三维凸包

3D Convex Hull

三维凸包（微小扰动版）

Idea：增量法。首先找到一个初始凸包，然后依次考虑剩下的点，若点在凸包内则忽略；否则删去这个点能看到的面，保留不能看到的面并加上新的面构成新凸包。（直接遍历所有的面判断是否可见，再遍历所有边判断是否是边界）

ATT：为了避免多点共面的情形，调用前对所有点进行**去重**和**微小扰动**。

Feature：代码简洁明了；不方便求凸包面数等信息（微小扰动会使面数增加）。

Complexity：$O(n^2)$

Code：

```
void addNoise(Point3 &P){ P.x += randeps(), P.y += randeps(), P.z += randeps(); }

struct Face{
    int v[3]; // the index of original points
    Vector3 Normal(Point3 P[]) const { return (P[v[1]] - P[v[0]]) ^ (P[v[2]] - P[v[0]]); }
    bool cansee(Point3 P[], int i){ return (P[i] - P[v[0]]) * Normal(P) > 0; }
};

bool vis[N][N];
vector<Face> ConvexHull3D(Point3 P[], int n){
    // P[] are points after adding noise and deleting multiple points
    memset(vis, 0, sizeof vis);
    vector<Face> cur;
    cur.push_back((Face){{1, 2, 3}});
    cur.push_back((Face){{3, 2, 1}});
    for(int i = 4; i <= n; i++){
        vector<Face> next;
        for(int j = 0; j < cur.size(); j++){ // add non-seen part into new convex hull
```

```
            Face &f = cur[j];
            bool res = f.cansee(P, i);
            if(!res)    next.push_back(f);
            for(int k = 0; k < 3; k++)  vis[f.v[k]][f.v[(k+1)%3]] = res;
        }
        for(int j = 0; j < cur.size(); j++){
            for(int k = 0; k < 3; k++){
                int a = cur[j].v[k], b = cur[j].v[(k+1)%3];
                if(vis[a][b] != vis[b][a] && vis[a][b]) // segment ab is a boundary
                    next.push_back((Face){{a, b, i}}); // add new faces into convex hull
            }
        }
        cur = next;
    }
    return cur;
}

int n, tn;
Point3 p[N], t[N];
double ans;

int main(){
    srand(20010130);
    scanf("%d", &n);
    for(int i = 1; i <= n; i++){
        p[i].read();
        bool same = false;
        for(int j = 1; j <= tn; j++){
            if(p[i] == t[j]){
                same = true;
                break;
            }
        }
        if(!same)   t[++tn] = p[i];
    }
    for(int i = 1; i <= tn; i++)    addNoise(t[i]);
    vector<Face> res = ConvexHull3D(t, tn);
    for(int i = 0; i < res.size(); i++)
        ans += Length( (t[res[i].v[1]] - t[res[i].v[0]]) ^ (t[res[i].v[2]] - t[res[i].v[0]]) ) / 2;
    printf("%.3f\n", ans);
    return 0;
}
```

