多边形布尔运算平台

原
理
文
档

2014213501 贾晨

2014 年 12 月 18 日

一、 公式及推导

1. 计算两条线段的交点

代码如下所示：

```
/*计算两线段交点,请先判线段是否相交(同时还是要判断是否平行!)*/
CP_Point LineSegInter(CP_Point u1, CP_Point u2, CP_Point v1, CP_Point v2)
{
    CP_Point ret = u1;
    double t = ((u1.m_x - v1.m_x)*(v1.m_y - v2.m_y) - (u1.m_y - v1.m_y)*(v1.m_x - v2.m_x))
        / ((u1.m_x - u2.m_x)*(v1.m_y - v2.m_y) - (u1.m_y - u2.m_y)*(v1.m_x - v2.m_x));
    ret.m_x += (u2.m_x - u1.m_x)*t;
    ret.m_y += (u2.m_y - u1.m_y)*t;
    return ret;
}
```

推导如下：

设线段 L1(p1->p2)和 L2(p3->p4)，其中 p1(x1, y1)为第一条线段的起点，p2(x2, y2)为第一条线段的终点，p3(x3, y3)为第二条线段的起点，p4(x4, y4)为第二段线段的终点。设两条线段的交点为(x0, y0)，则下列方程组必然成立：

1) x0-x1=k1(x2-x1)
2) y0-y1=k1(y2-y1)
3) x0-x3=k2(x4-x3)
4) y0-y3=k2(y4-y3)

其中 k1 和 k2 为任意不为 0 的常数（若为 0，则说明有重合的端点，这种情况在上面已经被排除了）。1)式与 2)式联系，3)式与 4)式联立，消去 k1 和 k2 可得：

1) x0(y2-y1)-x1(y2-y1)=y0(x2-x1)-y1(x2-x1)
2) x0(y4-y3)-x3(y4-y3)=y0(x4-x3)-y3(x4-x3)

将含有未知数 x0 和 y0 的项移到左边，常数项移动到右边，得：

1) (y2-y1)x0+(x1-x2)y0=(y2-y1)x1+(x1-x2)y1
2) (y4-y3)x0+(x3-x4)y0=(y4-y3)x3+(x3-x4)y3

设两个常数项分别为 b1 和 b2：

- b1=(y2-y1)x1+(x1-x2)y1
- b2=(y4-y3)x3+(x3-x4)y3

系数行列式为 D，用 b1 和 b2 替换 x0 的系数所得系数行列式为 D1，替

换 y0 的系数所得系数行列式为 D2，则有：

- |D|=(x2-x1)(y4-y3)-(x4-x3)(y2-y1)
- |D1|=b2(x2-x1)-b1(x4-x3)
- |D2|=b2(y2-y1)-b1(y4-y3)

由此，可求得交点坐标为：x0=|D1|/|D|, y0=|D2|/|D|。

二、 结论及证明

以下结论中用到的函数有：

```
/*计算交叉乘积(P1-P0)x(P2-P0)*/
double xmult(CP_Point p1, CP_Point p2, CP_Point p0){ ... }
```

1. 判断点是否在线段（包括端点）上

   代码如下所示：

```
/*判点是否在线段上,包括端点*/
int dot_online_in(CP_Point p, CP_Point l1, CP_Point l2)
{
    return zero(xmult(p, l1, l2))
        && (l1.m_x - p.m_x)*(l2.m_x - p.m_x)<eps
        && (l1.m_y - p.m_y)*(l2.m_y - p.m_y)<eps;
}
```

   证明如下：

   当点 p 在线段(l1,l2)上时，向量 l2->p 和向量 l1->p 在同一条直线上，即两向量的叉乘结果为 0。同时，需要满足点 p 在线段内部，即要满足 p 点的 x 坐标和 y 坐标都要在 l1 和 l2 之间。由此可得代码中的判决式。

