端点的一笔画，第n个端点总是和第一个端点重合，因此图案是一条闭合曲线。组成一笔画的线段可以相交，但是不会部分重叠。如图所示。求这些线段将平面分成多少部分？（包括封闭区域和无限大区域）
- 输入包含多组数据。每组数据第一行为n(4≤n≤300),第二行为n对整数，依次为一笔画上各顶点的坐标（均为绝对值不超过300的整数）。
- 对于每组数据，输出平面被分成的区域数。

Fig.1

中国计算机学会 China Computer Federation

例2. UVA11796 狗的距离

- 甲和乙两条狗分别沿着一条折线奔跑。两只狗的速度未知，但已知它们同时出发，同时到达，并且都是匀速奔跑。你的任务是求出甲和乙在奔跑过程中的最远距离和最近距离之差。
- 输入第一行为数据组数T(T≤1000)。每组数据第一行为整数A和B，分别表示甲和乙线路上的顶点数。第二行用2*A个整数描述甲的路线，即X1,Y1,X2,Y2,...,XA,YA。其中(X1,Y1)和(XA,YA)分别是路线的起点和终点。第三行用2*B个整数描述乙的路线。
- 对于每组数据，输出所求值。

例3. LA4728 正方形

- 给定平面上n个边平行于坐标轴的矩形，在它们的顶点上找出两个欧几里得距离最大的点。你的任务是输出这个最大距离的平方。
- 输入第一行为数据组数T。每组数据第一行为一个整数n($1 \le n \le 100\,000$)。以下n行每行3个整数x,y,w($0 \le x,y \le 10000, 1 \le w \le 10000$)，其中(x,y)为正方形的左下角顶点，w是边长。
- 对于每组数据，输出所有正方形的顶点中，两点最大距离的平方。

例4. POJ3525 离海最远的点

- 在大海的中央，有一个凸n边形的小岛。你的任务是求出岛上离海最远的点。输出它到海的距离。
- 输入包含多组数据。每组数据第一行为整数n(3≤n≤100),即小岛的顶点数。以下n行按照逆时针顺序给出各个顶点的坐标。坐标均为不超过10000的非负整数。输入结束标志为n=0。
- 对于每组数据，输出离海最远点与海的距离。

例5. POJ1755 铁人三项

- 铁人三项比赛分成连续的3段:游泳、自行车和赛跑。现在每个单项比赛的长度还没定，但已知各选手在每项比赛中的平均速度（假定该平均速度和赛程长度无关），所以你可以设计每项比赛的长度，让其中某个特定的选手获胜。你的任务是判断哪些选手有可能获得冠军（并列冠军不算）。
- 输入包含多组数据。每组数据第一行为选手个数n(1≤n≤100),以下n行每行包含3个整数vi,ui和wi(1≤vi,ui,wi≤10000),即第i选手在游泳、自行车和赛跑比赛中的平均速度。
- 对于每组数据，按照输入顺序给出对每个选手是否能夺冠（且不是并列）的判断，Yes表示有可能，No表示不可能。

例6. POJ1755 铁人三项

- 铁人三项比赛分成连续的3段:游泳、自行车和赛跑。现在每个单项比赛的长度还没定，但已知各选手在每项比赛中的平均速度（假定该平均速度和赛程长度无关），所以你可以设计每项比赛的长度，让其中某个特定的选手获胜。你的任务是判断哪些选手有可能获得冠军（并列冠军不算）。
- 输入包含多组数据。每组数据第一行为选手个数n(1≤n≤100),以下n行每行包含3个整数vi,ui和wi(1≤vi,ui,wi≤10000),即第i选手在游泳、自行车和赛跑比赛中的平均速度。
- 对于每组数据，按照输入顺序给出对每个选手是否能夺冠（且不是并列）的判断，Yes表示有可能，No表示不可能。

例7. LA4992 丛林警戒队

- 在丛林中有n个瞭望台，形成一个凸n多边形。这些瞭望台的保护范围就是这个凸多边形内的任意点。敌人进攻时，会炸毁一些瞭望台，使得总部暴露在那些剩下瞭望台的凸包之外。你的任务时选择一个点作为总部，使得敌人需要炸坏的瞭望台数量尽量多。
- 输入包含多组数据。每组数据第一行为整数n(3≤n≤50000)。以下n行每行两个整数，即每个瞭望塔的坐标，按照顺时针顺序给出。没有3个瞭望台共线的情况。坐标均为绝对值不超过106的整数。
- 对于每组数据，输出当总部位置最优时，敌人需要炸毁的瞭望台数目。

例8. UVA11275 三维三角形

- 给定两颗凸多面体行星，你的任务时求出二者重心的最近距离。两颗行星的密度都是均匀分布的。且可以任意旋转和平移。每颗行星顶点数不超过60.
- 输入包含多组数据，每组数据分成两部分，依次描述两颗行星。每颗行星第一行为顶点数n(4≤n≤60)。以下n行每行给出一个顶点坐标。这n个点保证是一个非退化凸多面体的各个顶点。
- 对于每组数据，输出两个行星重心的最近距离。