计算几何

西安电子科技大学

----曹绍升

计算几何简介

- 计算几何是一门几何学，研究图形几何性质的学科
- 计算几何也是一门计算科学，研究几何的算法性质
- 引入计算几何，让计算机学会处理几何问题

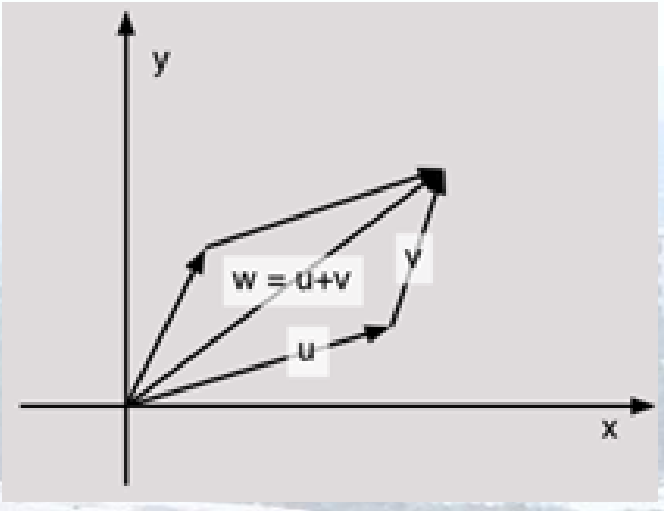
向量代数

叉积的三维意义

$$\mathbf{a}\times\mathbf{b}=\begin{pmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k}\\ x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\end{pmatrix}$$

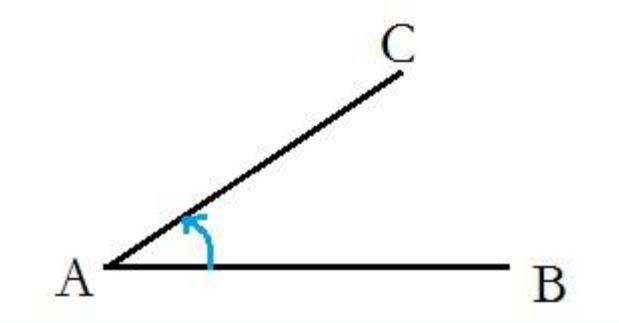
（黑板）

叉积的二维意义----有向面积：外积大小为由 $\boldsymbol{u}$、$\boldsymbol{v}$这两个向量围成的平行四边形有向面积

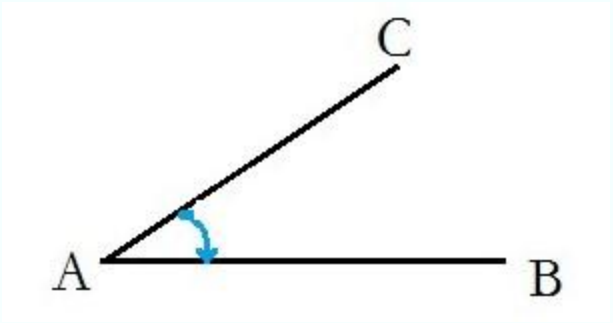


$$\mathbf{a}\times\mathbf{b}=\det\begin{pmatrix}x_1 & y_1\\ x_2 & y_2\end{pmatrix}$$

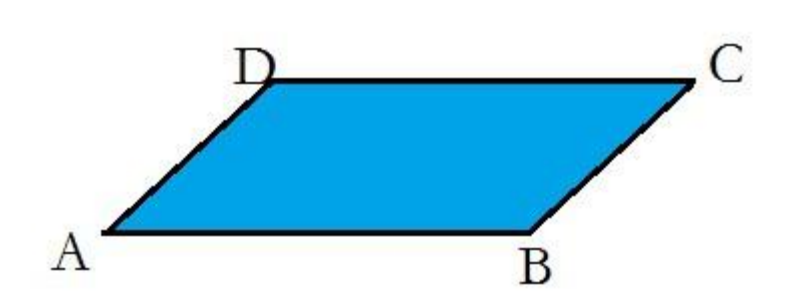
叉积的几何意义



$$\overrightarrow{AB}\times\overrightarrow{AC}>0$$



$$\overrightarrow{AC}\times\overrightarrow{AB}<0$$



$$S_{ABCD}=\overrightarrow{AB}\times\overrightarrow{AD}$$

$$S_{ABCD}=-\overrightarrow{AD}\times\overrightarrow{AB}$$

注意：有向面积可以为负值！

叉积实现

定义**point**结构体

```
typedef struct point{double x,y;}point;
```

行列式 $\mathbf{a}\times\mathbf{b}=\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$

```
double det(double x1,double y1,doublex2,double y2)
{
        return x1*y2-x2*y1;
}
```

