二维

基本定义

点

```
typedef double dbl;
const dbl pi = acos(-1), eps = 1e-8;
int sgn(dbl x) { return x < -eps ? -1 : x > eps; }

struct vec { dbl x, y; };
vec operator+(vec v1, vec v2) { return { v1.x + v2.x, v1.y + v2.y }; }
vec operator-(vec v1, vec v2) { return { v1.x - v2.x, v1.y - v2.y }; }
dbl operator*(vec v1, vec v2) { return v1.x * v2.x + v1.y * v2.y; }
dbl operator^(vec v1, vec v2) { return v1.x * v2.y - v1.y * v2.x; }
vec operator*(vec v, dbl k) { return { v.x * k, v.y * k }; }
vec operator/(vec v, dbl k) { return { v.x / k, v.y / k }; }

bool operator<(vec v1, vec v2) { return v1.x == v2.x ? v1.y < v2.y : v1.x <
v2.x; }
bool operator==(vec v1, vec v2) { return v1.x == v2.x && v1.y == v2.y; }
bool operator>(vec v1, vec v2) { return v2 < v1; }

dbl dot(vec v0, vec v1, vec v2) { return (v1 - v0) * (v2 - v0); }
dbl crx(vec v0, vec v1, vec v2) { return (v1 - v0) ^ (v2 - v0); }
dbl len(vec v) { return hypot(v.x, v.y); }
dbl len2(vec v) { return v * v; }
//  方位角，范围在[0,2pi)
dbl arg(vec v) { dbl r = atan2(v.y, v.x); return r < 0? 2 * pi + r : r; }
//  归一化
vec unif(vec v) { return v / len(v); }
//  方位角为f的单位向量
vec univ(dbl f) { return { cos(f), sin(f) }; }
//  将p绕原点旋转f度
vec rot(vec p, dbl f) { return { cos(f)*p.x-sin(f)*p.y, sin(f)*p.x+cos(f)*p.y };
}
//  将p绕点o旋转f度
vec rot(vec o, vec p, dbl f) { return o + rot(p - o, f); }
//  将p绕原点逆时针旋转pi/2度
vec t90(vec p) { return { -p.y, p.x }; }
```

直线

```
struct line { vec p, v; };
//  求p在l上的投影
vec proj(line l, vec p) { return p + l.v * ((l.p - p) * l.v / len2(l.v)); }
//  求p关于o的对称点
vec refl(vec o, vec p) { return o + o - p; }
//  求p关于l的对称点
vec refl(line l, vec p) { return refl(proj(l, p), p); }
//  求l1和l2的交点，平行时结果未定义
vec litsc(line l1, line l2) { return l2.p+l2.v*((l1.v^(l2.p-l1.p))/(l2.v^l1.v));
}
//  求p到l的距离，将fabs去掉后符号为正则p在l的右侧
dbl lpdis(line l, vec p) { return fabs(crx(l.p, p, l.p + l.v)) / len(l.v); }
```

线段

```
struct seg { vec p1, p2; };
bool onseg(seg s, vec p){return!sgn(crx(p,s.p1,s.p2))&&sgn(dot(p, s.p1,
s.p2))==-1;}

//  0为不相交，1为严格相交，2表示交点为某线段端点，3为线段平行且部分重合
int sitsc(seg s1, seg s2) {
    vec p1 = s1.p1, p2 = s1.p2, q1 = s2.p1, q2 = s2.p2;
    if (max(p1.x,p2.x)<min(q1.x,q2.x)||min(p1.x,p2.x)>max(q1.x,q2.x)) return 0;
    if (max(p1.y,p2.y)<min(q1.y,q2.y)||min(p1.y,p2.y)>max(q1.y,q2.y)) return 0;
    dbl x=crx(p2,p1,q1),y=crx(p2,p1,q2),z=crx(q2,q1,p1),w=crx(q2,q1,p2);
    if (sgn(x)==0&&sgn(y)==0) return 3;
    if (sgn(x)*sgn(y)<0&&sgn(z)*sgn(w)<0) return 1;
    if (sgn(x)*sgn(y)<=0&&sgn(z)*sgn(w)<=0) return 2;
    return 0;
}

//  求p到线段s的距离
dbl spdis(seg s, vec p) {
    if (dot(s.p1, s.p2, p) < eps) return len(p - s.p1);
    if (dot(s.p2, s.p1, p) < eps) return len(p - s.p2);
    return fabs(crx(p, s.p1, s.p2)) / len(s.p1 - s.p2);
}
```

