```
//USTS shiweichun
//based on ECUST luoyongjun
#include <bits/stdc++.h>
using namespace std;
const double pi = acos(-1.0);  //高精度圆周率
const double eps = 1e-8;       //偏差值
const int maxp = 1010;         //点的数量
int sgn(double x){  //判断x是否等于0
    if(fabs(x) < eps)  return 0;
    else return x<0?-1:1;
}
int Dcmp(double x, double y){  //比较两个浮点数：0 相等；-1 小于；1 大于
    if(fabs(x - y) < eps) return 0;
    else return x<y ?-1:1;
}
//---------------平面几何：点和线--------
struct Point{  //定义点和基本运算
    double x,y;
    Point(){}
    Point(double x,double y):x(x),y(y){}
    Point operator + (Point B){return Point(x+B.x,y+B.y);}
    Point operator - (Point B){return Point(x-B.x,y-B.y);}
    Point operator * (double k){return Point(x*k,y*k);}    //长度增大k倍
    Point operator / (double k){return Point(x/k,y/k);}    //长度缩小k倍
    bool operator == (Point B){return sgn(x-B.x)==0 && sgn(y-B.y)==0;}
    bool operator < (Point B){return sgn(x-B.x)<0 || (sgn(x-B.x)==0 && sgn(y-
B.y)<0);} //用于凸包
};
typedef Point Vector;   //定义向量
double Dot(Vector A,Vector B){return A.x*B.x + A.y*B.y;} //点积
double Len(Vector A){return sqrt(Dot(A,A));}   //向量的长度
double Len2(Vector A){return Dot(A,A);}        //向量长度的平方
double Angle(Vector A,Vector B){return acos(Dot(A,B)/Len(A)/Len(B));}   //A与B的
夹角
double Cross(Vector A,Vector B){return A.x*B.y - A.y*B.x;} //叉积
double Area2(Point A, Point B, Point C){return Cross(B-A, C-A);} //三角形ABC面积的2
倍
double Distance(Point A, Point B){return hypot(A.x-B.x,A.y-B.y);}   //两点的距离
double Dist(Point A,Point B){return sqrt((A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-
B.y));}
Vector Normal(Vector A){return Vector(-A.y/ Len(A), A.x/ Len(A));}  //向量A的单位
法向量

bool Parallel(Vector A, Vector B){return sgn(Cross(A,B)) == 0;}//向量平行或重合)

Vector Rotate(Vector A, double rad){  //向量A逆时针旋转rad度
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

struct Line{
    Point p1,p2;//线上的两个点
    Line(){}
    Line(Point p1,Point p2):p1(p1),p2(p2){}
   // Line(Point x,Point y){p1 = x;p2 = y;}
  //  Point(double x,double y):x(x),y(y){}
```

```
//根据一个点和倾斜角 angle 确定直线,0<=angle<pi
    Line(Point p,double angle){
        p1 = p;
        if(sgn(angle - pi/2) == 0){p2 = (p1 + Point(0,1));}
        else{p2 = (p1 + Point(1,tan(angle)));}
    }
//ax+by+c=0
    Line(double a,double b,double c){
        if(sgn(a) == 0){
            p1 = Point(0,-c/b);
            p2 = Point(1,-c/b);
        }
        else if(sgn(b) == 0){
            p1 = Point(-c/a,0);
            p2 = Point(-c/a,1);
        }
        else{
            p1 = Point(0,-c/b);
            p2 = Point(1,(-c-a)/b);
        }
    }
};

typedef Line Segment;   //定义线段，两端点是Point p1,p2

//返回直线倾斜角 0<=angle<pi
double Line_angle(Line v){
    double k = atan2(v.p2.y-v.p1.y, v.p2.x-v.p1.x);
    if(sgn(k) < 0)k += pi;
