计算几何入门

Tony Yin

2021 年 11 月 28 日

1 向量

1.1 相关概念

向量： 既有大小又有方向的量称为向量，记作 $\vec{a}$ 或 $\boldsymbol{a}$。

有向线段： 带方向的线段。用有向线段来直观地表示向量。起点为 A 终点为 B 的有向线段表示的向量，用符号简记为 $\overrightarrow{AB}$.

向量的模： 向量的大小（或长度），用 $|\overrightarrow{AB}|$ 或 $|\boldsymbol{a}|$ 表示。

零向量： 模为 0 的向量。零向量的方向任意。记为：$\vec{0}$ 或 $\mathbf{0}$。

单位向量： 模为 1 的向量称为该方向上的单位向量。

平行向量： 方向相同或相反的两个**非零**向量。规定 $\vec{0}$ 与任意向量平行。$\boldsymbol{a}$ 与 $\boldsymbol{b}$ 平行，记作：$\boldsymbol{a} \parallel \boldsymbol{b}$ 。

共线向量： 与平行向量的定义相同。任一组平行向量都可以平移到同一直线上。

向量的夹角： 已知两个非零向量 $\boldsymbol{a}, \boldsymbol{b}$，作 $\overrightarrow{OA} = \boldsymbol{a}, \overrightarrow{OB} = \boldsymbol{b}$，那么 $\theta = \angle AOB$ 就是向量 $\boldsymbol{a}$ 与向量 $\boldsymbol{b}$ 的夹角。记作：$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$。当 $\theta = \frac{\pi}{2}$ 时，称这两个向量垂直，记作 $\boldsymbol{a} \perp \boldsymbol{b}$。规定 $\theta \in [0, \pi]$。

1.2 向量的加减法

1.2.1 向量加法的三角形法则

对于平面上的任意两个向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$，在平面内任取一点 A，作 $\overrightarrow{AB} = \boldsymbol{a}$，$\overrightarrow{BC} = \boldsymbol{b}$，作向量 $\overrightarrow{AC}$. 称向量 $\overrightarrow{AC}$ 为向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的**和向量**，$\overrightarrow{AB} + \overrightarrow{BC} = \overrightarrow{AC}$.

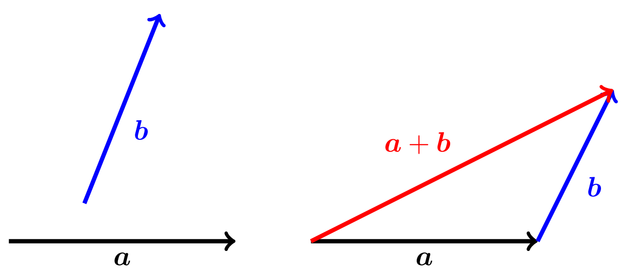

图 1

如图 1，把向量首尾顺次相连，向量的和为第一个向量的起点指向最后一个向量的终点；

1.2.2 向量加法的平行四边形法则

若要求和的两个向量**共起点**，那么它们的和向量为以这两个向量为邻边的平行四边形的对角线，起点为两个向量共有的起点，方向沿平行四边形对角线方向。

2　二维凸包

2.1　相关概念

凸多边形：所有内角大小都在 $[0,\pi]$ 范围内的**简单多边形**。

凸包：在平面上能包含所有给定点的最小凸多边形叫做凸包。可以理解为平面上有若干柱子，用一个橡皮筋套住所有柱子，绷紧后形成的多边形即为凸包。

2.2　求解方法—Andrew 算法

首先把所有点**排序**，以横坐标为第一关键字，纵坐标为第二关键字。

排序后，第一个点和末尾的点，一定在凸包上，容易通过反证法证明。

从左往右看，上下凸壳斜率的**单调性相反**，即所旋转的方向不同，所以要分开求。

我们 **升序枚举**求出下凸壳，然后**降序枚举**求出上凸壳，这样凸包的每条边都是向**逆时针方向**旋转的。

设当前枚举到点 P，即将把其加入凸包；当前栈顶的点为 S_1，栈中第二个点为 S_2.

求凸包时，若 P 与 S_1 构成的新线段是顺时针旋转的，即叉积满足：$\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$，则弹出栈顶，继续检查，直到 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} \geq 0$ 或者栈内仅剩一个元素为止。

若将弹出栈顶元素的条件改为 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} \leq 0$，同时停止条件改为 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} > 0$，则求出的凸包中不存在三点共线，可视情况更改。

2.3　参考代码

```cpp
inline bool check(Point s1, Point s2, Point p) {
    return Vec(s2, s1) * Vec(s1, p) > 0;
}
int Convex_hull_2d(int n, Point *p, Point *ret) {
    sort(p, p + n, less_cmp);
    int top = -1;
    for (int i = 0; i < n; i++) {
        while (top > 0 && !check(ret[top], ret[top - 1], p[i]))
            top--;
        ret[++top] = p[i];
    }
    int k = top;
    for (int i = n - 2; i >= 0; i--) {
        while (top > k && !check(ret[top], ret[top - 1], p[i]))
            top--;
        ret[++top] = p[i];
    }
    return top;
}
double Convex_hull_2d_L(int n, Point *p) {
    Point convex[N];
    int siz = Convex_hull_2d(n, p, convex);
    double ans = dis(convex[0], convex[siz - 1]);
    for (int i = 1; i < siz; i++)
        ans += dis(convex[i - 1], convex[i]);
    return ans;
}
double Convex_hull_2d_S(int n, Point *p) {
    Point convex[N];
```

```
    int siz = Convex_hull_2d(n, p, convex);
    double ans = 0;
    for (int i = 2; i < siz; i++)
        ans += area(convex[0], convex[i - 1], convex[i]);
    return ans;
}
```

