

反演变换

算法思想

给定反演中心 O 和反演半径 r ，剩余点 A 的反演点 A' 满足 $|OA| \times |OA'| = R^2$

可以发现不过 O 的圆 B 其反演图形也是不过 O 的圆 B'

圆 A 半径为 r_1 其反演图形的半径为 $\frac{1}{2}(\frac{1}{|OA|-r_1} - \frac{1}{|OA|+r_1})R^2$

设 O 点坐标为 (x_0, y_0) 圆的圆心是 A 坐标为 (x_1, y_1) 反演圆的圆心是 B

显然有

$$x_2 = x_0 + \frac{|OB|}{|OA|}(x_1 - x_0)$$

$$y_2 = y_0 + \frac{|OB|}{|OA|}(y_1 - y_0)$$

又因为 $|OB|$ 刚才已经算出来了，所以可以得到反演点坐标

过点 O 的圆 A 其反演图形是不过点 O 的直线。（因为另一个点在无穷远，所以圆无穷大，就变成直线了）然后求出圆心相对要反演的圆的对称点的反演点，然后连接反演点和 O 做个垂线，就是反演的线了。

两个图形相切，他们的反演图形也相切。

算法实现

```
struct Inversion {
    Point o; // 反演中心
    double r; // 反演半径
    Inversion() {}
    Inversion(Point _o, double _r) {
        o = _o;
        r = _r;
    }
    // 点的反演 flag 为 0 获取失败 1 获取成功
    void getPointInv(Point a, Point &aa, int &flag) {
        if(a == o) {
            flag = 0;
            aa = a;
            return;
        }
        Point ptmp = a - o;
        double len = ptmp.len();
        ptmp = ptmp.trunc(r * r / len);
        aa = o + ptmp;
    }
};
```

```
    flag=1;
}
//圆的反演 flag为1变成圆 -1变成直线
void getCircleInv(circle c, Line &l, circle &cc, int &flag) {
    if(c.relation(o)^1) {
        Point p1, p2, pp1, pp2;
        Line lt;
        flag=1;
        if(c.p==o) {
            cc.p=o;
            cc.r=r*r/c.r;
            return;
        }
        lt=Line(c.p, o);
        int ii=c.pointcrossline(lt, p1, p2);
        int f;
        getPointInv(p1, pp1, f);
        getPointInv(p2, pp2, f);
        Point pp=(pp1+pp2)/2;
        cc.p=pp;
        cc.r=pp1.distance(pp2)/2;
        return;
    }
    flag=-1;
    Point ptmp=c.p*2-o, pptmp, p1, p2;
    int f;
    getPointInv(ptmp, pptmp, f);
    p1=o-pptmp;
    p1=p1.rotleft();
    p1=p1+pptmp;
    l=Line(pptmp, p1);
}
//直线的反演 成圆
void getLineInv(Line L, circle &cc, int &flag) {
    if(L.relation(o)==3) {
        flag=0;
        return;
    }
    flag=1;
    Point p=L.lineprog(o), ans;
    int f;
    getPointInv(p, ans, f);
    cc.r=ans.distance(o)/2;
    cc.p=(ans+o)/2;
}
} iv;
```