    if(sgn(k-pi) == 0)k -= pi;
        return k;
}
//点和直线关系:1 在左侧;2 在右侧;0 在直线上
int Point_line_relation(Point p,Line v){
    int c = sgn(Cross(p-v.p1,v.p2-v.p1));
    if(c < 0)return 1;        //1：p在v的左边
    if(c > 0)return 2;        //2：p在v的右边
    return 0;                 //0：p在v上
}
// 点和线段的关系：0 点p不在线段v上；1 点p在线段v上。
bool Point_on_seg(Point p, Line v){
    return sgn(Cross(p-v.p1, v.p2-v.p1)) == 0 && sgn(Dot(p - v.p1,p- v.p2)) <=
0;
}

//两直线关系:0 平行,1 重合,2 相交
int Line_relation(Line v1, Line v2){
    if(sgn(Cross(v1.p2-v1.p1,v2.p2-v2.p1)) == 0){
       if(Point_line_relation(v1.p1,v2)==0)  return 1;//1 重合
       else return 0;//0 平行
    }
    return 2; //2 相交
}

 //点到直线的距离
 double Dis_point_line(Point p, Line v){
    return fabs(Cross(p-v.p1,v.p2-v.p1))/Distance(v.p1,v.p2);
 }
```

```
//点在直线上的投影
Point Point_line_proj(Point p, Line v){
    double k=Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}

//点p对直线v的对称点
Point Point_line_symmetry(Point p, Line v){
    Point q = Point_line_proj(p,v);
    return Point(2*q.x-p.x,2*q.y-p.y);
}

//点到线段的距离
double Dis_point_seg(Point p, Segment v){
    if(sgn(Dot(p- v.p1,v.p2-v.p1))<0 || sgn(Dot(p- v.p2,v.p1-v.p2))<0) //点的投影
不在线段上
        return min(Distance(p,v.p1),Distance(p,v.p2));
    return Dis_point_line(p,v); //点的投影在线段上
 }

 //求两直线ab和cd的交点
//调用前要保证两直线不平行或重合
 Point Cross_point(Point a,Point b,Point c,Point d){ //Line1:ab,  Line2:cd
    double s1 = Cross(b-a,c-a);
    double s2 = Cross(b-a,d-a);  //叉积有正负
    return Point(c.x*s2-d.x*s1,c.y*s2-d.y*s1)/(s2-s1);
 }
 //两线段是否相交：1 相交，0不相交
bool Cross_segment(Point a,Point b,Point c,Point d){//Line1:ab,  Line2:cd
    double c1=Cross(b-a,c-a),c2=Cross(b-a,d-a);
    double d1=Cross(d-c,a-c),d2=Cross(d-c,b-c);
    return sgn(c1)*sgn(c2)<0 && sgn(d1)*sgn(d2)<0;//注意交点是端点的情况不算在内
}
//判断点p是否在线L左边，即点p在线L在外面：
bool OnLeft(Line L,Point p)
{
    return sgn(Cross(L.v,p-L.p))>0;
}
//---------------平面几何：多边形----------------
struct Polygon{
    int n;   //多边形的顶点数
    Point p[maxp];   //多边形的点
    Line v[maxp];    //多边形的边
};
//判断点和任意多边形的关系：3 点上；2 边上；1 内部；0 外部
int Point_in_polygon(Point pt,Point *p,int n){ //点pt，多边形Point *p
    for(int i = 0;i < n;i++){  //点在多边形的顶点上
        if(p[i] == pt)return 3;
    }
    for(int i = 0;i < n;i++){//点在多边形的边上
        Line v=Line(p[i],p[(i+1)%n]);
        if(Point_on_seg(pt,v)) return 2;
    }
    int num = 0;
    for(int i = 0;i < n;i++){
        int j = (i+1)% n;
        int c = sgn(Cross(pt-p[j],p[i]-p[j]));
```

```
        int u = sgn(p[i].y - pt.y);
        int v = sgn(p[j].y - pt.y);
        if(c > 0 && u < 0 && v >=0) num++;
        if(c < 0 && u >=0 && v < 0) num--;
    }
    return num != 0; //1 内部; 0 外部
}

double Polygon_area(Point *p, int n){ //Point *p表示多边形。从原点开始划分三角形
    double area = 0;
    for(int i = 0;i < n;i++)
