HDU 4273 Rescue（三维凸包+重心模板） - CSDN博客

Rescue

Time Limit: 2000/1000 MS (Java/Others)   Memory Limit: 32768/32768 K (Java/Others)

Total Submission(s): 382   Accepted Submission(s): 282

Problem Description

I work at NASA outer space rescue team which needs much courage and patient. In daily life, I always receive a lot of mission, and I must complete it right now.
Today, team leader announced me that there is a huge spaceship dropping anchor in the out space, and we should reach there for rescue. As a working principle, at first, we should check whether there are persons living in the spaceship. So we carry a kind of machine called life sensor which can sense the life phenomenon when the distance between the machine and the living is not farther than the sense radius.
I have read the designing paper of the spaceship in advance. It has a form of a convex polyhedron, and we can assume it is isodense. For best control, control center of the whole ship is located at the center of the mass. It is sure that if someone is still alive, he will stay at the control center.
It's unfortunately that I find the door is stocked when I try to enter into the spaceship, so I can only sense the living out of the space ship. Now I have opened the machine and it's time to set the sense radius of it. I wonder the minimal radius of the machine which can allowe me to check whether there are persons living in the spaceship.

Input

There are multiple test cases.
The first line contains an integer n indicating the number of vertices of the polyhedron. (4 <= n <= 100)
Each of the next n lines contains three integers xi, yi, zi, the coordinates of the polyhedron vertices (-10,000 <= xi, yi, zi <= 10,000).
It guaranteed that the given points are vertices of the convex polyhedron, and the polyhedron is non-degenerate.

Output

For each test case, output a float number indicating the minimal radius of the machine. Your answer should accurate up to 0.001.

Sample Input

```
40 0 01 0 00 1 00 0 180 0 00 0 20 2 00 2 22 0 02 0 22 2 02 2 2
```

Sample Output

```
0.1441.000
```

Source

2012 ACM/ICPC Asia Regional Changchun Online

给你一个凸多面体，求重心都面的最小距离。
网上搜的模板，留起来用。。

[cpp]view plaincopy

```cpp
#include
#include
#include
#include
#include
usingnamespace std;
constint MAXN=550;
constdouble eps=1e-8;
struct Point {
double x,y,z;
    Point() {}
    Point(double xx,double yy,double zz):x(xx),y(yy),z(zz) {}
//两向量之差
    Point operator -(const Point p1) {
return Point(x-p1.x,y-p1.y,z-p1.z);
    }
//两向量之和
    Point operator +(const Point p1) {
```

```
return Point(x+p1.x,y+p1.y,z+p1.z);
    }
//叉乘
    Point operator *(const Point p) {
return Point(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);
    }
    Point operator *(double d) {
return Point(x*d,y*d,z*d);
    }
    Point operator / (double d) {
return Point(x/d,y/d,z/d);
    }
//点乘
double  operator ^(Point p) {
return (x*p.x+y*p.y+z*p.z);
    }
};
struct CH3D {
struct face {
//表示凸包一个面上的三个点的编号
int a,b,c;
//表示该面是否属于最终凸包上的面
bool ok;
    };
//初始顶点数
int n;
//初始顶点
    Point P[MAXN];
//凸包表面的三角形数
int num;
//凸包表面的三角形
    face F[8*MAXN];
//凸包表面的三角形
int g[MAXN][MAXN];
//向量长度
double vlen(Point a) {
return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
    }
//叉乘
    Point cross(const Point &a,const Point &b,const Point &c) {
return Point((b.y-a.y)*(c.z-a.z)-(b.z-a.z)*(c.y-a.y),
                     (b.z-a.z)*(c.x-a.x)-(b.x-a.x)*(c.z-a.z),
                     (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x)
                    );
    }
//三角形面积*2
double area(Point a,Point b,Point c) {
return vlen((b-a)*(c-a));
    }
//四面体有向体积*6
double volume(Point a,Point b,Point c,Point d) {
return (b-a)*(c-a)^(d-a);
    }
//正：点在面同向
double dblcmp(Point &p,face &f) {
        Point m=P[f.b]-P[f.a];
        Point n=P[f.c]-P[f.a];
        Point t=p-P[f.a];
return (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)
```

```
               dfs(p,f);
else {
               add.a=b;
               add.b=a;
               add.c=p;//这里注意顺序，要成右手系
               add.ok=true;
               g[p][b]=g[a][p]=g[b][a]=num;
               F[num++]=add;
           }
       }
   }
void dfs(int p,int now) { //递归搜索所有应该从凸包内删除的面
       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) {
       Point &a=P[F[s].a];
       Point &b=P[F[s].b];
       Point &c=P[F[s].c];
return fabs(volume(a,b,c,P[F[t].a]))
               fabs(volume(a,b,c,P[F[t].b]))
               fabs(volume(a,b,c,P[F[t].c]))
   }
//构建三维凸包
void create() {
int i,j,tmp;
       face add;
       num=0;
if(n<4)return;
//**********************************************
//此段是为了保证前四个点不共面
bool flag=true;
for(i=1; i
if(vlen(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
if(vlen((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
if(fabs((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++) {
```

```
            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);
            g[add.a][add.b]=g[add.b][add.c]=g[add.c][add.a]=num;
            F[num++]=add;
        }
for(i=4; i
for(j=0; j
if(F[j].ok&&dblcmp(P[i],F[j])>eps) {
                    dfs(i,j);
break;
                }
            }
        }
        tmp=num;
for(i=num=0; i
if(F[i].ok)
                F[num++]=F[i];
    }
//表面积
double area() {
double res=0;
if(n==3) {
            Point p=cross(P[0],P[1],P[2]);
            res=vlen(p)/2.0;
return res;
        }
for(int i=0; i
            res+=area(P[F[i].a],P[F[i].b],P[F[i].c]);
return res/2.0;
    }
double volume() {
double res=0;
        Point tmp(0,0,0);
for(int i=0; i
            res+=volume(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
            flag=1;
for(j=0; j
if(same(i,j)) {
                    flag=0;
break;
                }
            res+=flag;
        }
return res;
    }
//三维凸包重心
    Point barycenter() {
        Point ans(0,0,0),o(0,0,0);
double all=0;
for(int i=0; i
double vol=volume(o,P[F[i].a],P[F[i].b],P[F[i].c]);
```

```
            ans=ans+(o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0*vol;
            all+=vol;
        }
        ans=ans/all;
return ans;
    }
//点到面的距离
double ptoface(Point p,int i) {
return fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p)/vlen((P[F[i].b]-P[F[i].a])*(P[F[i].c]-P[F[i].a])));
    }
};
CH3D hull;
int main() {
while(scanf("%d",&hull.n)==1) {
for(int i=0; i
            scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);
        }
        hull.create();
        Point p=hull.barycenter();
double ans=1e20;
for(int i=0; i
            ans=min(ans,hull.ptoface(p,i));
        }
        printf("%.3lf\n",ans);
    }
return 0;
}
```