计算几何

一.计算几何边界情况：

1.两直线平行，三点公线，除零

2.将所有点排序后处理，相同x坐标或y坐标的处理顺序

3.判断直线和多边形是否相交时，漏掉了直线恰好通过多边形顶点的情况。

4.判断两个实心物体是否相交时，忘记了其中一个完全在另一个内部

5.在物体移动的模拟类问题中，将初始接触但随后分离的情况误判为碰撞

二.注意浮点误差

(a < 0) -> (a < -eps)

(a <= 0) -> (a < eps)

(a == 0) -> (abs(a) < eps)

const double eps = 1e-8;

const double pi = acos(-1.0);

const int maxn = 100;//多边形类

//计算几何误差修正

//给定一个double类型的数，判断他的符号

/\*

接口：

int cmp(double x);

输入：x,判断符号的数

输出：x的符号，-1表示负数，1表示x是正数，0表示0

\*/

int cmp(double x) {

if (fabs(x) < eps) {

return 0;

}

if (x > 0) return 1;

return -1;

}

//计算几何点类

//计算一个数的平方

inline double sqr(double x) {

return x \* x;

}

struct point {

double x, y;

point() {}

point(double a, double b) : x(a), y(b) {}

void input() {

scanf("%lf%lf", &x, &y);

}

friend point operator + (const point &a, const point &b) {

return point(a.x + b.x, a.y + b.y);

}

friend point operator - (const point &a, const point &b) {

return point(a.x - b.x, a.y - b.y);

}

friend bool operator == (const point &a, const point &b) {

return (cmp(a.x - b.x) == 0 && cmp(a.y - b.y) == 0);

}

friend point operator \* (const point &a, const double &b) {

return point(a.x \* b, a.y \* b);

}

friend point operator \* (const double &a, const point &b) {

return point(b.x \* a, b.y \* a);

}

friend point operator / (const point &a, const double &b) {

return point (a.x / b, a.y / b);

}

double norm() {

return sqrt(sqr(x) + sqr(y));

}

};

//叉积

double det(const point &a, const point &b) {

return a.x \* b.y - a.y \* b.x;

}

//点积

double dot(const point &a, const point &b) {

return a.x \* b.x + a.y \* b.y;

}

//两个点的距离

double dist(const point &a, const point &b) {

return (a - b).norm();

}

//向量绕远点逆时针旋转A(弧度)

point rotate\_point(const point &p, double A) {

double tx = p.x, ty = p.y;

return point(tx \* cos(A) - ty \* sin(A), tx \* sin(A) + ty \* cos(A));

}

//计算几何线段类

struct line {

point a, b;

line() {}

line (point x, point y) : a(x), b(y) {}

};

//用两个点a,b生成的一个线段或直线

line point\_make\_line(const point a, const point b) {

return line(a, b);

}

//求p点到线段st的距离

double dis\_point\_segment(const point p, const point s, const point t, point &cp) {

if (cmp(dot(p - s, t - s)) < 0) return (p - s).norm();

if (cmp(dot(p - t, s - t)) < 0) return (p - t).norm();

return fabs(det(s - p, t - p) / dist(s, t));

}

//求p点到线段st的垂足，保存在cp中

void PointProjLine(const point p, const point s, const point t, point &cp) {

double r = dot((t - s), (p - s)) / dot(t - s, t - s);

cp = s + r \* (t - s);

}

//判断p点是否再线段st上（包括端点）

bool PointOnSegment(point p, point s, point t) {

return cmp(det(p - s, t - s)) == 0 && cmp(dot(p - s, p - t)) <= 0;

}

//判断a和b是否平行

bool parallel(line a, line b) {

return !cmp(det(a.a - a.b, b.a - b.b));

}

//判断a和b(直线)是否相交，如果相交则返回true且交点保存在res中

bool line\_make\_point(line a, line b, point &res) {

if (parallel(a, b)) return false;

double s1 = det(a.a - b.a, b.b - b.a);

double s2 = det(a.b - b.a, b.b - b.a);

res = (s1 \* a.b - s2 \* a.a) / (s1 - s2);

return true;

}

//判断线段a,b是否相交

bool segment\_make\_point(line a, line b, point &res) {

if (parallel(a, b)) {

if (PointOnSegment(a.a, b.a, b.b) || PointOnSegment(a.b, b.a, b.b)) return true;

if (PointOnSegment(b.a, a.a, a.b) || PointOnSegment(b.b, a.a, a.b)) return true;

return false;

}

line\_make\_point(a, b, res);

if (PointOnSegment(res, a.a, a.b) && PointOnSegment(res, b.a, b.b)) return true;

return false;

}

//将直线a沿法向量方向平移距离len得到的直线

line move\_d(line a, const double &len) {

point d = a.b - a.a;

d = d / d.norm();

d = rotate\_point(d, pi / 2);

return line(a.a + d \* len, a.b + d \* len);

}

//计算几何多边形类

struct polygon {

int n;//多边形点数

point a[maxn];//多边形顶点坐标（按顺时针顺序）

polygon() {}

double perimeter() {//计算多边形周长

double sum = 0;

a[n] = a[0];

for (int i = 0; i < n; i ++) sum += (a[i + 1] - a[i]).norm();

return sum;

}

double area() {//计算多边形面积

double sum = 0;

a[n] = a[0];

for (int i = 0; i < n; i ++) sum += det(a[i + 1], a[i]);

return sum / 2.;

}

int Point\_In(point t) {//判断点是否在多边形内部（O(N))

int num = 0, i, d1, d2, k;

a[n] = a[0];

for (i = 0; i < n; i ++) {

if (PointOnSegment(t, a[i], a[i + 1])) return 2;

k = cmp(det(a[i + 1] - a[i], t - a[i]));

d1 = cmp(a[i].y - t.y);

d2 = cmp(a[i + 1].y - t.y);

if (k > 0 && d1 <= 0 && d2 > 0) num ++;

if (k < 0 && d2 <= 0 && d1 > 0) num --;

}

return num != 0;

}

};

//多边形的重心(O(N))

point polygon\_MassCenter(polygon p) {

point ans = point(0, 0);

if (cmp(p.area()) == 0) return ans;

p.a[p.n] = p.a[0];

for (int i = 0; i < p.n; i ++) ans = ans + (p.a[i] + p.a[i + 1]) \* det(p.a[i + 1], p.a[i]);

return ans / p.area() / 6.;

}