        area += Cross(p[i],p[(i+1)%n]);
    return area/2;    //面积有正负，不能简单地取绝对值
}

Point Polygon_center(Point *p, int n){ //求多边形重心。Point *p表示多边形。
    Point ans(0,0);
    if(Polygon_area(p,n)==0) return ans;
    for(int i = 0;i < n;i++)
        ans = ans + (p[i]+p[(i+1)%n]) * Cross(p[i],p[(i+1)%n]); //面积有正负
    return ans/Polygon_area(p,n)/6.;
}

//Convex_hull()求凸包。凸包顶点放在ch中，返回值是凸包的顶点数
int Convex_hull(Point *p,int n,Point *ch){
    sort(p,p+n);          //对点排序：按x从小到大排序，如果x相同，按y排序
    n=unique(p,p+n)-p;    //去除重复点
    int v=0;
    //求下凸包。如果p[i]是右拐弯的，这个点不在凸包上，往回退
    for(int i=0;i<n;i++){
        while(v>1 && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2]))<=0)
            v--;
        ch[v++]=p[i];
    }
    int j=v;
    //求上凸包
    for(int i=n-2;i>=0;i--){
        while(v>j && sgn(Cross(ch[v-1]-ch[v-2],p[i]-ch[v-2]))<=0)
            v--;
        ch[v++]=p[i];
    }
    if(n>1) v--;
    return v;   //返回值v是凸包的顶点数
}
//旋转卡壳计算两点的最大距离(凸包的直径)
double rotateCalipers(Point *ch, int n)//ch:凸包的集合 n凸包集合大小
{
    double ans = -INF;
    ch[n] = ch[0];//首尾相连
    int q = 1;
    for (int i = 0; i < n; i++) //枚举边(i,i+1)
    {
        //寻找面积最大的三角形
        while (cross(ch[q] - ch[i + 1], ch[i] - ch[i + 1]) < cross(ch[q + 1] -
ch[i + 1], ch[i] - ch[i + 1]))
        {
            q = (q + 1) % n;
        }
```

```
        //有两条线平行的情况
        ans = max(ans, max(Distance(ch[q] - ch[i]), Distance(ch[q + 1] - ch[i +
1])));

    }
    return ans;
}
 //求半平面交，返回凸多边形
vector<Point> HPI(vector<Line> L){
    int n=L.size();
    sort(L.begin(),L.end());//将所有半平面按照极角排序。
    int first,last;        //指向双端队列的第一个和最后一个元素
    vector<Point> p(n);  //两个相邻半平面的交点
    vector<Line> q(n);   //双端队列
    vector<Point> ans;   //半平面交形成的凸包
    q[first=last=0]=L[0];
    for(int i=1;i<n;i++){
        //情况1：删除尾部的半平面
        while(first<last && !OnLeft(L[i], p[last-1])) last--;
        //情况2：删除首部的半平面：
        while(first<last && !OnLeft(L[i], p[first]))  first++;
        q[++last]=L[i];      //将当前的半平面加入双端队列尾部
        //极角相同的两个半平面，保留左边：
        if(fabs(Cross(q[last].v,q[last-1].v)) < eps){
            last--;
            if(OnLeft(q[last],L[i].p)) q[last]=L[i];
        }
        //计算队列尾部半平面交点：
        if(first<last) p[last-1]=Cross_point(q[last-1],q[last]);
    }
    //情况3：删除队列尾部的无用半平面
    while(first<last && !OnLeft(q[first],p[last-1])) last--;
    if(last-first<=1) return ans;   //空集
    p[last]=Cross_point(q[last],q[first]);  //计算队列首尾部的交点。
    for(int i=first;i<=last;i++)  ans.push_back(p[i]);   //复制。
    return ans;                //返回凸多边形
}
//---------------平面几何：圆----------------
struct Circle{
    Point c;//圆心
    double r;//半径
    Circle(){}
    Circle(Point c,double r):c(c),r(r){}
    Circle(double x,double y,double _r){c=Point(x,y);r = _r;}
};

//点和圆的关系: 0 点在圆内, 1 圆上, 2 圆外.