代码练习

给定两个圆和圆外一点，求过这个点且与这两个圆都外切的圆，输出他们的圆心坐标和半径。

显然这个圆如果被那个点反演是一条线，然后另外两个圆还是圆，所以变成求公切线的问题，最后再反演回去，最后要注意要外切，所以两个圆心要都和给定点在切线的同一侧才可以，判断一下就行了。

```
// 如果 A B 两点在直线同侧 返回 true
bool theSameSideOfLine(Point A, Point B) {
    return sgn((A-s)^(e-s)) * sgn((B-s)^(e-s)) > 0;
}

// a[i] 和 b[i] 分别是第i条切线在圆A和圆B上的切点 f[i]为1内切 为2外切
int getTangents(circle A, Point* a, Point* b, int* f) {
    circle BB>(*this);
    circle B=BB;
    int cnt = 0;
    if (A.r < B.r) {
        swap(A, B);
        swap(a, b);
    }
    double d2 =(A.p.x - B.p.x) * (A.p.x - B.p.x) + (A.p.y - B.p.y) * (A.p.y - B.p.y);
    double rdifff = A.r - B.r;
    double rsum = A.r + B.r;
    if (sgn(d2 - rdifff * rdifff) < 0) return 0; // 内含

    double base = atan2(B.p.y - A.p.y, B.p.x - A.p.x);
    Point pa=Point(A.r,0);
    Point pb=Point(B.r,0);
    Point p0=Point(0,0);
    if (sgn(d2) == 0 && sgn(A.r - B.r) == 0) return -1; // 无限多条切线
    if (sgn(d2 - rdifff * rdifff) == 0) { // 内切，一条切线
        a[cnt] = A.p+pa.rotate(p0,base);
        b[cnt] = B.p+pb.rotate(p0,base);
        f[cnt] = 1;
        ++cnt;
        return 1;
    }
    // 有外公切线
    double ang = acos(rdifff / sqrt(d2));
    a[cnt] = A.p+pa.rotate(p0,base+ang);
    b[cnt] = B.p+pb.rotate(p0,base+ang);
    f[cnt] = 2;
    ++cnt;
    a[cnt] = A.p+pa.rotate(p0,base-ang);
```

```
b[cnt] = B.p+pb.rotate(p0,base-ang);
f[cnt] = 2;
++cnt;
if (sgn(d2 - rsum * rsum) == 0) { // 一条内公切线
    a[cnt] = A.p+pa.rotate(p0,base);
    b[cnt] = B.p+pb.rotate(p0,base+pi);
    f[cnt] = 1;
    ++cnt;
} else if (sgn(d2 - rsum * rsum) > 0) { // 两条内公切线
    double ang = acos(rsum / sqrt(d2));
    a[cnt] = A.p+pa.rotate(p0,base+ang);
    b[cnt] = B.p+pb.rotate(p0,base+pi+ang);
    f[cnt] = 1;
    ++cnt;
    a[cnt] = A.p+pa.rotate(p0,base-ang);
    b[cnt] = B.p+pb.rotate(p0,base+pi-ang);
    f[cnt] = 1;
    ++cnt;
}
return cnt;

} // 两圆公切线 返回切线的条数, -1表示无穷多条切线
```

```
Point LA[1010], LB[1010];
circle ansc[1010];
int fl[1010];
int main() {
    int t;
    cin>>t;
    circle c1,c2;
    circle cc1,cc2;
    Point pt;
    while(t--){
        c1.p.input();
        scanf("%lf",&c1.r);
        c2.p.input();
        scanf("%lf",&c2.r);
        pt.input();
        iv=Inversion(pt,10.0);
        Line lt;
        int f;
        iv.getCircleInv(c1,lt,cc1,f);
        iv.getCircleInv(c2,lt,cc2,f);
        Line l1,l2,l3,l4;
        circle C1,C2,C3,C4;
        int q = cc2.getTangents(cc1, LA, LB, fl), ans = 0;
        for (int i = 0; i < q; ++i) {
            Line lt=Line(LA[i],LB[i]);
```

```
        if (lt.theSameSideOfLine(cc1.p, cc2.p)) {
            if (!lt.theSameSideOfLine(pt, cc1.p)) continue;
            iv.getLineInv(lt, ansc[ans], f);
            ans++;
        }
    }
    printf("%d\n", ans);
    for (int i = 0; i < ans; ++i) {
        printf("%.8f %.8f %.8f\n", ansc[i].p.x, ansc[i].p.y,
ansc[i].r);
    }
}
return 0;
}
```

From:

<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:

https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:%E7%8E%8B%E6%99%BA%E5%BD%AA:%E5%8F%8D%E6%BC%94

Last update: 2021/08/06 18:33