Computer Science and Application 计算机科学与应用, 2022, 12(5), 1246-1269
Published Online May 2022 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2022.125126

Hans汉斯

基于射线法的全场景平面直边无孔多边形父子关系判断算法

王振耀

特变电工沈阳变压器集团有限公司，辽宁 沈阳

收稿日期：2022年4月13日；录用日期：2022年5月10日；发布日期：2022年5月18日

摘　要

提出了一种实现平面直边无孔多边形父子关系判断的算法。该算法既能满足无孔矩形间的父子关系判断，又能够满足无孔异形多边形的父子关系判断。本算法应用于“上海西门子医疗器械有限公司MES系统”的工厂建模地图绘制中，表明我们无需依赖第三方组件，即可满足各种平面直边无孔多边形间父子关系的判断。

关键词

多边形冲突检测，多边形父子关系判断，算法

A Ray-Based Algorithm for Determining the Parent-Child Relationship of Planar Straight-Edge Non-Porous Polygons in the Whole Scene

Zhenyao Wang

TBEA Shenyang Transformer Group Co., Ltd., Shenyang Liaoning

Received: Apr. 13th, 2022; accepted: May 10th, 2022; published: May 18th, 2022

Abstract

An algorithm is proposed to realize the parent-child relationship judgment of planar straight-edge nonporous polygons. The algorithm is able to satisfy both parent-child relationship determination

文章引用: 王振耀. 基于射线法的全场景平面直边无孔多边形父子关系判断算法[J]. 计算机科学与应用, 2022, 12(5): 1246-1269. DOI: 10.12677/csa.2022.125126

between non-porous rectangles and non-porous heterogeneous polygons. The algorithm is applied to the factory modeling map of "MES system of Shanghai Siemens Medical Devices Co., Ltd.", which shows that we can satisfy the determination of parent-child relationship among various planar straight-sided aperture-free polygons without relying on third-party components.

Keywords

Polygon Conflict Detection, Polygon Parent-Child Relationship Determination, Algorithms

1. 引言

平面直边无孔多边形父子关系判断是空间拓扑关系的一项重要的内容，有着诸多重要的应用，技术和方法比较成熟有效。但目前国内基本都依赖于国外的各种第三方组件，对于其原理的理解尚为黑盒状态。即使有部分相关文献的发表，其最终的实现效果尚无法覆盖全部场景。

目前，工程上应用最多的还是射线法，但其应用场景也仅局限在了简单多边形间的父子关系判断上。

本文研究的内容将在简单多边形间的父子关系判断之上，进一步探讨复杂的凹凸多边形间的父子关系判断的算法，最终得到的算法将同样适用于简单多边形间的父子关系判断。

2. 平面直边无孔多边形父子关系判断的常用方法及其存在的主要问题

从技术上讲，这样的问题很早就得到了解决，而且在很多文献中都对此有所介绍，但从目前公开的文献中看，绝大多数仅是从实现上做了大概的介绍。比如文献[1]，其所提供的算法，仅能应用于特定场景，无法满足全场景的判断。文献[2]，其所提供的算法过于复杂，计算量太大，且仅针对简单多边形，覆盖场景不全。而文献[3]和文献[4]虽然提供的算法简单，但同样也只能针对简单多边形间的父子关系做判断，针对复杂的凹凸多边形就无法适用了。例如，使用文献[1]、文献[2]、文献[3]、文献[4]的算法针对图二中的所有关系进行判断时，总是有一些判断结果与事实不服。

本文介绍的基于射线法的平面无孔直边多边形父子关系判断算法。它是为满足“上海西门子医疗器械有限公司 MES 系统”的工厂建模地图绘制功能而设计的。

全场景大致分为三类，一类是普通平面直边无孔多边形间的父子关系判断，一类是普通平面直边无孔多边形于异形平面直边无孔多边形间的父子关系判断，一类是异形平面直边无孔多边形间的父子关系判断。