int Point_circle_relation(Point p, Circle C){
    double dst = Distance(p,C.c);
    if(sgn(dst - C.r) < 0) return 0;  //点在圆内
    if(sgn(dst - C.r) ==0) return 1;   //圆上
    return 2;   //圆外
}

//直线和圆的关系：0 直线在圆内, 1 直线和圆相切, 2 直线在圆外
int Line_circle_relation(Line v,Circle C){
    double dst = Dis_point_line(C.c,v);
    if(sgn(dst-C.r) < 0) return 0; //直线在圆内
```

```
    if(sgn(dst-C.r) ==0) return 1;  //直线和圆相切
    return 2; //直线在圆外
}

//线段和圆的关系：0 线段在圆内，1 线段和圆相切，2 线段在圆外
int Seg_circle_relation(Segment v,Circle C){
    double dst = Dis_point_seg(C.c,v);
    if(sgn(dst-C.r) < 0) return 0; //线段在圆内
    if(sgn(dst-C.r) ==0) return 1;  //线段和圆相切
    return 2;    //线段在圆外
}

//直线和圆的交点    hdu 5572
int Line_cross_circle(Line v,Circle C,Point &pa,Point &pb){//pa，pb是交点。返回值是
交点个数
    if(Line_circle_relation(v, C)==2) return 0;//无交点
    Point q = Point_line_proj(C.c,v);  //圆心在直线上的投影点
    double d = Dis_point_line(C.c,v);   //圆心到直线的距离
    double k = sqrt(C.r*C.r-d*d);   //
    if(sgn(k) == 0){   //1个交点，直线和圆相切
        pa = q;
        pb = q;
        return 1;
    }
    Point n=(v.p2-v.p1)/ Len(v.p2-v.p1);  //单位向量
    pa = q + n*k;
    pb = q - n*k;
    return 2;//2个交点
}
//---最小圆覆盖------

#include <bits/stdc++.h>
using namespace std;
const double eps=1e-8;
const int maxn = 505;
int sgn(double x)
{
    if(fabs(x) < eps)  return 0;
    else return x<0?-1:1;
}
struct Point
{
    double x, y;
};
double Distance(Point A, Point B)
{
    return hypot(A.x-B.x,A.y-B.y);
}
//求三角形abc的外接圆的圆心：
Point circle_center(const Point a, const Point b, const Point c)
{
    Point center;
    double a1=b.x-a.x, b1=b.y-a.y, c1=(a1*a1+b1*b1)/2;
    double a2=c.x-a.x, b2=c.y-a.y, c2=(a2*a2+b2*b2)/2;
    double d =a1*b2-a2*b1;
    center.x =a.x+(c1*b2-c2*b1)/d;
    center.y =a.y+(a1*c2-a2*c1)/d;
    return center;
```

```
}
//求最小覆盖圆，返回圆心c，半径r：
void min_cover_circle(Point *p, int n, Point &c, double &r)
{
    random_shuffle(p, p + n);  //随机函数，打乱所有点。这一步很重要
    c=p[0];
    r=0;                    //从第1个点p0开始。圆心为p0，半径为0
    for(int i=1; i<n; i++)        //扩展所有点
        if(sgn(Distance(p[i],c)-r)>0)  //点pi在圆外部
        {
            c=p[i];
            r=0;                     //重新设置圆心为pi，半径为0
            for(int j=0; j<i; j++)        //重新检查前面所有的点。
                if(sgn(Distance(p[j],c)-r)>0)    //两点定圆
                {
                    c.x=(p[i].x + p[j].x)/2;
                    c.y=(p[i].y + p[j].y)/2;
                    r=Distance(p[j],c);
                    for(int k=0; k<j; k++)
                        if (sgn(Distance(p[k],c)-r)>0) //两点不能定圆，就三点定圆
                        {
                            c=circle_center(p[i],p[j],p[k]);
                            r=Distance(p[i], c);
                        }
                }
        }
}
int main()
{
    int n;                  //点的个数