圆

```
struct cir { vec o; dbl r; };
//  求两个圆相交面积
dbl ccarea(cir c1, cir c2) {
    if (c1.r > c2.r) swap(c1, c2);
    dbl d = len(c1.o - c2.o);
    if (sgn(d - (c1.r + c2.r)) >=0) return 0;
    if (sgn(d - abs(c1.r - c2.r)) <= 0) {
        dbl r = min(c1.r, c2.r);
        return r * r * pi;
    }
    dbl x = (d * d + c1.r * c1.r - c2.r * c2.r) / (2 * d);
    dbl t1 = acos(x / c1.r), t2 = acos((d - x) / c2.r);
    return c1.r * c1.r * t1 + c2.r * c2.r * t2 - d * c1.r * sin(t1);
}

//  求圆与直线的交点，没有返回false
```

```
bool clitsc(cir c, line l, vec& p1, vec& p2) {
    dbl x = l.v * (l.p - c.o), y = l.v * l.v;
    dbl d = x * x - y * ((l.p - c.o) * (l.p - c.o) - c.r * c.r);
    if (sgn(d) == -1) return false; d = max(d, (dbl)0);
    vec p = l.p - l.v * (x / y), w = l.v * (sqrt(d) / y);
    p1 = p + w; p2 = p - w; return true;
}

//  求圆与圆的交点，没有返回false
bool ccitsc(cir c1, cir c2, vec& p1, vec& p2) {
    dbl s1 = len(c1.o - c2.o);
    if (sgn(s1 - c1.r - c2.r) == 1 || sgn(s1 - abs(c1.r - c2.r)) == -1) return
false;
    dbl s2 = (c1.r * c1.r - c2.r * c2.r) / s1, a = (s1 + s2) / 2, b = (s1 - s2)
/ 2;
    vec o = (c2.o - c1.o) * (a / (a + b)) + c1.o;
    vec d = t90(unif(c2.o - c1.o)) * sqrt(c1.r * c1.r - a * a);
    p1 = o + d; p2 = o - d; return true;
}
//  求点到圆的切线，没有返回false
bool cptan(cir c, vec p, vec& p1, vec& p2) {
    dbl x = (p-c.o)*(p-c.o), y=x-c.r*c.r;
    if (y < eps) return false;
    vec o = (p-c.o)*(c.r*c.r/x);
    vec d =t90((p-c.o)*(-c.r*sqrt(y)/x));
    o = o + c.o; p1 = o + d; p2 = o - d;
    return true;
}
//  求两个圆的外侧公切线，没有返回false
bool ccetan(cir c1, cir c2, line& l1, line& l2) {
    //  assert(c1 != c2)
    if (!sgn(c1.r - c2.r)) {
        vec v = t90(unif(c2.o - c1.o) * c1.r);
        l1 = { c1.o + v, c2.o - c1.o };
        l2 = { c1.o - v, c2.o - c1.o };
        return true;
    }
    else {
        vec p = (c2.o*c1.r-c1.o*c2.r) / (c1.r-c2.r);
        vec p1, p2, q1, q2;
        if (cptan(c1,p,p1,p2)&&cptan(c2,p,q1,q2)) {
            if (c1.r < c2.r) swap(p1, p2), swap(q1, q2);
            l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
            return true;
        }
    }
    return false;
}
//  求两个圆的内侧公切线，没有返回false
bool ccitan(cir c1, cir c2, line& l1, line& l2) {
    vec p = (c1.o * c2.r + c2.o * c1.r) / (c1.r + c2.r);
    vec p1, p2, q1, q2;
    if (cptan(c1, p, p1, p2) && cptan(c2, p, q1, q2)) {
        l1 = { p1, q1 - p1 }; l2 = { p2, q2 - p2 };
        return true;
    }
    return false;
}
```