本文的算法很好的完成了全场景的判断，本文采用纯数学的方法实现，该算法自动化程度高、运行高效、场景覆盖面广，具有很强的使用价值。

3. 基于射线法的平面直边无孔多边形父子关系判断算法

3.1. 基本目的

二维直边多边形的父子关系冲突判断，通常的做法是，判断子多边形是否有点在父多边形外。但这种方法覆盖的场景并不全面，因为存在满足以上条件但仍然冲突的场景。如下图 1 所示：

Figure 1. Relationship Judgment
图 1. 关系判断

除此之外整个平面直边无孔多边形父子关系判断还有其它场景，如下图 2 所示：

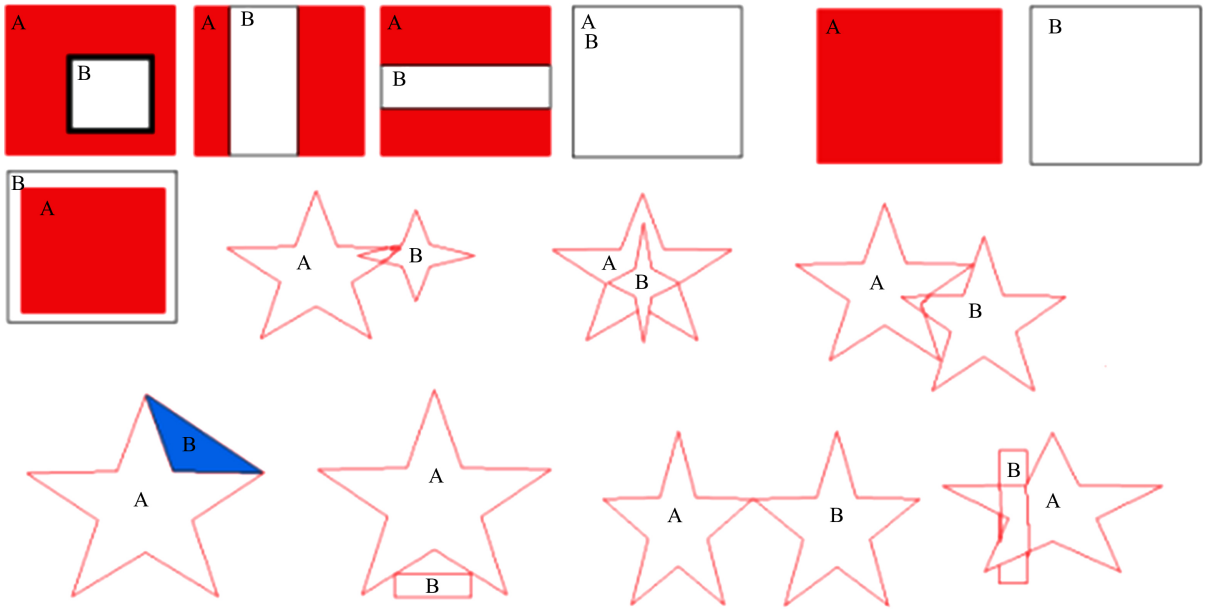

Figure 2. Polygonal parent-child relationship determination scenario
图 2. 多边形父子关系判断场景

本算法的目的是满足图 1 及图 2 中的所有场景中的平面直边无孔多边形间的父子关系判断。

3.2. 基本算法

该算法需要两个要素：组成多边形的顶点的集合，组成多边形的直线段的集合。只有组成多边形的线段的端点与组成多边形的定点重合且组成多边形的线段是直的，才是符合要求的。

从以上图 1 和图 2 可知，平面直边无孔多边形的父子关系判断的三个关键点：

1) 子多边形的点全部在父多边形内。
2) 父多边形的点全部在子多边形外。
3) 父多边形的线段与子多边形的线段不交叉。

3.3. 算法的详细说明及实现

3.3.1. 创建多边形

定义一个代表多边形类，此类包含两个属性。

属性：

1) Points：当前多边形的所有端点的集合。

2) LineSagements：基于 Points 组成当前多边形的线段的集合。

代码见附录 1

3.3.2. 创建点

定义一个代表点的类，此类仅包含两个 double 类型的属性 X、Y。

其中 X 为点的 X 坐标，Y 为点的 Y 坐标。

代码见附录 2

3.3.3. 创建线段

定义一个代表线段的类型，此类包含 5 个属性、1 个方法。

属性：

1) Start：线段的起始点。

2) End：线段的终止点。

3) FunType：线段的类型。

4) Params：double 类型的列表，线段的斜率及线段与 Y 轴的交点。当列表中的第一个对象代表当前线段的斜率，第二个对象代表当前线段与 Y 轴的交点的 Y 坐标；其中当列表中仅存在一个对象时，表示当前线段与 Y 轴不相交。