灰太狼拯救大白菜

话说，由于灰太狼抓羊无术，为了和老婆填饱肚子，于是他决定种大白菜吃- -！

可是，冬天就快要到了，如果不采取什么措施，白菜就会冻死…

为此，聪明的灰太狼，发明了一种神奇的东西----“半圆形大棚”！

灰太狼拯救大白菜

这种神奇的东西可以让大白菜在冬天也能够健康成长，可是灰太狼为了防止羊群们来偷吃，将大白菜种的都很分散。

经过仔细测量，灰太狼统计出了所有大白菜的坐标，并且经过三天三夜的计算，灰太狼确定出圆心的坐标和“半圆形大棚”的半径，但是他却不知道最多能拯救多少棵大白菜。

Figure 1a

Figure 1b

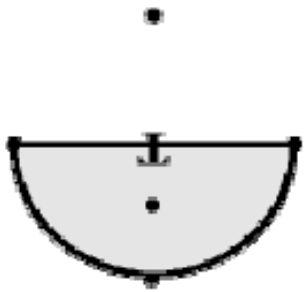
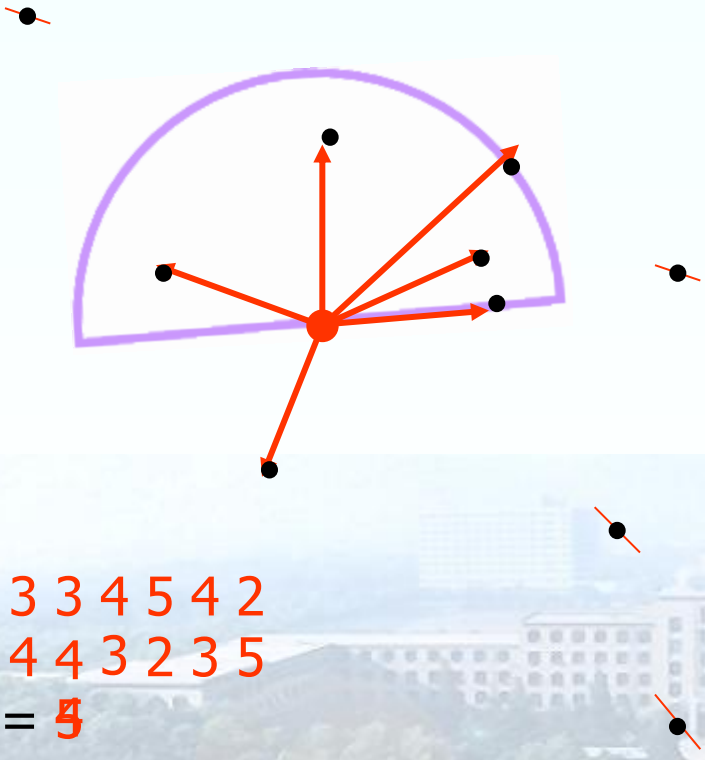


Figure 2

灰太狼拯救大白菜

基本思路：



nl = 3 3 4 5 4 2
nr = 4 4 3 2 3 5
Max = 5

1.到圆心的距离大于半径的点直接排除。

2.以圆心和任意一点确定一 有向线段作为半径位置，分别计数该有向线段左边点的个数(nl)和右边点的个数(nr)。

3.重复步骤2直到所有点都被枚举过。

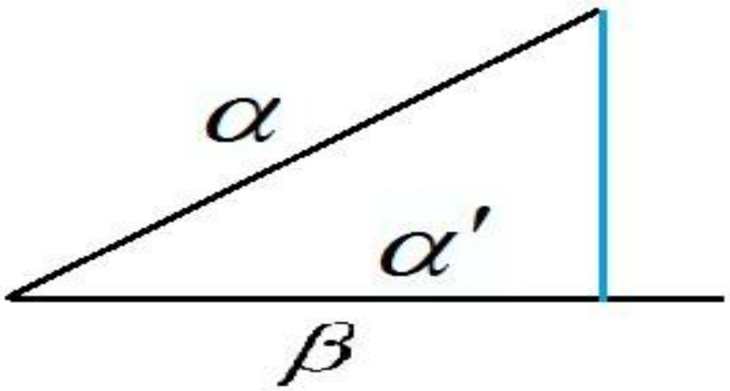
4.枚举过程中出现的最大的nl或nr就是所求的结果。

向量内积

- 定义

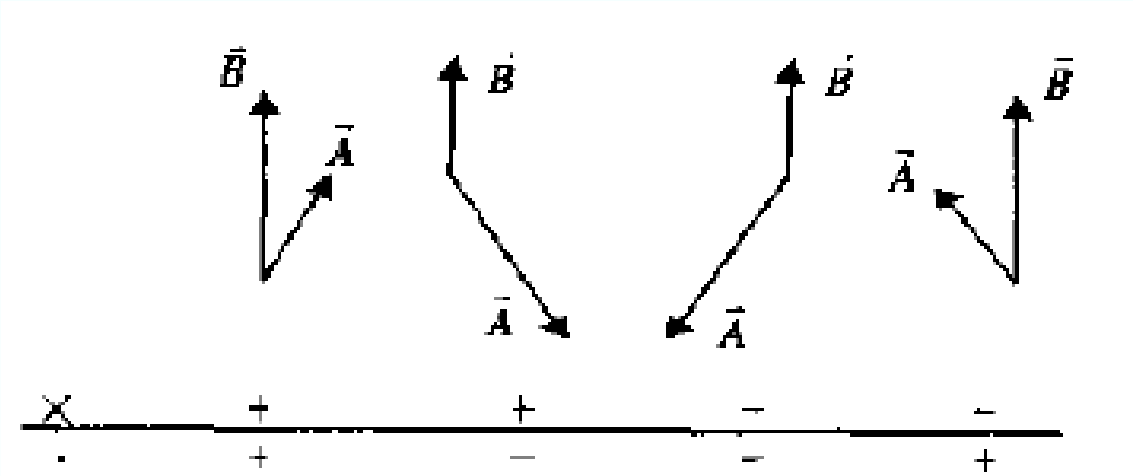
$$\vec{a}\bullet\vec{b}=(x_a,y_a)\bullet(x_b,y_b)=x_a\times x_b+y_a\times y_b$$