三角形

```
//  内切圆圆心
vec incenter(vec a, vec b, vec c) {
    dbl d1 = len(b-c),d2=len(c-a),d3=len(a-b),
        s = fabs(crx(a,b,c));
    return (1/(d1+d2+d3))*(d1*a+d2*b+d3*c);
}
//  外接圆圆心
vec circumcenter(vec a, vec b, vec c) {
    b=b-a; c=c-a; dbl d1 = b*b, d2 = c*c, d = 2*(b^c);
    return a - (1/d)*vec{b.y*d2-c.y*d1,c.x*d1-b.x*d2};
}
```

多边形

询问点在多边形内

```

```

多边形重心

```

```

半平面交

```
bool judge(line l0, line l1, line l2) { return sgn((litsc(l1, l2)-
l0.p)^l0.v)==1; }
int halfplane_intersection(line* lv, int n, vec* pv) {
    static pair<pair<dbl,dbl>, int> a[N];
    for (int i = 1; i <= n; ++i)
        a[i] = { { arg(lv[i].v), lv[i].p*univ(arg(lv[i].v)-pi/2) }, i };
    sort(a + 1, a + n + 1);
    static int b[N], q[N]; int w = 0, l = 1, r = 0;
    for (int i = 1; i <= n; ++i)
        if (i == 1 || sgn(a[i].first.first-a[i-1].first.first))
            b[++w] = a[i].second;
    for (int i = 1; i <= w; ++i) {
        while (l<r&&judge(lv[b[i]],lv[q[r]],lv[q[r-1]]))--r;
        while (l<r&&judge(lv[b[i]],lv[q[l]],lv[q[l+1]]))++l;
        q[++r]=b[i];
    }
    while(l<r&&judge(lv[q[l]],lv[q[r]],lv[q[r-1]]))--r;
    while(l<r&&judge(lv[q[r]],lv[q[l]],lv[q[l+1]]))++l;
    if (r <= l + 1) return 0;
    int m = 0; q[r+1]=q[l];
    for (int i = l; i <= r; ++i)
        pv[++m]=litsc(lv[q[i]],lv[q[i+1]]);
    return m;
}
```

极角分类

将整点按极角分类。

`op[i]`：点`i`的对称点编号

`id[i]`：点`i`的方位角编号

`lb[c]-rb[c]`：极角编号为`c`的所有点在极角序内的范围，左闭右开。

`cnt`：不同极角个数

`pos[i]`：极角序为`i`的点编号

`rk[i]`：编号为`i`的点的极角序

```
namespace azimuth {

int n, m, w; vec qv[N]; bool flg[N];
int op[N], id[N], pos[N], rk[N];
int lb[N], rb[N], cnt;
bool upr(int i) { return qv[i].y > 0 || (qv[i].y == 0 && qv[i].x > 0); }
bool cmp(int i, int j) { return (qv[i] ^ qv[j]) > 0; }

void build(vec* pv, int n_) {
    n = n_; m = 2 * n; cnt = 0; iota(pos, pos + m, 0);
    copy_n(pv, n, qv); copy_n(pv, n, qv + n);
    for (int i = 0; i != n; ++i) qv[n + i] = vec{0,0} - qv[n + i];
    w = partition(pos, pos + m, upr) - pos;
    sort(pos, pos + w, cmp); sort(pos + w, pos + m, cmp);
    for (int i = 0; i != m; ++i) rk[pos[i]] = i;
    for (int l = 0, r = 0; l != m; l = r) {
        while (r != m && (qv[pos[l]]*qv[pos[r]]>0) && !(qv[pos[l]]^qv[pos[r]]))
++r;
         for (int i = l; i != r; ++i) id[pos[i]] = cnt; lb[cnt] = l; rb[cnt] =
r; ++cnt;
    }
    copy_n(qv, m, qv + m); copy_n(pos, m, pos + m); copy_n(rk, m, rk + m);
    copy_n(id, m, id + m); copy_n(lb, cnt, lb + cnt); copy_n(rb, cnt, rb + cnt);
    fill_n(flg, n, 1); fill_n(flg + m, n, 1);
    for (int i = m; i != 2 * m; ++i) rk[i] += m, pos[i] += m, id[i] += cnt;
    for (int i = cnt; i != 2 * cnt; ++i) lb[i] += m, rb[i] += m;
    for (int i = 0; i != n; ++i) {
        if (rk[i] < rk[i + n]) op[i] = i+n, op[i + n] = i+2*n, op[i + m] =
i+3*n;
        else op[i + n] = i, op[i] = i+3*n, op[i+3*n]=i+2*n;
    }
}

}
```