    Point p[maxn];       //输入点
    Point c;
    double r;  //最小覆盖圆的圆心和半径
    while(~scanf("%d",&n) && n)
    {
        for(int i=0; i<n; i++) scanf("%lf %lf",&p[i].x,&p[i].y);
        min_cover_circle(p,n,c,r);
        printf("%.2f %.2f %.2f\n",c.x,c.y,r);
    }
    return 0;
}

//-------------------三维几何----------------
//三维：点
struct Point3{
    double x,y,z;
    Point3(){}
    Point3(double x,double y,double z):x(x),y(y),z(z){}
    Point3 operator + (Point3 B){return Point3(x+B.x,y+B.y,z+B.z);}
    Point3 operator - (Point3 B){return Point3(x-B.x,y-B.y,z-B.z);}
    Point3 operator * (double k){return Point3(x*k,y*k,z*k);}
    Point3 operator / (double k){return Point3(x/k,y/k,z/k);}
    bool operator == (Point3 B){return sgn(x-B.x)==0 && sgn(y-B.y)==0 && sgn(z-
B.z)==0;}
};
typedef Point3 Vector3;
//点积。和二维点积函数同名。C++允许函数同名。
double Dot(Vector3 A,Vector3 B){return A.x*B.x+A.y*B.y+A.z*B.z;}
```

```
//叉积
Vector3 Cross(Vector3 A,Vector3 B){return Point3(A.y*B.z-A.z*B.y,A.z*B.x-
A.x*B.z,A.x*B.y-A.y*B.x);}
double Len(Vector3 A){return sqrt(Dot(A,A));}   //向量的长度
double Len2(Vector3 A){return Dot(A,A);}        //向量长度的平方
double Distance(Point3 A,Point3 B){
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y)+(A.z-B.z)*(A.z-B.z));
}
double Angle(Vector3 A,Vector3 B){return acos(Dot(A,B)/Len(A)/Len(B));}   //A与B
的夹角
//三维：线
struct Line3{
    Point3 p1,p2;
    Line3(){}
    Line3(Point3 p1,Point3 p2):p1(p1),p2(p2){}
};
typedef Line3 Segment3;   //定义线段，两端点是Point p1,p2

//三角形面积的2倍
double Area2(Point3 A,Point3 B,Point3 C){return Len(Cross(B-A, C-A));}

//三维：点到直线距离
double Dis_point_line(Point3 p, Line3 v){
    return Len(Cross(v.p2-v.p1,p-v.p1))/Distance(v.p1,v.p2);
}

//三维：点在直线上
bool Point_line_relation(Point3 p,Line3 v){
    return sgn( Len(Cross(v.p1-p,v.p2-p))) == 0 && sgn(Dot(v.p1-p,v.p2-p))== 0;
}
//三维：点到线段距离。
double Dis_point_seg(Point3 p, Segment3 v){
    if(sgn(Dot(p- v.p1,v.p2-v.p1)) < 0 || sgn(Dot(p- v.p2,v.p1-v.p2)) < 0)
        return min(Distance(p,v.p1),Distance(p,v.p2));
    return Dis_point_line(p,v);
}
//三维：点 p 在直线上的投影
Point3 Point_line_proj(Point3 p, Line3 v){
    double k=Dot(v.p2-v.p1,p-v.p1)/Len2(v.p2-v.p1);
    return v.p1+(v.p2-v.p1)*k;
}
//三维：平面
struct Plane{
    Point3 p1,p2,p3;//平面上的三个点
    Plane(){}
    Plane(Point3 p1,Point3 p2,Point3 p3):p1(p1),p2(p2),p3(p3){}
};
//平面法向量
Point3 Pvec(Point3 A,Point3 B,Point3 C){ return Cross(B-A,C-A);}
Point3 Pvec(Plane f){ return Cross(f.p2-f.p1,f.p3-f.p1);}
//四点共平面
bool Point_on_plane(Point3 A,Point3 B,Point3 C,Point3 D){
    return sgn(Dot(Pvec(A,B,C),D-A)) == 0;
}
//两平面平行
int Parallel(Plane f1, Plane f2){
    return Len(Cross(Pvec(f1),Pvec(f2))) < eps;
}
```

```
//两平面垂直
int Vertical (Plane f1, Plane f2){
    return sgn(Dot(Pvec(f1),Pvec(f2)))==0;
}
//直线与平面的交点p，返回值是交点个数  ???