5) Intersection：与其它线段的交点列表。

方法：

6) GetFuncParam：求当前线段的斜率，并设置当前线段的类型，当当前线段为斜线时，同时求出当前线段与 Y 轴的交点的 Y 坐标。

注意：须在构造线段对象的同时，调用 GetFuncParam 方法。

代码见附录 3

3.3.4. 创建线段类型的枚举

枚举类型如下：

1) XX：竖线

2) YY：横线

3) XnXYnY：斜线

代码见附录 4

3.3.5. 创建点与线的关系的枚举

枚举类型如下：

1) VertexOverlap：顶点重合

2) Cross：相交

3) UnCross：不相交

4) OnLineSagement：点在线上

代码见附录 5

3.3.6. 在多边形类中增加冲突检测方法 CheckIfInArea

该方法的入参为父多边形对象，返回值为 true 或者 false，其中 true 代表这两个多边形的父子关系成立，false 则代表这两个多边形的父子关系不成立。

方法体如下：

1) 若父多边形的 LineSagements 为空，则当前方法返回 true。

2) 循环当前多边形的 Points 中的对象，使用循环出的对象，调用其自身的 CheckIfInArea 方法，将父多边形传入这个方法，并对调用的返回结果取反，若取反后的结果为 true，则当前方法返回 false。

3) 循环父多边形的 Points 中的对象，使用循环出的对象，调用其自身的 CheckIfInArea 方法，将当前多边形传入这个方法，若返回的结果为 true，则当前方法返回 false。

4) 调用当前多边形对象的 WhetherPolygonIntersection 方法，并将父多边形传入这个方法，若返回结果为 false，则当前方法返回 true；若返回结果为 false，则判断当前多边形的线段是否有在父多边形外的部分。

判断方法如下：

1) 循环取当前多边形的 LineSagements 中的对象，赋值给临时变量 edg。

2) 判断 edg 的交点列表中是否存在对象。若存在对象则执行下一步，否则进行下一次循环。

3) 判断 edg 的类型，若为竖线则进行第 4 步，若为横线则进行第 5 步，若为斜线则进行第 6 步。

4) 将 edg 的交点列表按照每个交点的 Y 坐标进行升序，然后赋值给临时变量 jiaodianList；循环取 jiaodainList 中的元素，赋值给临时变量 start，取下一个元素赋值给临时变量 end，创建一个新的点对象，赋值给临时变量 z，其中 z 的 X 坐标为 start 的 X 坐标，z 的 Y 坐标为 start 的 Y 坐标加上 end 的 Y 坐标减去 start 的 Y 坐标的差的 1/2；

$$z.Y = start.Y + \left(\left(end.Y - start.Y \right) / 2 \right)$$

使用 z 调用其自身的 CheckIfInArea 方法，将父多边形传入该方法，然后判断返回结果，若返回结果为 true 则跳过本次循环，直接进行下一次，若返回结果为 false，则当前方法返回 false。

5) 将 edg 的交点列表按照每个交点的 X 坐标进行升序，然后赋值给临时变量 jiaodianList；循环取 jiaodainList 中的元素，赋值给临时变量 start，取下一个元素赋值给临时变量 end，创建一个新的点对象，赋值给临时变量 z，其中 z 的 X 坐标为 start 的 X 坐标加上 end 的 X 坐标减去 start 的 X 坐标的差的 1/2；

$$z.X = start.X + \left(\left(end.X - start.X \right) / 2 \right)$$

z 的 Y 坐标为 start 的 Y 坐标；

使用 z 调用其自身的 CheckIfInArea 方法，将父多边形传入该方法，然后判断返回结果，若返回结果为 true 则跳过本次循环，直接进行下一次，若返回结果为 false，则当前方法返回 false。