例：计算包含原点的三角形个数。

我们枚举每个三元组中极角序最小的那几个点，用全部三元组个数减去不包含原点的三元组。

设当前极角序有`x`个点，相反极角序有`z`个点，当前极角序到相反极角序（顺时针）之间有`y`个点

共3种情况：

1. 极角序最小的有一个点：`x*(cn2(y+z)-cn2(z))`
2. 极角序最小的有两个点：`cn2(x)*(y+z)`
3. 极角序最小的有三个点：`cn3(x)`

因为三点与原点共线且其中两个点在同一边的情况会减重，所以情况1.要减去cn2(z)。

```
int cn2(int n) { return n * (n - 1) / 2; }
int cn3(int n) { return n * (n - 1) * (n - 2) / 6; }
int s[N];

int count() {
    int res = cn3(n);
    for (int i = 0; i != 2 * m; ++i) s[i + 1] = s[i] + flg[pos[i]];
    for (int i = 0; i != cnt; ++i) {
        int l = lb[i], r = rb[i], p = pos[l];
        int j = id[op[p]], l2 = lb[j], r2 = rb[j];
        int x = s[r] - s[l], y = s[l2] - s[r], z = s[r2] - s[l2];
        res -= x * (cn2(y + z) - cn2(z)) + cn2(x) * (y + z) + cn3(x);
    }
    return res;
}
```

凸包

按逆时针输出。

```
int convex_hull(vec* p, int n, vec* c) {
    sort(p + 1, p + n + 1); n = unique(p + 1, p + n + 1) - p - 1;
    int m = 0;
    c[1] = p[++m];
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && sgn(crx(c[m - 1], c[m], p[i])) != 1) m--;
        c[++m] = p[i];
    }
    int t = m;
    for (int i = n - 1; i; --i) {
        while (m > t && sgn(crx(c[m - 1], c[m], p[i])) != 1) m--;
        c[++m] = p[i];
    }
    if (m > 1) m--; c[m + 1] = c[1]; return m;
}
```

闵可夫斯基和

```
//  To be tested
int minkowski_sum(vec* cv1, int n1, vec* cv2, int n2, vec* cv) {
    if (n1 == 1 || n2 == 1) {
        if (n1 == 1) swap(n1, n2), swap(cv1, cv2);
        for (int i = 1; i <= n1; ++i)
            cv[i] = cv1[i] + cv2[1];
        return n1;
    }
    static vec dv1[N], dv2[N], dv;
    cv1[n1 + 1] = cv1[1]; cv2[n2 + 1] = cv2[1];
    for (int i = 1; i <= n1; ++i) dv1[i] = cv1[i + 1] - cv1[i];
```

```
    for (int i = 1; i <= n2; ++i) dv2[i] = cv2[i + 1] - cv2[i];
    int m = 0; cv[++m] = cv1[1] + cv2[1];
    int p1 = 1, p2 = 1;
    while (p1 <= n1 || p2 <= n2) {
        if (p1 <= n1 && p2 <= n2)
            dv = (sgn((dv1[p1])^(dv2[p2]))!=-1?dv1[p1++]:dv2[p2++]);
        else if (p1 <= n1)
            dv = dv1[p1++];
        else
            dv = dv2[p2++];
        while (m > 1 && !sgn((cv[m] - cv[m - 1]) ^ dv)) {
            dv = dv + cv[m] - cv[m - 1];
            m--;
        }
        cv[m + 1] = cv[m] + dv;
        m++;
    }
    if (m > 1) m--; return m;
}
```