2. 判断两点是否在线段同侧

   代码如下所示：

```
/*判两点在线段同侧,点在线段上返回0*/
int same_side(CP_Point p1, CP_Point p2, CP_Point l1, CP_Point l2)
{
    return xmult(l1, p1, l2)*xmult(l1, p2, l2)>eps;
}
```

   证明如下：

   若点 p1 和 p2 在线段(l1,l2)的同一侧，那么向量 l2->l1 叉乘 p1->l2 的结果和向量 l2->l1 叉乘 p2->l2 的结果的符号应该相同。由此可得代码中的判决式。

3. 判断两直线是否平行

代码如下所示：

```
/*判两直线平行*/
int parallel(CP_Point u1, CP_Point u2, CP_Point v1, CP_Point v2)
{
    double temp = (u1.m_x - u2.m_x)*(v1.m_y - v2.m_y) - (v1.m_x - v2.m_x)*(u1.m_y - u2.m_y);
    return zero(temp);
}
```

证明如下：

若两直线平行，则将它们的方向向量的起点移至原点后，两个向量应该在同一条直线上，即两个向量叉乘结果为 0。由此可得代码中的判决式。

4. 判断三点是否共线

代码如下所示：

```
/*判三点共线*/
int dots_inline(CP_Point p1, CP_Point p2, CP_Point p3)
{
    return zero(xmult(p1, p2, p3));
}
```

证明如下：

若三点（p1，p2，p3）共线，那么向量 p3->p1 与向量 p3->p2 的叉乘结果应该为 0。由此可得代码中的判决式。

5. 判断点是否在多边形内部

设点为 p，以点 p 向左引一条水平射线 L，从射线 L 左端的无穷远处开始一直到点 P 的过程中，当遇到多边形的第一个交点时 L 进入了多边形，当遇到第二个交点时，L 穿出了多边形…可知，规律如下：当在遇到 P 点之前 L 与多边形的交点为偶数个时，说明 p 点不在多边形内，当在遇到 p 点之前 L 与多边形得交点为奇数个时，说明 P 点在多边形内。

但是，这个规律并不具有普遍性，还有几种特殊情况需要额外考虑：

(1)当点 P 在多边形的某条边上时，可以直接判断其在多边形中。

(2)对于多边形的水平边不作考虑。

(3)对于多边形的顶点与 L 相交，则需要判断该顶点是否为顶点所在的边的那个纵坐标较大的顶点，如果是较大的那个顶点与 L 相交则计数，否则忽略。

6. 判断线段是否在多边形内部

用线段间交点将多边形的边分割，然后再判断线段是否在多边形内部就

变得相当简单，因为不会再存在线段横跨多边形内外的情况，所以直接判断线段中点在不在多边形内即可。若中点在多边形内部，则线段也在多边形内；若中点不在多边形内，线段也不在多边形内。

7. 多边形合法性检查

需要做的合法性检查有如下几种：

1) 每个环中应该有多于 2 个顶点，且外环应为逆时针，内环应为顺时针（手动添加内外环时可能会出错）；
2) 环 必须是简单多边形，即环中任意两条边不能有除端点外的其它顶点；
3) 多边形的一个区域中，内环必须在该区域的外环的内部；
4) 多边形的一个区域中，内环之间不能相互包含；
5) 多边形的一个区域中，内环间交点只能是端点，不能是边或其它；
6) 同一个多边形的不同区域不能相交，若相交，交点只能是端点，不能是边或其它。（注意当多边形只有两个外环，且这两个外环相互嵌套的情况！）