6) 判断 edg 的起始点的 Y 坐标是否小于等于 edg 的终止点的 Y 坐标，若是则进行第 7 步，若不是则进行第 8 步。

7) 将 edg 的交点列表先按照每个交点的 X 坐标进行升序，然后按照每个交点的 Y 坐标升序，然后赋值给临时变量 jiaodianList；循环取 jiaodainList 中的元素，赋值给临时变量 start，取下一个元素赋值给临时变量 end，创建一个新的点对象，赋值给临时变量 z，其中 z 的 X 坐标为 start 的 X 坐标加上 end 的 X 坐标减去 start 的 X 坐标的差的 1/2 (公式 2)；

$$z.Y = start.Y + \left(\left(end.Y - start.Y \right) / 2 \right)$$

使用 z 调用其自身的 CheckIfInArea 方法，将父多边形传入该方法，然后判断返回结果，若返回结果为 true 则跳过本次循环，直接进行下一次，若返回结果为 false，则当前方法返回 false。

8) 将 edg 的交点列表先按照每个交点的 X 坐标进行升序，然后按照每个交点的 Y 坐标降序，然后赋值给临时变量 jiaodianList；循环取 jiaodainList 中的元素，赋值给临时变量 start，取下一个元素赋值给临时变量 end，创建一个新的点对象，赋值给临时变量 z，其中 z 的 X 坐标为 start 的 X 坐标加上 end 的 X 坐标减去 start 的 X 坐标的差的 1/2 (公式 2)；

$$z.Y = start.Y + \left(\left(end.Y - start.Y \right) / 2 \right)$$

使用 z 调用其自身的 CheckIfInArea 方法，将父多边形传入该方法，然后判断返回结果，若返回结果为 true 则跳过本次循环，直接进行下一次，若返回结果为 false，则当前方法返回 false。

代码见附录 6

3.3.7. 在点的类中增加点与线段关系的判断方法 CheckPointInLineSagement

该方法的入参为需要与之判断的线段，返回值为当前点与传入线段的关系。主要功能是判断此点向右引出的射线是否穿过传入的线段。

穿过的定义如下：

1) 此点在传入线段的顶点上。

2) 此点在传入线段的线上。

3) 此点不符合以上两种情况，且此点向右引出的射线与传入的线段有交点。

方法主体如下：

1) 判断此点是否在传入线段的顶点上，若是，则返回关系 VertexOverlap (顶点重合)。否则进行下一步

2) 判断传入线段的类型。

若为竖线：

判断此点的 X 坐标是否等于传入线段的起始点的 X 坐标；且此点的 Y 坐标是否小于等于传入线段的起始点的 Y 坐标且此点的 Y 坐标大于等于传入线段的终止点的 Y 坐标，或者此点的 Y 坐标大于等于传入线段的起始点的 Y 坐标且此点的 Y 坐标小于等于传入线段的终止点的 Y 坐标。若是，则此方法返回关系 OnLineSagement (点在线上)。

若为横线：

判断此点的 y 坐标是否等于传入线段的起始点的 y 坐标；且此点的 x 坐标是否大于等于传入线段的起始点的 x 坐标且此点的 X 坐标小于等于传入线段的终止点的 X 坐标，或者此点的 X 坐标小于等于传入线段的起始点的 X 坐标且此点的 X 坐标大于等于传入线段的终止点的 X 坐标。若是，则此方法返回关系 OnLineSagement (点在线上)。

若为斜线：

则进行第 3 步

3) 判断传入线段的起始点的 Y 坐标是否小于此点的 Y 坐标且传入线段的终止点的 y 坐标大于等于此点的 Y 坐标；或者，传入线段的起始点的 Y 坐标大于等于此点的 y 坐标且传入线段的终止点的 Y 坐标小于此点的 y 坐标。若不是则进行第 4 步。否则需要先求出传入线段与射线 Y 坐标相同的点的 X 坐标，然后将这个 X 坐标与当前点的 x 坐标进行比较，以判断当前点与传入线段的关系。