动态凸包

```
typedef double dbl;
typedef long long ll;
int sgn(ll x) { return x < 0 ? -1 : x > 0; }

struct vec { ll x, y; };
bool operator<(vec a, vec b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }
bool operator>(vec a, vec b) { return a.x == b.x ? a.y > b.y : a.x > b.x; }
bool operator!=(vec a, vec b) { return a.x != b.x || a.y != b.y; }
bool operator==(vec a, vec b) { return a.x == b.x && a.y == b.y; }
vec operator+(vec a, vec b) { return { a.x + b.x, a.y + b.y }; }
vec operator-(vec a, vec b) { return { a.x - b.x, a.y - b.y }; }
ll operator*(vec a, vec b) { return a.x * b.x + a.y * b.y; }
ll operator^(vec a, vec b) { return a.x * b.y - a.y * b.x; }
vec operator*(vec a, ll k) { return { a.x * k, a.y * k }; }

ll dot(vec o, vec a, vec b) { return (a - o) * (b - o); }
ll crx(vec o, vec a, vec b) { return (a - o) ^ (b - o); }

typedef function<bool(int, int)> cmp_t;

template<class pr>
struct convex_hull_half {
    struct node : vec { node(vec v = vec{}) : vec(v), prv(0), nxt(0) {} int prv,
nxt; };
    struct cmp { cmp_t* p; bool operator()(int i, int j) { return (*p)(i, j); }
};
    vector<node> w; set<int, cmp> s; cmp_t c;
    dbl C; ll S;

    void clear() { w.clear(); s.clear(); w.push_back(node()); }
    vec min() { return w[w[0].nxt]; }
    vec max() { return w[w[0].prv]; }
    auto lower_bound(vec v) { w[0].x = v.x; w[0].y = v.y; return
s.lower_bound(0); }
    convex_hull_half() : s({ &c }), C(0), S(0) { clear(); }
```

```
    int get_pos(vec v, int& lp, int& rp) {
        c = [&](int i, int j) { return pr()(w[i], w[j]); };
        auto it = lower_bound(v);
        lp = it != s.begin() ? *prev(it) : 0;
        rp = it != s.end() ? *it : 0;
        if (!lp && !rp) return 1;
        else if (!lp || !rp) return c(lp|rp,0)||c(0,lp|rp);
        else return sgn(crx(w[lp], w[0], w[rp]));
    }

    int contains(vec v) { int lp, rp; return get_pos(v, lp, rp); }

    bool insert(vec v) {
        int lp, rp;
        if (get_pos(v, lp, rp) != 1) return false;
        for (int j; lp && (j = w[lp].prv); lp = j)
            if (crx(w[j], w[lp], v) <= 0) s.erase(lp); else break;
        for (int j; rp && (j = w[rp].nxt); rp = j)
            if (crx(w[j], w[rp], v) >= 0) s.erase(rp); else break;
        int p = w.size(); w.push_back(node(v));
        for (int i = lp; i != rp; i = w[i].nxt) {
            C -= dis(w[i], w[w[i].nxt]);
            S -= w[i] ^ w[w[i].nxt];
        }
        C += dis(w[lp], w[p]) + dis(w[p], w[rp]);
        S += (w[lp] ^ w[p]) + (w[p] ^ w[rp]);
        s.insert(w[w[p].prv = lp].nxt = w[w[p].nxt = rp].prv = p);
        return true;
    }

    void upd_max(ll& r, vector<vec>& t, vec v, ll x) {
        if (r > x) return;
        else if (r == x) t.push_back(v);
        else t = { v }, r = x;
    }

    void crxmax(vec v, ll& r, vector<vec>& t) {
        c = [&](int i, int) { return w[i].nxt?(w[0]^(w[w[i].nxt]-w[i]))>0:0; };
        int p = *lower_bound(v);
        upd_max(r, t, w[p], v ^ w[p]);
        if (w[p].nxt && (v ^ w[w[p].nxt]) == (v ^ w[p]))