3 三维凸包

3.1 相关概念

3.1.1 模长

$|\boldsymbol{a}| = \sqrt{{x_a}^2 + {y_a}^2 + {z_a}^2}$，代表向量长度。

3.1.2 点积

$\boldsymbol{a} \cdot \boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}| \cos\langle \boldsymbol{a}, \boldsymbol{b} \rangle = x_a x_b + y_a y_b + z_a z_b$，结果为标量。

3.1.3 叉积

$\boldsymbol{a} \times \boldsymbol{b} = (y_a z_b - z_a y_b, z_a x_b - x_a z_b, x_a y_b - y_a x_b)$，结果为向量，是 $\boldsymbol{a}, \boldsymbol{b}$ 所在平面的法向量。

3.1.4 法向量

3.2 求解方法—增量法

和求解二维凸包的方法类似，考虑把一个新的点加入，现有的凸包会如何变化。

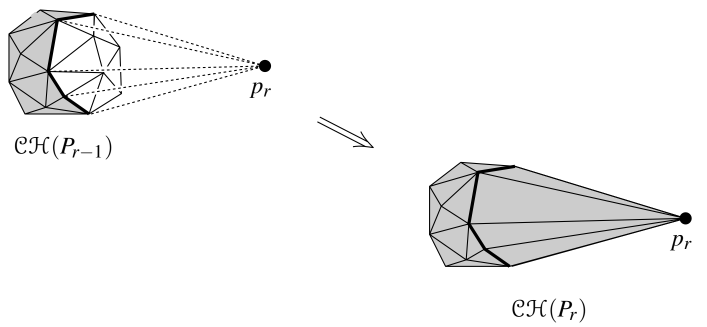

图 2

如图二，把 P_r 加入凸包的时候，把它想象为光源，照向当前已知的凸包。
在新凸包中，保留不会被照到的面，再加上 P_r 与能照到的边缘构成的若干新面，

3.3 参考代码

```
#include <bits/stdc++.h>
using namespace std;
const int N = 110;
const double pi = 3.1415926535898;
const double eps = 1e-10;
inline int fcmp(double x, double y) { //x>y->1, x=y->0, x<y->-1
    if(fabs(x - y) < eps) return 0;
    else return x > y ? 1 : -1;
```

```
}
inline double rd() { return (rand()) % 2 ? eps : -eps; }
struct Point3 {
    double x, y, z;
    Point3(){};
    Point3(double a, double b, double c) : x(a), y(b), z(c) {}
    Point3 operator + (const Point3 &b) {
        return Point3(x + b.x, y + b.y, z + b.z);
    }
    Point3 operator - (const Point3 &b) {
        return Point3(x - b.x, y - b.y, z - b.z);
    }
    Point3 operator * (const Point3 &b) {
        return Point3(y*b.z - z*b.y, z*b.x - x*b.z, x*b.y - y*b.x);
    }
    double operator & (const Point3 &b) {
        return x * b.x + y * b.y + z * b.z;
    }
    bool operator == (const Point3 &b) {
        return fcmp(x, b.x) == 0 && fcmp(y, b.y) == 0 && fcmp(z, b.z) == 0;
    }
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
    void shake() { x += rd(), y += rd(), z += rd(); }
} p[N];
struct plane{
    int v[3]; //逆时针
    plane(){};
    plane(int a, int b, int c) { v[0] = a, v[1] = b, v[2] = c; }
    Point3 normal() {
        return (p[v[1]] - p[v[0]]) * (p[v[2]] - p[v[0]]);
    }
    bool is_above(Point3 A) {
        return (normal() & (A - p[v[0]])) >= 0;
    }
    double area() {
        return normal().len() / 2.0;
    }
};
int Convex_hull_3d(int n, plane *ret) {
    plane tmp[N];
    bool g[N][N];
    for (int i = 0; i < n; i++) p[i].shake();
    int top = -1;
    ret[++top] = plane(0, 1, 2);
    ret[++top] = plane(0, 2, 1);
    for (int i = 3; i < n; i++) {
        int cnt = -1;
        for (int j = 0; j <= top; j++) {
```

```
                bool flag = ret[j].is_above(p[i]);
                if (!flag)
                    tmp[++cnt] = ret[j];
                for (int k = 0; k < 3; k++)
                    g[ret[j].v[k]][ret[j].v[(k + 1) % 3]] = flag;
            }
            for (int j = 0; j <= top; j++) {
                for (int k = 0; k < 3; k++) {
                    int a = ret[j].v[k], b = ret[j].v[(k + 1) % 3];
                    if (g[a][b] && !g[b][a])
                        tmp[++cnt] = plane(a, b, i);
                }
            }
            for (int j = 0; j <= cnt; j++) ret[j] = tmp[j];
            top = cnt;
        }
        return (top + 1);
}
double Convex_hull_area(int n) {
    plane convex[N];
    int siz = Convex_hull_3d(n, convex);
    double ret = 0;
    for (int i = 0; i < siz; i++) ret += convex[i].area();
    return ret;
}
int main() {
    srand(time(NULL));
    int n; cin >> n;
    for (int i = 0; i < n; i++) cin >> p[i].x >> p[i].y >> p[i].z;
    printf("%.6lf\n", Convex_hull_area(n));
    return 0;
}
```