若交点的 X 坐标等于当前点的 x 坐标，则返回关系 OnLineSagement (点在线上)，否则判断交点的 x 坐标是否大于当前点的 x 坐标，若是则返回关系 Cross (相交)，否则进行第 4 步。

交点的 X 坐标为：

$$rs.X+(t.Y-rs.Y)*(rz.X-rs.X)/(rz.Y-rs.Y)$$

t.Y：当前点的 Y 坐标

rs.X：传入线段的起始点的 X 坐标

rs.Y：传入线段的起始点的 Y 坐标

rz.X：传入线段的终止点的 X 坐标

rz.Y：传入线段的终止点的 Y 坐标

4) 返回关系 UnCross (不相交)。

代码见附录 7

3.3.8. 在点的类中增加子多边形的点是否在父多边形内的判断方法 CheckIfInArea

该方法的入参为父多边形，返回值为 true 或者 false，其中 true 代表当前点在传入的多边形内(包含当前点与传入多边形的顶点重合的情况) flse 代表当前点在传入的多边形外。

方法主体如下：

1) 定义临时变量射线穿过传入多边形边界的次数 cnt，并赋值为 0。

2) 循环取传入多边形的 LineSagements 中的元素，判断其类型，若为 Cross 类型则将 cnt 的值+1。若为 OnLineSagement 类型，则返回 true。若为 VertexOverlap 类型则返回 true。若为 UnCross 类型则进行下一次循环。

3) 判断 cnt 是否为奇数，若为奇数则返回 true。否则返回 false。

代码见附录 8

3.3.9. 在点的类中增加父多边形的点是否在子多边形内的判断方法 CheckIfInChildArea

该方法的入参为子多边形，返回值为 true 或者 false，其中 true 代表当前点在传入的多边形内(包含当前点与传入多边形的顶点重合的情况)，flse 代表当前点在传入的多边形外。

方法主体如下：

1) 定义临时变量射线穿过传入多边形边界的次数 cnt，并赋值为 0。

2) 循环取传入多边形的 LineSagements 中的元素，判断其类型，若为 Cross 类型则将 cnt 的值+1。若为 OnLineSagement 类型，则返回 false。若为 VertexOverlap 类型则返回 false。若为 UnCross 类型则进行下一次循环。

3) 判断 cnt 是否为奇数，若为奇数则返回 true。否则返回 false。

代码见附录 9

3.3.10. 在多边形的类中增加多边形自身的线段是否与另一多边形的所有线段相交的方法 WhetherPolygonIntersection

该方法的入参为多边形，返回值为 true 或者 false，其中 true 代表当前多边形的线段与传入多边形中的线段有相交的情况，false 代表当前多边形的线段与传入多边形中的线段没有相交的情况。

方法主体如下：

1) 循环取当前多边形的线段，赋值给临时变量 edg。

2) 将 edg 的起始点赋值给临时变量 a。

3) 将 edg 的终止点赋值给临时变量 b。

4) 循环取传入多边形的线段，赋值给临时变量 fatherEdge。

5) 将 fatherEdge 的起始点赋值给临时变量 c。

6) 将 fatherEdge 的终止点赋值给临时变量 d。

7) 定义临时变量 denominator。

8) 求 denominator 的值。

$$\text{denominator} = (b.Y - a.Y)*(d.X - c.X) - (a.X - b.X)*(c.Y - d.Y)$$

9) 判断 denominator 是否等于 0，若不等于 0 则进行第 32 步，否则进行下一步。

10) 判断 edg 的类型，若为竖线，则进行第 11 步，若为横线，则进行第 18 步，若为斜线，则进行第 25 步。

11) 判断 edg 的起始点的 X 坐标是否等于 fatherEdge 的起始点的 X 坐标，若不是则跳出当前循环，进行下一次循环。若是则进行下一步。

12) 判断 b 的 Y 坐标是否大于 c 的 Y 坐标或者 a 的 Y 坐标是否小于 d 的 Y 坐标。若是则跳出当前循环，进行下一次循环。

13) 判断 a 的 Y 坐标是否等于 c 的 Y 坐标并且 b 的 Y 坐标是否等于 d 的 Y 坐标。若是则跳出当前循环，进行下一次循环。

14) 判断 a 的 Y 坐标是否大于 c 的 Y 坐标并且 b 的 Y 坐标小于等于 c 的 Y 坐标并且 b 的 Y 坐标大于等于 d 的 Y 坐标。若是则将 c 加入到 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