            upd_max(r, t, w[w[p].nxt], v ^ w[w[p].nxt]);
    }

    void dotmax(vec v, ll& r, vector<vec>& t) { return crxmax({ v.y, -v.x }, r,
t); }

    void upd_ltan(vec v, vec tp, vector<vec>& t) {
        if (t.empty() || crx(v, tp, t.front()) < 0)
            t.assign(1, tp);
        else if(crx(v, tp, t.front()) == 0)
            t.push_back(tp);
    }

    void upd_rtan(vec v, vec tp, vector<vec>& t) {
        if (t.empty() || crx(v, tp, t.front()) > 0)
            t.assign(1, tp);
```

```
        else if (crx(v, tp, t.front()) == 0)
            t.push_back(tp);
    }

    void ltan(vec v, vec rb, vector<vec>& t) {
        c = [&](int i, int) { return w[i].nxt?pr()(w[i], rb)&&crx(w[0], w[i],
w[w[i].nxt])>0:0; };
        auto it = lower_bound(v); assert(it != s.end());
        int p = *it; upd_ltan(v, w[p], t);
        if (w[p].nxt) upd_ltan(v, w[w[p].nxt], t);
    }

    void rtan(vec v, vec lb, vector<vec>& t) {
        c = [&](int i, int) { return w[i].nxt?pr()(w[i], lb)||crx(w[0], w[i],
w[w[i].nxt])<0:0; };
        auto it = lower_bound(v); assert(it != s.end());
        int p = *it; upd_rtan(v, w[p], t);
        if (w[p].nxt) upd_rtan(v, w[w[p].nxt], t);
    }

    void tan(vec v, vector<vec>& lt, vector<vec>& rt) {
        if (pr()(v, min())) { rtan(v, min(), rt); return; }
        if (pr()(max(), v)) { ltan(v, max(), lt); return; }
        int lp, rp, res = get_pos(v, lp, rp);
        if (res == -1) return;
        if (res == 0) {
            if (v == w[rp]) {
                rp = w[rp].nxt;
                upd_ltan(v, w[lp], lt);
                upd_ltan(v, v, lt);
                upd_rtan(v, w[rp], rt);
                upd_rtan(v, v, rt);
            }
            else {
                upd_ltan(v, w[lp], lt);
                upd_rtan(v, w[rp], rt);
            }
        }
        else if (res == 1) {
            ltan(v, w[rp], lt);
            rtan(v, w[lp], rt);
        }
    }

    size_t size() { return s.size(); }
};

void unique(vector<vec>& t) {
    sort(t.begin(), t.end());
    t.erase(unique(t.begin(), t.end()), t.end());
}
struct convex_hull {
    convex_hull_half<less<vec>> lwr;
    convex_hull_half<greater<vec>> upr;

    bool insert(vec v) { bool il = lwr.insert(v), iu = upr.insert(v); return il
|| iu; }
    int contains(vec v) { return max(lwr.contains(v), upr.contains(v)); }
```

```
    ll dotmax(vec v, vector<vec>& t) {
        ll res = LLONG_MIN;
        lwr.dotmax(v, res, t);
        upr.dotmax(v, res, t);
        unique(t);
        return res;
    }

    ll crxmax(vec v, vector<vec>& t) {
        ll res = LLONG_MIN;
        lwr.crxmax(v, res, t);
        upr.crxmax(v, res, t);
        unique(t);
        return res;
    }

    void tan(vec v, vector<vec>& lt, vector<vec>& rt) {
        lwr.tan(v, lt, rt);
        upr.tan(v, lt, rt);
        unique(lt);
        unique(rt);
    }

    size_t size() {
        size_t ls = lwr.size(), us = upr.size();
        if (!ls && !us) return 0;
        else if (ls == 1 && us == 1) return 1;
        else return ls + us - 2;
    }

};
```