int Line_cross_plane(Line3 u,Plane f,Point3 &p){
    Point3 v = Pvec(f);
    double x = Dot(v, u.p2-f.p1);
    double y = Dot(v, u.p1-f.p1);
    double d = x-y;
    if(sgn(x) == 0 && sgn(y) == 0) return -1;//-1：v在f上
    if(sgn(d) == 0)return 0;                    //0：v与f平行
    p = ((u.p1 * x)-(u.p2 * y))/d;              //v与f相交
    return 1;
}

//四面体有向体积*6
double volume4(Point3 A,Point3 B,Point3 C,Point3 D){return Dot(Cross(B-A,C-A),D-
A);}
//---最小球覆盖-------
#include<cstdio>
#include<cmath>
#include<algorithm>
#include<iostream>
using namespace std;
const double sep=1e-6;
const double base=0.98;
const int maxx=1e3+5;
struct node
{
    double x,y,z;
} t[maxx];
int n;
double x,y;
double z;
double dis(node a,node b)
{
    double x=(a.x-b.x);
    double y=(a.y-b.y);
    double z=(a.z-b.z);
    return sqrt(x*x+y*y+z*z);
}
void solve()
{
    double ans=1000000.0,temp=1000.0,maxlen;
    node mid;
    mid.x=mid.y=0;
    mid.z=0;
    int s=0;
    while( temp>sep )
    {
        maxlen=0;
        for( int i=1 ; i<=n ; i++ )
        {
            if( maxlen<dis(mid,t[i]) )
            {
                maxlen=dis(mid,t[i]);
                s=i;
```

```
            }
        }
        ans=min(ans,maxlen);
        mid.x+=(t[s].x-mid.x)/maxlen*temp;
        mid.y+=(t[s].y-mid.y)/maxlen*temp;
        mid.z+=(t[s].z-mid.z)/maxlen*temp;
        temp*=base;
    }
    // printf("%.2lf %.2lf %.2lf %.2lf\n",mid.x,mid.y,mid.z,ans);
    printf("%.5lf\n",ans);
}

int main()
{
    while(scanf("%d",&n)!=EOF)
    {
        if(n==0)break;
        for( int i=1 ; i<=n ; i++ )
        {
            scanf("%lf %lf %lf",&t[i].x,&t[i].y,&t[i].z);
        }
        solve();
    }
    return 0;
}
------------------------
//----三维凸包-------

#include <bits/stdc++.h>
using namespace std;
const int MAXN=1050;
const double eps=1e-8;
struct Point3     //三维：点
{
    double x,y,z;
    Point3() {}
    Point3(double x,double y,double z):x(x),y(y),z(z) {}
    Point3 operator + (Point3 B)
    {
        return Point3(x+B.x,y+B.y,z+B.z);
    }
    Point3 operator - (Point3 B)
    {
        return Point3(x-B.x,y-B.y,z-B.z);
    }
    Point3 operator * (double k)
    {
        return Point3(x*k,y*k,z*k);
    }
    Point3 operator / (double k)
    {
        return Point3(x/k,y/k,z/k);
    }
};
typedef Point3 Vector3;
double Dot(Vector3 A,Vector3 B)
{
    return A.x*B.x+A.y*B.y+A.z*B.z;
```

```
}
Point3 Cross(Vector3 A,Vector3 B)
{
    return Point3(A.y*B.z-A.z*B.y,A.z*B.x-A.x*B.z,A.x*B.y-A.y*B.x);
}
double Len(Vector3 A)
{
    return sqrt(Dot(A,A));   //向量的长度
}
double Area2(Point3 A,Point3 B,Point3 C)
{
    return Len(Cross(B-A, C-A));
}
//四面体有向体积*6:
double volume4(Point3 A,Point3 B,Point3 C,Point3 D)
{
    return Dot(Cross(B-A,C-A),D-A);
}
struct CH3D
{
    struct face
    {
        int a,b,c;   //凸包一个面上的三个点的编号
        bool ok;      //该面是否在最终凸包上
    };
    int n;              //初始顶点数
    Point3 P[MAXN];  //初始顶点
    int num;            //凸包表面的三角形数
    face F[8*MAXN];   //凸包表面的三角形
    int g[MAXN][MAXN]; //凸包表面的三角形//点i到点j是属于哪个面
    //点在面的同向
    double dblcmp(Point3 &p,face &f)
    {
        Point3 m=P[f.b]-P[f.a];
        Point3 n=P[f.c]-P[f.a];
        Point3 t=p-P[f.a];
        return Dot(Cross(m,n),t);
    }
    void deal(int p,int a,int b)
    {
        int f=g[a][b];     //搜索与该边相邻的另一个平面
        face add;
        if(F[f].ok)
        {
            if(dblcmp(P[p],F[f])>eps)
                //如果p点能看到该面f，则继续深度探索f的3条边，以更新新的凸面
                dfs(p,f);