- 几何意义：α在β的投影α′与β的长度乘积



点积性质

- 点积可以区分前后方向：



- 前后和投影的关系（联想点积的定义）

线段问题 - 相交

叉积----“左右”判定

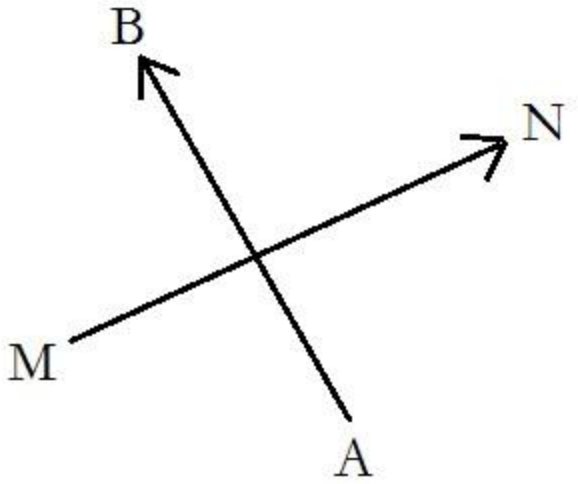
点积----“前后”判定

因此，根据叉积和点积，我们就可以确定线段的位置关系。

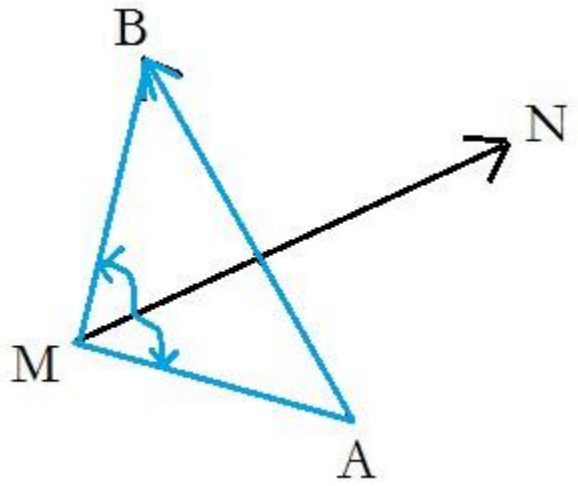
算法分析

- 最为直观的想法：解析几何法
  - 弊端：运算误差大，运算次数多
  - 误差来源：浮点运算
  - 另外，编程难度事实上也很大（区域判断）
- 换个思路试试
  - 为何必须求解直线方程？