三维

```
typedef double dbl;
struct vec { dbl x, y, z; };
vec operator+(vec v1, vec v2) { return { v1.x + v2.x, v1.y + v2.y, v1.z + v2.z
}; }
vec operator-(vec v1, vec v2) { return { v1.x - v2.x, v1.y - v2.y, v1.z - v2.z
}; }
dbl operator*(vec v1, vec v2) { return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}
vec operator^(vec v1, vec v2) {
    return { v1.y * v2.z - v1.z * v2.y,
             v1.z * v2.x - v1.x * v2.z,
             v1.x * v2.y - v1.y * v2.x };
}
vec operator*(vec v, dbl k) { return { v.x * k, v.y * k, v.z * k }; }
vec operator/(vec v, dbl k) { return { v.x / k, v.y / k, v.z / k }; }

struct plane {
    dbl a, b, c, d;
    dbl operator()() { return a * a + b * b + c * c; }
    dbl operator()(vec v) { return (a*v.x+b*v.y+c*v.z+d); }
};
dbl dis(plane p, vec v) { return fabs(p(v)) / sqrt(p()); }
```

```
vec sym(plane p, vec v) {
    return v - vec{p.a, p.b, p.c} * 2 * p(v) / p();
}
```

凸包

```
namespace CH3D {

vec p[N]; int n, l[N][N];

typedef pair<int, int> pii;

pii get_first_edge() {
    int u = 0;
    for (int i = 1; i < n; ++i)
        if (tie(p[i].x,p[i].y,p[i].z)<tie(p[u].x,p[u].y,p[u].z)) u = i;
    int v = u ? 0 : 1;
    for (int i = 0; i < n; ++i) {
        if (u == i || v == i) continue;
        vec v1 = p[v] - p[u], v2 = p[i] - p[u], v3 = v2 ^ v1;
        if (v3.z>0||(v3.z==0&&(v3.y<0||(v3.y==0&&(v3.x>0||v2*v2>v1*v1))))) v =
i;
    }
    return { u, v };
}

pii get_next_edge(int u, int v) {
    int w = -1;
    for (int i = 0; i < n; ++i) {
        if (u == i || v == i) continue;
        vec v1 = p[u] - p[v], v2 = p[w] - p[v], v3 = p[i] - p[v];
        ll d1 = (v3 ^ v1) * v2, d2 = (v3 ^ v2) * (v2 ^ v1);
        if (w==-1||d1<0||(d1==0&&(d2>0||(d2==0&&v3*v3>v2*v2)))) w = i;
    }
    l[u][v] = w;
    return { v, w };
}

void build() {
    for (int i = 0; i < n; ++i) fill_n(l[i], n, -1);
    queue<pii> q; q.push(get_first_edge());
    while (!q.empty()) {
        int u, v; tie(u, v) = q.front(); q.pop();
        if (l[u][v] == -1) q.push(get_next_edge(u, v));
        if (l[v][u] == -1) q.push(get_next_edge(v, u));
    }
}

bool f[N][N];
void process() {
    for (int i = 0; i < n; ++i) fill_n(f[i], n, 0);
    double sum = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j || f[i][j] || l[i][j] == -1) continue;
            int u = i, v = j;
            while (!f[u][v]) {
```

```
                f[u][v] = 1;  //  Triangle iuv
                vec t = (p[u] - p[i]) ^ (p[v] - p[i]);

                int w = l[u][v];
                u = v; v = w;
            }
        }
    }
}

}
```

##