15) 判断 a 的坐标是否小于等于 c 的 Y 坐标并且 a 的 Y 坐标大于等于 d 的 Y 坐标并且 b 的 Y 坐标小于 d 的 Y 坐标。若是则将 d 加入到 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

16) 判断 c 的 Y 坐标是否大于等于 a 的 Y 坐标且 d 的 Y 坐标小于等于 b 的 Y 坐标。若是则跳出当前循环，进行下一次循环。

17) 判断 a 的 Y 坐标是否大于等于 c 的 Y 坐标并且 b 的 Y 坐标小于等于 d 的 Y 坐标。若是则将 c 和 d 加入 edge 的 Intersection 中。然后跳出当前循环，进行下一次循环。

18) 判断 edg 的起始点的 Y 坐标是否等于 fatherEdge 的起始点的 Y 坐标。若不是则跳出当前循环，进行下一次循环。否则进行下一步。

19) 判断 b 的 X 坐标是否小于 c 的 X 坐标或者 a 的 X 坐标大于 d 的 X 坐标。若是则跳出当前循环，进行下一次循环。

20) 判断 a 的 X 坐标是否等于 c 的 X 坐标且 b 的 X 坐标等于 d 的 X 坐标。若是则跳出当前循环，进行下一次循环。

21) 判断 a 的 X 坐标是否小于 c 的 X 坐标并且 b 的 X 坐标大于等于 c 的 X 坐标并且 b 的 X 坐标小于等于 d 的 X 坐标。若是则将 c 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

22) 判断 b 的 X 坐标是否大于 d 的 X 坐标并且 a 的 X 坐标大于等于 c 的 X 坐标并且 a 的 X 坐标小于等于 d 的 X 坐标。若是则将 d 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

23) 判断 c 的 X 坐标是否小于等于 a 的 X 坐标并且 d 的 X 坐标大于等于 b 的 X 坐标。若是则跳出当前循环，进行下一次循环。

24) 判断 a 的 X 坐标是否小于等于 c 的 X 坐标并且 b 的 X 坐标大于等于 d 的 X 坐标。若是则将 c 和 d 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

25) 判断 edg 的斜率是否等于 fatherEdge 的斜率并且 edg 与 Y 轴交点的 Y 坐标等于 fatherEdge 与 Y

轴交点的 Y 坐标。若不是则跳出当前循环，进行下一次循环。否则进行下一步。

26) 判断 a 的 X 坐标是否小于 c 的 X 坐标并且 b 的 X 坐标小于 c 的 X 坐标或者判断 a 的 X 坐标是否大于 d 的 X 坐标并且 b 的 X 坐标大于 d 的 X 坐标。若是则跳出当前循环，进行下一次循环。

27) 判断 a 的 X 坐标是等于 c 的 X 坐标并且 a 的 Y 坐标等于 c 的 Y 坐标并且 b 的 X 坐标等于 d 的 X 坐标并且 b 的 Y 坐标等于 d 的 Y 坐标。若是则跳出当前循环，进行下一次循环。

28) 判断 a 的 X 坐标是否小于 c 的 X 坐标并且 b 的 X 坐标大于等于 c 的 X 坐标并且 b 的 X 坐标小于等于 d 的 X 坐标。若是则将 c 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

29) 判断 a 的 X 坐标是否大于等于 c 的 X 坐标并且 a 的 X 坐标小于等于 d 的 X 坐标并且 b 的 X 坐标大于 d 的 X 坐标。若是则将 d 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

30) 判断 a 的 X 坐标是否大于等于 c 的 X 坐标并且 a 的 X 坐标小于等于 d 的 X 坐标并且 b 的 X 坐标大于等于 c 的 X 坐标并且 b 的 X 坐标小于等于 d 的 X 坐标。若是则跳出当前循环，进行下一次循环。