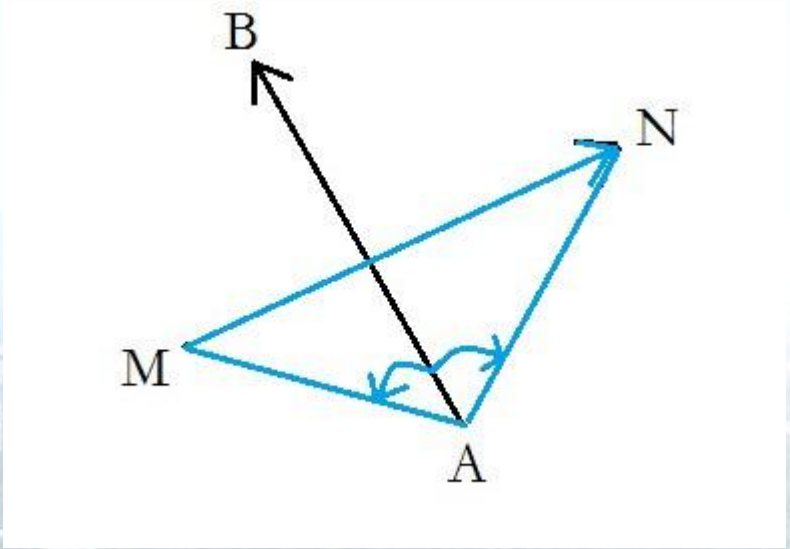
跨立实验



MN、AB为两条相交线段



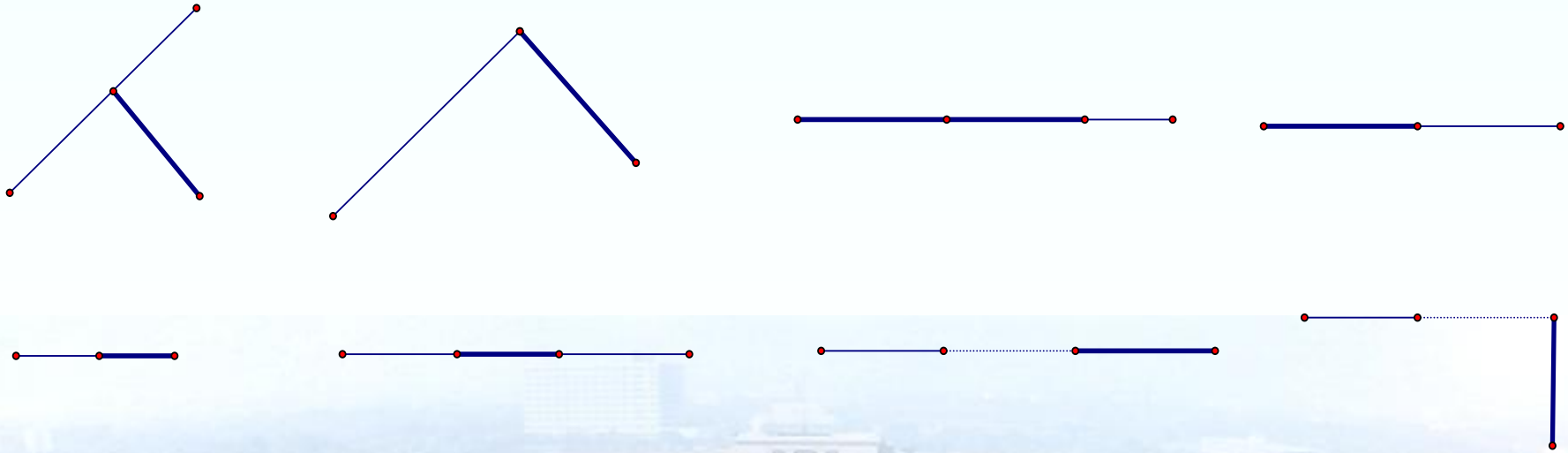
$$(\overrightarrow{MN}\times\overrightarrow{MB})\bullet(\overrightarrow{MN}\times\overrightarrow{MA})<0$$



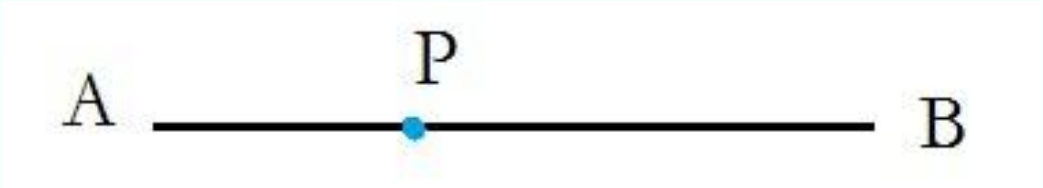
$$(\overrightarrow{AB}\times\overrightarrow{AM})\bullet(\overrightarrow{AB}\times\overrightarrow{AN})<0$$

线段相交判定

- 如果叉积为零如何处理？
  - 以下为叉积为零的八种情形，试判断哪些仍属于相交（自己思考）



定比分点公式



若$A(x_1, y_1), B(x_2, y_2)$，且 $\lambda = \dfrac{|AP|}{|PB|}$

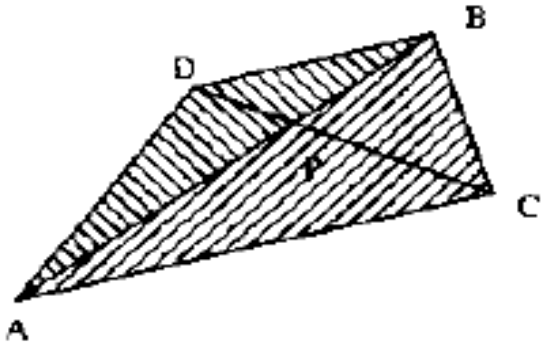
则有：

$$x_P = \frac{x_1 + \lambda x_2}{1 + \lambda}$$

$$y_P = \frac{y_1 + \lambda y_2}{1 + \lambda}$$

叉积求交点

- 即便知道线段已经相交，也不必利用解析几何来求交点
- 面积----是我们的工具

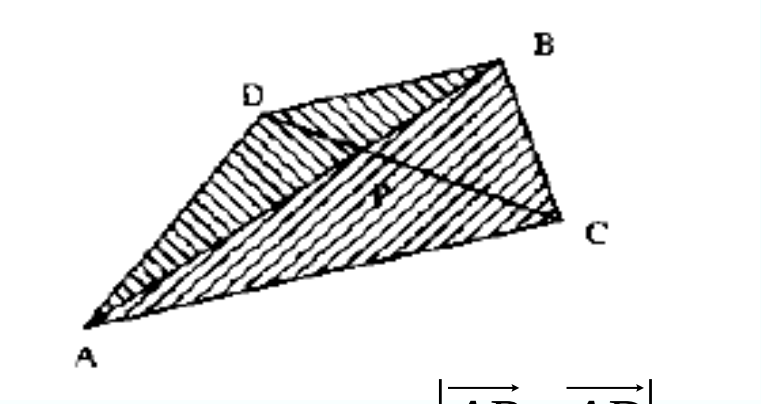


- P分割DC的比值和面积的关系：

$$\frac{|DP|}{|CP|}=\frac{S_{\triangle ABD}}{S_{\triangle ABC}}=\frac{\left|\overrightarrow{AD}\times\overrightarrow{AB}\right|}{\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|}$$