三维凸包（严谨版）

Idea：仍然是增量法。实现改用 `dfs`。

Featrue：代码相对繁琐，包装在了一个 `struct` 中；可维护的信息多。

Complexity：$O(n^2)$

Code：

```
struct ConvexHull3D{
    struct Face{
        int v[3]; // the index of original points
        bool inres; // is this face on the convex hull
        Vector3 Normal(Point3 P[]) const { return (P[v[1]] - P[v[0]]) ^ (P[v[2]] - P[v[0]]); }
        bool cansee(Point3 P[], int i){ return (P[i] - P[v[0]]) * Normal(P) > 0; }
    };

    int n; // number of original points
    Point3 P[N]; // original points
    Face F[N<<3]; int fid; // store faces on convex hull
    int belong[N][N]; // belong[i][j] store which face is vector (ij) on

    void dfs(int i, int a, int b){
        int f = belong[a][b];
        if(F[f].inres == false)  return;
        if(F[f].cansee(P, i)){
            F[f].inres = false;
            dfs(i, F[f].v[1], F[f].v[0]);
            dfs(i, F[f].v[2], F[f].v[1]);
            dfs(i, F[f].v[0], F[f].v[2]);
        }
        else{
            Face tmp;
            tmp.v[0] = b, tmp.v[1] = a, tmp.v[2] = i;
            tmp.inres = true;
            belong[b][a] = belong[a][i] = belong[i][b] = ++fid;
            F[fid] = tmp;
        }
    }
    void deal(int i, int j){
        F[j].inres = false;
```

```
        dfs(i, F[j].v[1], F[j].v[0]);
        dfs(i, F[j].v[2], F[j].v[1]);
        dfs(i, F[j].v[0], F[j].v[2]);
    }
    void solve(){
        if(n < 4)   return;
        fid = 0;

        //---------------------- get P[1],P[2],P[3],P[4] right ----------------------//
        bool flag = false;
        for(int i = 2; i <= n; i++){
            if(P[i] != P[1]){
                swap(P[i], P[2]);
                flag = true;
                break;
            }
        }
        if(!flag)   return;
        flag = false;
        for(int i = 3; i <= n; i++){
            if(sgn(Length((P[2]-P[1]) ^ (P[i]-P[1]))) != 0){
                swap(P[i], P[3]);
                flag = true;
                break;
            }
        }
        if(!flag)   return;
        flag = false;
        for(int i = 4; i <= n; i++){
            if(sgn(((P[3]-P[1]) ^ (P[2]-P[1])) * (P[i]-P[1])) != 0){
                swap(P[i], P[4]);
                flag = true;
                break;
            }
        }
        if(!flag)   return;

        //------------------------ store P[1],P[2],P[3],P[4] ------------------------//
        Face tmp;
        for(int i = 1; i <= 4; i++){
            tmp.v[0] = i % 4 + 1;
            tmp.v[1] = (i + 1) % 4 + 1;
            tmp.v[2] = (i + 2) % 4 + 1;
            tmp.inres = true;
```

```
            if(tmp.cansee(P, i))    swap(tmp.v[1], tmp.v[2]);
            belong[tmp.v[0]][tmp.v[1]] = belong[tmp.v[1]][tmp.v[2]]
= belong[tmp.v[2]][tmp.v[0]] = ++fid;
            F[fid] = tmp;
        }

        //---------------------------- add in new points ----------
------------------//
        for(int i = 5; i <= n; i++){
            for(int j = 1; j <= fid; j++){
                if(F[j].inres == true && F[j].cansee(P, i)){
                    deal(i, j);
                    break;
                }
            }
        }

        int tid = fid; fid = 0;
        for(int i = 1; i <= tid; i++) if(F[i].inres) F[++fid] =
F[i];
    }


    inline double SurfaceArea(){
        double res = 0;
        for(int i = 1; i <= fid; i++)
            res += TriangleArea(P[F[i].v[0]], P[F[i].v[1]],
P[F[i].v[2]]);
        return res;
    }
    inline double Volume(){
        double res = 0;
        Point3 O(0, 0, 0);
        for(int i = 1; i <= fid; i++)
            res += TetrahedronVolume(O, P[F[i].v[0]], P[F[i].v[1]],
P[F[i].v[2]]);
        return res;
    }
    inline int cntTriangleFaces(){
        return fid;
    }
    bool sameFace(int i, int j){
        return sgn(TetrahedronVolume(P[F[j].v[0]], P[F[i].v[0]],
P[F[i].v[1]], P[F[i].v[2]])) == 0
            && sgn(TetrahedronVolume(P[F[j].v[1]], P[F[i].v[0]],
P[F[i].v[1]], P[F[i].v[2]])) == 0
```

```
            && sgn(TetrahedronVolume(P[F[j].v[2]], P[F[i].v[0]], P[F[i].v[1]], P[F[i].v[2]])) == 0;
    }
    inline int cntPolygonFaces(){
        int res = 0;
        for(int i = 1; i <= fid; i++){
            bool same = false;
            for(int j = 1; j < i; j++){
                if(sameFace(i, j)){
                    same = true;
                    break;
                }
            }
            if(!same)   res++;
        }
        return res;
    }
};

ConvexHull3D ch;

int main(){
    scanf("%d", &ch.n);
    for(int i = 1; i <= ch.n; i++)  ch.P[i].read();
    ch.solve();
    printf("%.3f\n", ch.SurfaceArea());
    return 0;
}
```