31) 判断 a 的 X 坐标是否小于等于 c 的 X 坐标并且 b 的 X 坐标大于等于 d 的 X 坐标。若是则将 c 和 d 加入 edg 的 Intersection 中，然后跳出当前循环，进行下一次循环。

32) 定义线段所在直线的交点坐标 x

$$x = \frac{((b.X - a.X)*(d.X - c.X)*(c.Y - a.Y) + (b.Y - a.Y)*(d.X - c.X)*a.X - (d.Y - c.Y)*(b.X - a.X)*c.X)}{denominator}$$

33) 定义线段所在直线的交点坐标 y

$$y = \frac{-((b.Y - a.Y)*(d.Y - c.Y)*(c.X - a.X) + (b.X - a.X)*(d.Y - c.Y)*a.Y - (d.X - c.X)*(b.Y - a.Y)*c.Y)}{denominator}$$

34) 判断以下两个公式的结果是否都小于等于 0，若是则将线段所在直线的交点加入 edg 的 Intersection 中，若不是则跳出当前循环，进行下一次循环。

35) 循环结束后，判断当前多边形的 LineSagements 中是否包含 Intersection 中包含元素的元素，若是则返回 true，否则返回 false。

代码见附录 10

4. 结论

本文提出的算法，只需要按顺序提供需要进行判断的两个平面直边无孔多边形的顶点的坐标，即可通过本文的算法判断出他们之间的父子关系是否成立。本文给出的算法相较文献[1]、文献[2]、文献[3]、文献[4]，其处理流程更加清晰，覆盖场景更加全面，编码简单，易于实现，无编程语言限制，代码复用率高，且只要是平面直边无孔多边形，不论是简单多边形还是凹、凸多边形，均可正确判断出它们之间的父子关系是否成立。

该算法用于“上海西门子医疗器械有限公司 MES 系统”工厂建模地图绘制中，满足了区域地图规划，验证了该算法的可行性。

致　谢

致谢本文的算法得到了在上海西门子医疗器械有限公司任职的陈勋齐的大力支持与论证，在此表示衷心的感谢！

参考文献

[1] 王家耀. 空间信息系统原理[M]. 北京: 科学出版社, 2001.

[2] 傅清祥，王晓东．求解简单多边形间包含关系的扫描线算法[J]．计算机辅助设计于图形学学报，1997(2): 157-163.

[3] 刘晓婧，赵俊三．判定点于多边形及简单多边形之间的空间关系[J]．科技情报开发与经济，2008，18(28): 223-224.

[4] 薛胜，潘懋，王勇．多边形叠置分析算法研究[J]．计算机工程与应用. 2003(2): 57-60.

附录

附录 1

```
publicclass Area
{
public Area(List<Point> points)
{
       Points = points;
if(this.Points !=null&&this.Points.Any())
{
          LineSagements =new List<LineSagement>();
for(int i =0; i < points.Count -1; i++)
{
              LineSagements.Add(new LineSagement(points[i], points[i +1]));
}
          LineSagements.Add(new LineSagement(points.First(),
points.Last()));
}
}

public List<Point> Points {get;set;}
public List<LineSagement> LineSagements {get;set;}
}
```

附录 2

```
publicclass Point
{
public Point(double x,double y)
{
this.X = x;
this.Y = y;
}
publicdouble X {get;privateset;}
publicdouble Y {get;privateset;}
}
```

附录 3

```
publicclass LineSagement
{
public LineSagement(Point start, Point end)
{
```

```
this.Start = start;
this.End = end;
       GetFuncParam(this);
}
public Point Start {get;privateset;}
public Point End {get;privateset;}
public LineType_Enum FunType {get;set;}
public List<double> Params {get;set;}=new List<double>();
/// <summary>
/// 交点列表
/// </summary>
public List<Point> Intersection {get;set;}=new List<Point>();
publicvoid GetFuncParam(LineSagement lineSegment)
{
double x1 =this.Start.X;
double y1 =this.Start.Y;
double x2 =this.End.X;
double y2 =this.End.Y;

if(x1 == x2)
{
//type=2
this.FunType = LineType_Enum.XX;
this.Params.Add(x1);
}
elseif(y1 == y2)
{
//type=1
this.FunType = LineType_Enum.YY;
this.Params.Add(y1);
}
else
{
//type=3
this.FunType = LineType_Enum.XnXYnY;
double a =(y2 - y1)/(x2 - x1);//斜率
double b = y1 -(x1 *((y2 - y1)/(x2 - x1)));//与Y轴的交点
this.Params.Add(a);
this.Params.Add(b);
}
```

```
}
}
```