叉积求交点

- 接下来就是定比分点求坐标了：



$$\frac{|DP|}{|CP|}=\frac{S_{\triangle ABD}}{S_{\triangle ABC}}=\frac{\left|\overrightarrow{AB}\times\overrightarrow{AD}\right|}{\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|}$$

$$x_p=\frac{S_{\triangle ABD}\bullet x_C+S_{\triangle ABC}\bullet x_D}{S_{\triangle ABD}+S_{\triangle ABC}}=\frac{\left|\overrightarrow{AB}\times\overrightarrow{AD}\right|\bullet x_C+\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|\bullet x_D}{\left|\overrightarrow{AD}\times\overrightarrow{AB}\right|+\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|}$$

$$y_p=\frac{S_{\triangle ABD}\bullet y_C+S_{\triangle ABC}\bullet y_D}{S_{\triangle ABD}+S_{\triangle ABC}}=\frac{\left|\overrightarrow{AB}\times\overrightarrow{AD}\right|\bullet y_C+\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|\bullet y_D}{\left|\overrightarrow{AD}\times\overrightarrow{AB}\right|+\left|\overrightarrow{AC}\times\overrightarrow{AB}\right|}$$

注意误差问题

计算机在运行浮点运算时，往往会出现误差，比如本来结果为0，但是可能计算出来的结果却是0.00000000001。

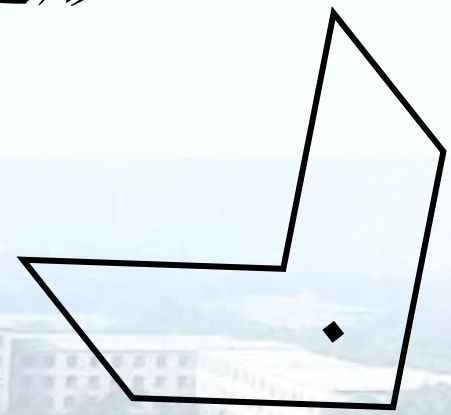
如果直接使用if(res==0)来判断，则显然出错！

那么

可用if(fabs(res)<eps)来判断（eps为精度，可以设置eps=1E-8）

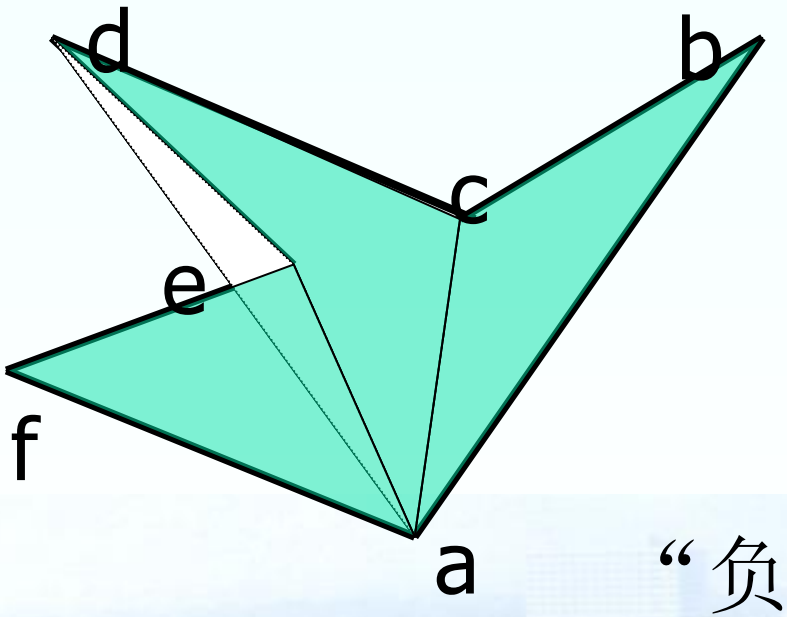
多边形问题

- 求多边形的面积。
  - 前面已经讲过，两向量的**叉积的几何意义**是以这两个向量为邻边的平行四边形的有向面积，我们可以利用这一点来求简单多边形的**面积**。
  - 所谓简单多边形就是任何不相邻的两条边都没有交点，包括凸多边形和凹多边形。