对于上述检查，构想了如下检查流程及算法：

1) 遍历多边形中所有环，判断环的顶点个数是否大于 2。接下来，判断是否符合顺/逆时针序，方法为：找到环中的一个凸顶点，此顶点联结的两条边的方向即为此环的方向；（第 1 种合法性）
2) 将一个多边形中所有的边抽取出来，单独放到边结构体中，检查它们两两之间有没有除端点外的其它交点，或者重合边；（第 2、5 种合法性）
3) 判断多边形同一区域中的内环是否在该区域的外环中（因为 2 中已经检查了边相交的情况，所以只可能出现内环完全在外环外的不合法情况，不可能出现一半在外环内，一半在外环外的不合法情况），可以将该区域的外环作为一个多边形，判断该区域所有内环的所有的边的中点是否在这个多边形内来判断这个内环是不是在外环内；（第 3 种合法性）
4) 判断多边形同一个区域中的内环是否存在包含关系（因为 2 中已经检查了边相交的情况，所以只可能出现完全被包含的不合法情况，不可能出现包含一部分的不合法情况），可以将内环作为一个单独的多边形（点顺序应该不用考虑，因为只需要判断交点个数），判断其它内环中所有的边的中点是否在这个多边形内来判断他们是否存在包含关系；（第 4 种合法性）

5) 判断多边形中一个区域被另一个区域完全包含的情况（因为 2 中已经检查了边相交的情况，所以只可能出现完全被包含的不合法情况），可以将一个区域作为一个单独的多边形，判断其它区域外环的所有的边的中点是否在这个多边形内来判断他们是否存在包含关系。（第 6 种合法性）

三、 算法复杂性分析

设通过多边形 A 和 B 的交点将多边形切割后，多边形 A 和 B 共有 V 个顶点，E=V-1 条边。由此分析算法复杂性如下：

1. 合法性检查

1) 函数 LoopDirection()

此函数需要遍历多边形 A 和 B 的所有环来判断它们的方向是否正确，对于每个环，需要遍历它的所有的顶点来寻找最右侧的凸顶点，通过判断凸顶点相连的两条线段的方向来判断环的方向。

所以此函数的时间复杂度为 O(V)。

2) 函数 EdgeIntersection()

此函数需要判断一个多边形中的所有的边两两之间是否有除端点外的其它交点，需要两重循环遍历所有的边对，故此函数的时间复杂度为 $O(E^2)=O(V^2)$。

3) 函数 InOutLoop()

此函数用来检查多边形同一区域中的内环是否在该区域的外环中，方法为：将该区域的外环作为一个多边形，判断该区域所有内环中所有的边的中点是否在这个多边形内来判断这个内环是不是在外环内。

此函数时间复杂度的上界为 $O(VE) =O(V^2)$。

4) 函数 InInLoop()

此函数用来检查多边形同一个区域中的内环是否存在包含关系，方法为：将内环作为一个单独的多边形（点顺序应该不用考虑，因为只需要判断交点个数），判断其它内环中所有的边的中点是否在这个多边形内来判断他们是否存在包含关系。

此函数时间复杂度的上界为 $O(V^2)$。

5) 函数 InInRegion()

此函数用来检查同一个多边形中一个区域被另一个区域完全包含的情况，方法为：将一个区域作为一个单独的多边形，判断其它区域的外环中所有的边的中点是否在这个多边形内来判断他们是否存在包含关系。

此函数时间复杂度的上界为 $O(V^2)$。

综上，合法性检查的时间复杂度为 $O(V^2)$。

2. 布尔运算

由于四类布尔运算（A∪B、A∩B、A-B、B-A）的原理及过程类似，故在此将它们统一按照布尔运算的时间复杂度来进行分析。步骤如下：

1) 求 A 和 B 线段的交点，并将其添加到 A 和 B 中。由于要遍历所有的边对，所以时间复杂度为 $O(E^2)=O(V^2)$；
2) 遍历 A/B 中所有边，判断它们和多边形 B/A 的关系，从而决定是否将它们添加到布尔运算结果边集合中。其中判断它们和 B/A 的关系需要遍历 B/A 中的所有边。所以时间复杂度为 $O(E^2)=O(V^2)$；
3) 将边集合按照起点的 x-y 坐标排序，时间复杂度为 $O(ElgE)=O(VlgV)$；
4) 将边集合中所有的边组合到多边形结构中，每次需要遍历边集合中余下的所有边，所以总的操作次数为$(E-1)+(E-2)+\ldots+1=O(E^2)=O(V^2)$。

综上所述，布尔运算总体时间复杂度为 $O(V^2)$。