            else
            {
                //p点看不到f面，则p点和a、b点组成一个三角形。
                add.a=b;
                add.b=a;
                add.c=p;
                add.ok=true;
                g[p][b]=g[a][p]=g[b][a]=num;
                F[num++]=add;
            }
        }
```

```
}
void dfs(int p,int now) //维护凸包，如果点p在凸包外更新凸包
{
    F[now].ok=0;
    deal(p,F[now].b,F[now].a);
    deal(p,F[now].c,F[now].b);
    deal(p,F[now].a,F[now].c);
}
bool same(int s,int t) //判断两个面是否为同一面
{
    Point3 &a=P[F[s].a];
    Point3 &b=P[F[s].b];
    Point3 &c=P[F[s].c];
    return fabs(volume4(a,b,c,P[F[t].a]))<eps &&
           fabs(volume4(a,b,c,P[F[t].b]))<eps &&
           fabs(volume4(a,b,c,P[F[t].c]))<eps;
}
//构建三维凸包
void create()
{
    int i,j,tmp;
    face add;
    num=0;
    if(n<4)return;
    //前四个点不共面:
    bool flag=true;
    for(i=1; i<n; i++) //使前两个点不共点
    {
        if(Len(P[0]-P[i])>eps)
        {
            swap(P[1],P[i]);
            flag=false;
            break;
        }
    }
    if(flag)return;
    flag=true;
    //使前三个点不共线
    for(i=2; i<n; i++)
    {
        if(Len(Cross(P[0]-P[1],P[1]-P[i]))>eps)
        {
            swap(P[2],P[i]);
            flag=false;
            break;
        }
    }
    if(flag)return;
    flag=true;
    //使前四个点不共面
    for(int i=3; i<n; i++)
    {
        if(fabs(Dot(Cross(P[0]-P[1], P[1]-P[2]),P[0]-P[i]))>eps)
        {
            swap(P[3],P[i]);
            flag=false;
            break;
        }
```

```
        }
        if(flag)return;
        for(i=0; i<4; i++) //构建初始四面体(4个点为p[0],p[1],p[2],p[3])
        {
            add.a=(i+1)%4;
            add.b=(i+2)%4;
            add.c=(i+3)%4;
            add.ok=true;
            if(dblcmp(P[i],add)>0)swap(add.b,add.c);
            //保证逆时针，即法向量朝外，这样新点才可看到。
            g[add.a][add.b]=g[add.b][add.c]=g[add.c][add.a]=num;
            //逆向的有向边保存
            F[num++]=add;
        }
        for(i=4; i<n; i++) //构建更新凸包
        {
            for(j=0; j<num; j++)
            {
                //判断点是否在当前3维凸包内，i表示当前点,j表示当前面
                if(F[j].ok&&dblcmp(P[i],F[j])>eps)
                {
                    //对当前凸包面进行判断，看是否点能否看到这个面
                    dfs(i,j);//点能看到当前面，更新凸包的面
                    break;
                }
            }
        }
        tmp=num;
        for(i=num=0; i<tmp; i++)
            if(F[i].ok)
                F[num++]=F[i];
    }
    //凸包的表面积
    double area()
    {
        double res=0;
        for(int i=0; i<num; i++)
            res+=Area2(P[F[i].a],P[F[i].b],P[F[i].c]);
        return res/2.0;
    }
    //体积
    double volume()
    {
        double res=0;
        Point3 tmp(0,0,0);
        for(int i=0; i<num; i++)
            res+=volume4(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
        return fabs(res/6.0);
    }
    //表面三角形个数
    int triangle()
    {
        return num;
    }
    //表面多边形个数
    int polygon()
    {
        int i,j,res,flag;
```

```
        for(i=res=0; i<num; i++)
        {
            flag=1;
            for(j=0; j<i; j++)
                if(same(i,j))
                {
                    flag=0;
                    break;
                }
            res+=flag;
        }
        return res;
    }
};
CH3D hull;
int main()
{
    while(scanf("%d",&hull.n)==1)
    {
        for(int i=0; i<hull.n; i++)
            scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);
        hull.create();
        printf("%d\n",hull.polygon());
    }
    return 0;
}
-----------------
```