多边形面积

求下面多边形的面积，已知个顶点的坐标。

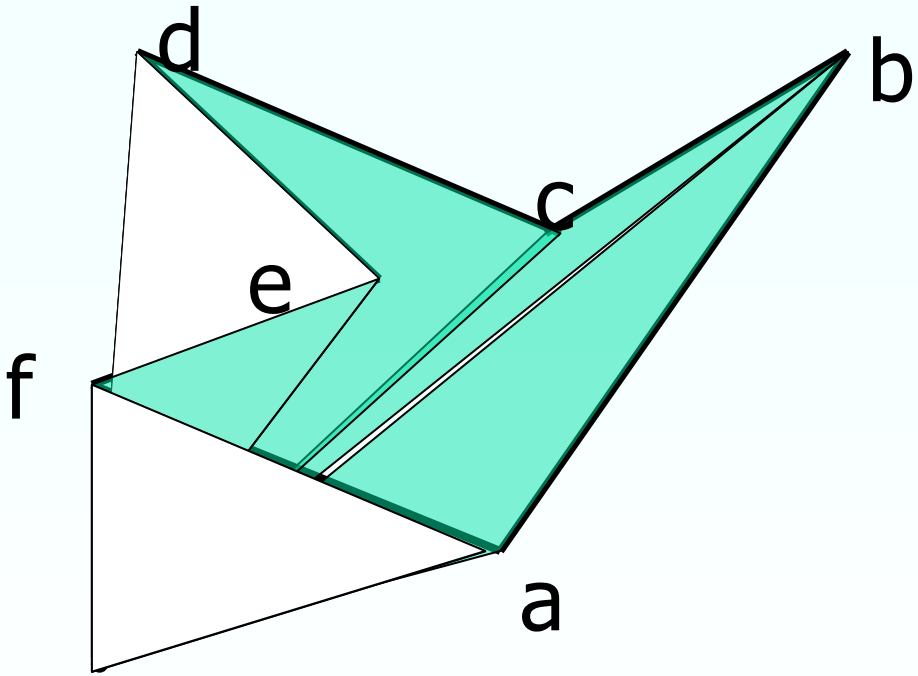


注意：

在引入叉积后，面积就可以为负值，就有了“负面积”的概念。

“负面积”方便了我们的运算

多边形面积



0

$$A=\frac{1}{2}\sum_{i=1}^{n}\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix},(x_{n+1}=x_1,y_{n+1}=y_1)$$

多边形面积

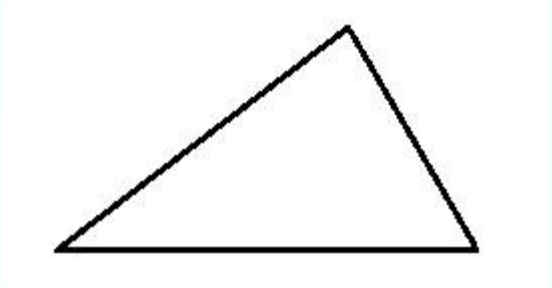
```
double area(point p[],int n)
{
        double s=0;
        int i;

        p[n].x=p[0].x;
        p[n].y=p[0].y;

        for(i=0;i<n;i++)
                s+=det(p[i].x,p[i].y,p[i+1].x,p[i+1].y);

        return fabs(s/2.0);
}
```

多边形重心

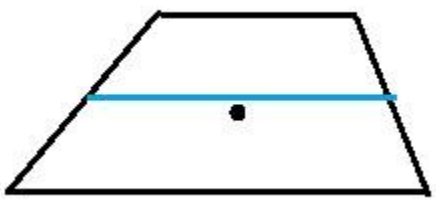


对于三角形的重心

$$\left(\frac{x_1+x_2+x_3}{3},\frac{y_1+y_2+y_3}{3}\right)$$

那么

推广至多边形，猜测为 $\left(\frac{\sum_{i=1}^{N}x_i}{N},\frac{\sum_{i=1}^{N}y_i}{N}\right)$？



不妨假设一个梯形，那么根据上式，则它的重心一定在中位线上。但是，梯形的下底边大于上底边，所以重心一定在中位线之下！

因此，**猜测有误！**

多边形重心

加权平均

将多边形拆分为N个三角形，分别求其重心和面积，可以想象，原来的质量均匀分布在内部的区域上，而现在质量仅分布在这N个重心点上（等价变换），这时就可以利用刚才猜想的公式了。

多边形重心公式

（有兴趣的同学，请自己推导下）

$$C_x = \frac{1}{6A}\sum_{i=1}^{N}(x_i + x_{i+1})\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$$

$$C_y = \frac{1}{6A}\sum_{i=1}^{N}(y_i + y_{i+1})\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \end{vmatrix}$$

多边形重心

```
point cen_gravity(point p[],int n)
{
        point pp;                   //pp is the center of gravity
        int i;
        double x=0,y=0,s,temp;
        p[n].x=p[0].x;
        p[n].y=p[0].y;

        for(i=0;i<n;i++)
        {
                temp=det(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
                x+=(p[i].x+p[i+1].x)*temp;
                y+=(p[i].y+p[i+1].y)*temp;
        }
        s=area(p,n);
        pp.x=x/(6.0*s);
        pp.y=y/(6.0*s);
        return pp;
}
```

