三维基础

3D Computational Geometry

```cpp
struct Point3{
    double x, y, z;
    Point3(double x=0, double y=0, double z=0):x(x), y(y), z(z) {}
    void read(){ scanf("%lf%lf%lf", &x, &y, &z); }
};
typedef Point3 Vector3;
Vector3 operator + (Vector3 A, Vector3 B){ return Vector3(A.x+B.x, A.y+B.y, A.z+B.z); }
Vector3 operator - (Vector3 A, Vector3 B){ return Vector3(A.x-B.x, A.y-B.y, A.z-B.z); }
Vector3 operator * (double k, Vector3 A){ return Vector3(A.x * k, A.y * k, A.z * k); }
Vector3 operator * (Vector3 A, double k){ return k * A; }
Vector3 operator / (Vector3 A, double k){ return Vector3(A.x / k, A.y / k, A.z / k); }
bool operator == (Vector3 A, Vector3 B){ return cmp(A.x, B.x) == 0 && cmp(A.y, B.y) == 0 && cmp(A.z, B.z) == 0; }
bool operator != (Vector3 A, Vector3 B){ return !(A == B); }
double operator * (Vector3 A, Vector3 B){ return A.x * B.x + A.y * B.y + A.z * B.z; }
Vector3 operator ^ (Vector3 A, Vector3 B){ return Vector3(A.y*B.z - A.z*B.y, A.z*B.x - A.x*B.z, A.x*B.y - A.y*B.x); }
double Length(Vector3 A){ return sqrt(A * A); }
double Angle(Vector3 A, Vector3 B){ return acos(A * B / Length(A) / Length(B)); }
double ParallelogramArea(Point3 A, Point3 B, Point3 C){ return Length((B - A) ^ (C - A)); }
double ParallelepipedVolume(Point3 A, Point3 B, Point3 C, Point3 D){ return ((B - A) ^ (C - A)) * (D - A); }
double TriangleArea(Point3 A, Point3 B, Point3 C){ return Length((B - A) ^ (C - A)) / 2; }
double TetrahedronVolume(Point3 A, Point3 B, Point3 C, Point3 D){ return ((B - A) ^ (C - A)) * (D - A) / 6.0; }
double DistancePointToPoint(Point3 A, Point3 B){ return Length(A-B); }
bool PointInTriangle(Point3 P, Point3 A, Point3 B, Point3 C){
    double area1 = TriangleArea(P, A, B);
    double area2 = TriangleArea(P, B, C);
    double area3 = TriangleArea(P, C, A);
    return sgn(area1 + area2 + area3 - TriangleArea(A, B, C)) == 0;
}

struct Line3{
    Point3 p;
    Vector3 v;
    Line3() {}
    Line3(Point3 p, Vector3 v):p(p), v(v) {}
    Point3 getPoint(double t){ return Point3(p + v * t); }
};
struct Plane3{
    Point3 p;
    Vector3 v; // normal vector
    Plane3() {}
    Plane3(Point3 p, Vector3 v):p(p), v(v) {}
};

bool PointOnLine(Point3 P, Line3 L){ return sgn(Length((P - L.p) ^ L.v)) == 0; }
bool LineParallel(Line3 L1, Line3 L2){ return sgn(Length(L1.v ^ L2.v)) == 0; }
bool LineSame(Line3 L1, Line3 L2){ return LineParallel(L1, L2) && sgn(Length((L2.p - L1.p) ^ L1.v)) == 0; }
bool LineCoplanar(Line3 L1, Line3 L2){ return sgn((L2.p - L1.p) * (L1.v ^ L2.v)) == 0; }
double DistancePointToLine(Point3 P, Line3 L){ return Length(L.v ^ (P - L.p)) / Length(L.v); }
double DistancePointToSegment(Point3 P, Point3 A, Point3 B){
    if(A == B)  return Length(P - A);
    Vector3 v1 = B - A, v2 = P - A, v3 = P - B;
    if(sgn(v1 * v2) < 0)    return Length(v2);
    else if(sgn(v1 * v3) > 0)   return Length(v3);
    else    return Length(v1 ^ v2) / Length(v1);
}
Point3 PointLineProjection(Point3 P, Line3 L){ return L.p + L.v * ((L.v * (P - L.p)) / (L.v * L.v)); }
double DistanceLineToLine(Line3 L1, Line3 L2){
    if(LineCoplanar(L1, L2)){
        if(sgn(Length(L1.v ^ L2.v)) == 0) return DistancePointToLine(L1.p, L2); // parallel
        return -1; // intersected
    }
    Vector3 v = L1.v ^ L2.v, v0 = L2.p - L1.p;
    return v * v0 / Length(v);
```

```
}
double DistancePointToPlane(Point3 P, Plane3 A){ return fabs((P - A.p) * A.v) / Length(A.v); }
bool PointOnPlane(Point3 P, Plane3 A){ return sgn((P - A.p) * A.v) == 0; }
Point3 PointPlaneProjection(Point3 P, Plane3 A){ A.v = A.v / Length(A.v); return P - A.v * ((P - A.p) * A.v); }
bool LinePlaneParallel(Line3 L, Plane3 A){ return sgn(L.v * A.v) == 0; }
bool LineOnPlane(Line3 L, Plane3 A){ return LinePlaneParallel(L, A) && PointOnPlane(L.p, A); }
Point3 LinePlaneIntersection(Line3 L, Plane3 A){
    double t = ((A.p - L.p) * A.v) / (L.v * A.v);   // if L.v * A.v == 0, then line is parallel to plane or on plane
    return L.p + L.v * t;
}
bool TriangleSegmentIntersection(Point3 A, Point3 B, Point3 C, Point3 P1, Point3 P2, Point3 &P){
    // ABC is the triangle; P1P2 is the segment; intersection is stored in P
    Vector3 n = (B - A) ^ (C - A);
    if(sgn(n * (P2 - P1)) == 0) return false; // parallel or coplanar
    double t = ((A - P1) * n) / ((P2 - P1) * n);
    if(cmp(t, 0) < 0 || cmp(t, 1) > 0)  return false; // intersection is not on segment
    P = P1 + (P2 - P1) * t;
    return PointInTriangle(P, A, B, C);
}

struct Sphere{
    Point3 p;
    double r;
    Sphere() {}
    Sphere(Point3 p, double r=0):p(p), r(r) {}
};
double torad(double deg){ return deg / 180 * PI; }
Point3 ConvertLatitudeLongitude(double r, double lati, double longi){
    lati = torad(lati), longi = torad(longi);
    return Point3(r*cos(lati)*cos(longi), r*cos(lati)*sin(longi), r*sin(lati));
}
double ArcDistancePointToPoint(double r, Point3 A, Point3 B){
    double d = DistancePointToPoint(A, B);
    return 2 * asin(d / 2 / r) * r;
}
```