点在多边形内外判断

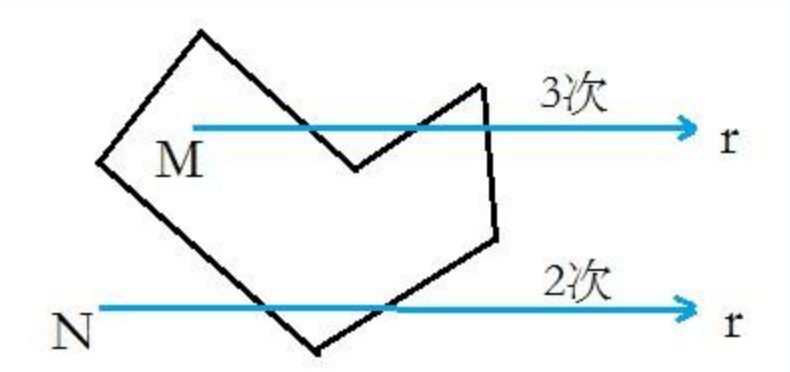
- 一般而言，该问题有射线法和转角法两种处理手段
  - 射线法：从这个点出发引向无穷远点一条直线，根据交点情况确定点的位置
  - 转角法：计算多边形每条边的转角，若最后相消为0则在外部，否则在内部

特点

射线法：特殊情况不易处理

转角法：三角运算时间开销大

射线法----点在多边形内外判断



可根据射线与多边形相交次数的奇偶性判断内外

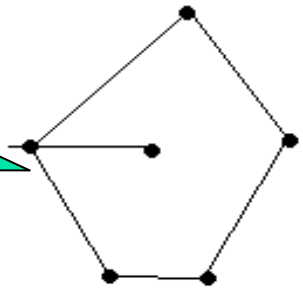
结论

奇数次在内，偶数次在外

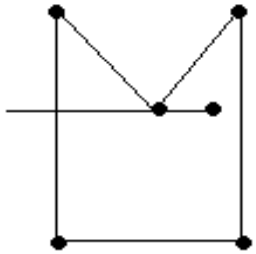
射线法

- 特殊情况：

交点只能计算一个



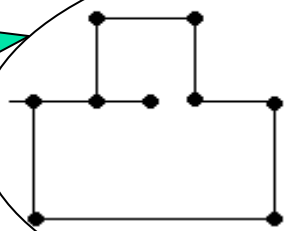
图a



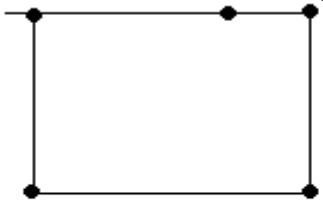
图b

L和多边形顶点的交点不应被计算

这条边应该被忽略不计



图c



图d

伪代码

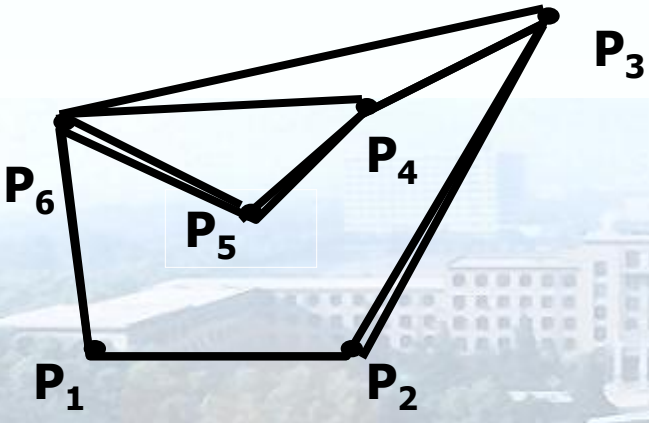
- count ← 0;
- 以P为端点，作从右向左的射线L;
- //设P'的纵坐标和P相同，横坐标为正无穷大（很大的一个正数），则P和
- //P'就确定了射线L。
- for 多边形的每条边s
- do if P在边s上
- then return true;
- if s不是水平的
- then if s的一个端点在L上
- if 该端点是s两端点中纵坐标较大的端点
- then count ← count+1
- else if s和L相交
- then count ← count+1;
- if count mod 2 = 1
- then return true;
- else return false;

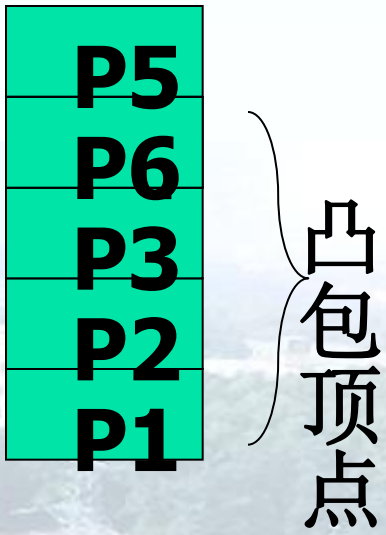
凸包算法

- 现在已经被证明凸包算法的时间复杂度下界是O(n*logn)
- 但是当凸包的顶点数h也被考虑进去的话，Krikpatrick和Seidel的剪枝搜索算法可以达到O(n*logh)，在渐进意义下达到最优

Graham-Scan算法

- 试探性凸包
  - 我们尝试从p1(最低点，一定属于凸包)出发，沿着多边形顶点逆时针的顺序，试探性的增长凸包.显然，一个点如果属于凸包，那么它到达下一个点一定需要左转，否则，该点一定不属于凸包。





算法总结：

- Graham-Scan需要一个序，如果输入是平面点集，首先需要对所有的点按极角排序
  - 显然，“极角大小”比较用“左右手”关系比较(差积)，即：BCi的极角比BCj的极角大⇔BCi在BCj左边
- 该算法带有简单的回溯，因此宜用栈实现
  - 栈中存储的是到目前为止的“局部凸包”
  - 如果当前边对于栈顶边右转，就退栈。一直到“局部凸包”完整。

凸包----Graham-Scan算法

伪代码

```
push(p1);push(p2);
i=3;
while i<=n do
        if pi在栈顶边 p_t p_{t-1} 左手方向
                then push(pi) 并且 i++
                else pop ( );
```

极角序

极角坐标系与极角概念（见黑板）

叉积表征“左右关系”，故可以进行极角序排序

```
int PolarCmp(const point &p1,const point &p2)
{
        int u=dcmp(cross(bp,p1,p2));
        return u>0 || (u==0 && dcmp(dirsqr(bp,p1)-dirsqr(bp,p2))<0);
}

void _sort()
{
        sort(p,p+n,PolarCmp);                    //调用STL，利用快排进行排序
}
```

极角序

```
int dcmp(double x) {
        if (x<-eps)
                return -1;
        else
                return (x>eps);
}
double cross (point p0,point p1,point p2)
{// 若返回值大于0，则p0p2在p0p1逆时针方向
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}
double sqr(double x) {
        return x*x;
}
double dirsqr(point p1,point p2){
        return sqr(p1.x-p2.x)+sqr(p1.y-p2.y);
}
```

水平序

除了极角序之外，还有一种更好的排序方法，称之为水平序排序。它很好的解决了共线点问题，详细请参见《算法艺术与信息学竞赛》P394，下面给出代码实现。

```
double cross(point p, point t1, point t2)
{
        return (t1.x - p.x)*(t2.y-p.y) - (t1.y - p.y)*(t2.x-p.x);
}

int cmp(const void* t1, const void* t2)
{
        point *p1 = (point*)t1;
        point *p2 = (point*)t2;
        if(p1->y == p2->y)
                return p1->x - p2->x;
        return p1->y - p2->y;
}
```

水平序

```
void make_bag(){
        int i, j;
        qsort(p,n,sizeof(point),cmp);
        bag[0] = p[0];
        len = 1;
        for(i=1;i<n;i++)
        {
                while(len>=2&&cross(bag[len-2], bag[len-1], p[i])<=0)
                        len--;
                bag[len++] = p[i];
        }

        j = len+1;
        for(i=n-2;i>=0;i--)
        {
                while(len>=j&&cross(bag[len-2], bag[len-1], p[i])<=0
                        len--;
                bag[len++] = p[i];
        }
        len--;
}
```

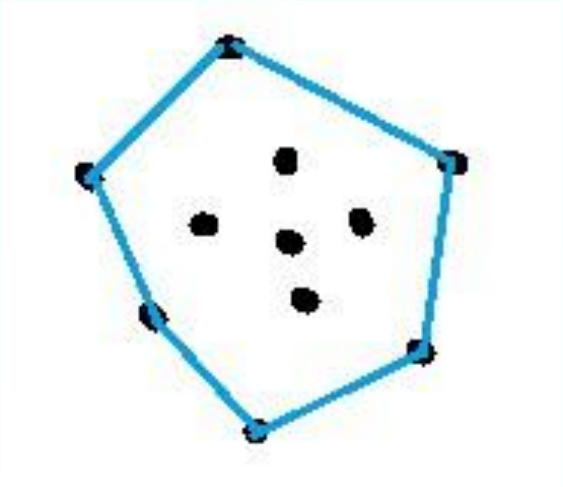
聪明的灰太狼

吃了一冬天的白菜后，灰太狼忍无可忍，于是决定开始新的捕羊计划。

他假扮成小羊，冒着生命危险去青青草原踩点，一不小心他发现，羊窝都是离散分布的，于是乎聪明的灰太狼想到一个绝妙的方法，那就是“栅栏围捕法”----就是在夜黑风高的晚上，在羊窝的最外圈围上一个大栅栏，然后一鼓作气灭掉羊群！

但是由于自己的私房钱刚被老婆没收了大部分，因此这个栅栏圈要尽量的小来减少费用，所以请你想一个好的方法帮帮灰太狼。（羊窝看成一个点，已知各个点的点坐标）

聪明的灰太狼



题目为一个典型的凸包问题

可以先对点进行排序，再Graham-Scan凸包即可。（注意共线点）

凸包

其他求解方法

- MELKMAN 算法
- 分治法
- 增量法
- 半平面交
- 等等

凸包的应用

- 点集直径
- 最小外接矩形
- 等等

一些常见问题

- 离散化问题 **zju1128**
- 多边形费马点 **poj2420**
- 最远点对 **poj2178**
- 半平面交 **poj1279**

需要注意的细节

- 圆周率经常使用acos(-1)来表示
- 角度制和弧度制的转换，C/C++中的三角函数均为弧度制
- 尽量少用除法，开方，三角函数，容易失去精度。用除法时注意除数不为0
- 注意精度问题

谢谢大家!