附录 4

```
publicenum LineType_Enum
{
/// <summary>
/// 竖线
/// </summary>
    XX,
/// <summary>
/// 横线
/// </summary>
    YY,
/// <summary>
/// 斜线
/// </summary>
    XnXYnY
}
```

附录 5

```
publicenum PointWithLineSagementState_Enum
{
/// <summary>
/// 顶点重合
/// </summary>
    VertexOverlap,
/// <summary>
/// 相交
/// </summary>
    Cross,
/// <summary>
/// 不相交
/// </summary>
    UnCross,
/// <summary>
/// 在线段上
/// </summary>
    OnLineSagement
}
```

附录 6

```
publicbool CheckIfInArea(Area area)
{
if(area.LineSagements ==null)
{
returntrue;
}
//若子点有在父外的则结束
foreach(Point point inthis.Points)
{
if(!point.CheckIfInArea(area))
returnfalse;
}
//若父点有在子内的则结束
foreach(Point point in area.Points)
{
if(point.CheckIfInChildArea(this))
returnfalse;
}
//所有子线段与父线段没有相交则不冲突
if(WhetherPolygonIntersection(area))
{
foreach(LineSagement edg inthis.LineSagements)
{
if(edg.Intersection.Any())
{
if(edg.FunType == LineType_Enum.XX)
{
                List<Point> jiaodainList = edg.Intersection.OrderBy(m =>
m.Y).ToList();
for(int i =0; i < jiaodainList.Count -1; i++)
{
                    Point start = jiaodainList[i];
                    Point end = jiaodainList[i +1];
                    Point z =new Point(start.X, start.Y +((end.Y -
start.Y)/2));
if(z.CheckIfInArea(area))
{
continue;
```

```
}
else
{
returnfalse;
}
}
}
elseif(edg.FunType == LineType_Enum.YY)
{
                List<Point> jiaodainList = edg.Intersection.OrderBy(m =>
m.X).ToList();
for(int i =0; i < jiaodainList.Count -1; i++)
{
                    Point start = jiaodainList[i];
                    Point end = jiaodainList[i +1];
                    Point z =new Point(start.X +((end.X - start.X)/2),
start.Y);
if(z.CheckIfInArea(area))
{
continue;
}
else
{
returnfalse;
}
}
}
elseif(edg.FunType == LineType_Enum.XnXYnY)
{
if(edg.Start.Y <= edg.End.Y)
{
                    List<Point> jiaodainList = edg.Intersection.OrderBy(m =>
m.X).ThenBy(m => m.Y).ToList();
for(int i =0; i < jiaodainList.Count -1; i++)
{
                        Point start = jiaodainList[i];
                        Point end = jiaodainList[i +1];
                        Point z =new Point(start.X +((end.X - start.X)/2),
start.Y +((end.Y - start.Y)/2));
if(z.CheckIfInArea(area))
```

```
{
continue;
}
else
{
returnfalse;
}
}
}
else
{
                    List<Point> jiaodainList = edg.Intersection.OrderBy(m =>
m.X).ThenByDescending(m => m.Y).ToList();
for(int i =0; i < jiaodainList.Count -1; i++)
{
                        Point start = jiaodainList[i];
                        Point end = jiaodainList[i +1];
                        Point z =new Point(start.X +((end.X - start.X)/2),
start.Y -((start.Y - end.Y)/2));
if(z.CheckIfInArea(area))
{
continue;
}
else
{
returnfalse;
}
}
}
}
}
}
}
else
{
returntrue;
}
returntrue;
}
```

附录 7

```
public PointWithLineSagementState_EnumCheckPointInLineSagement(LineSagement
sagement)
{
double px =this.X;
double py =this.Y;
//bool flag = false;

   Point pi = sagement.Start;
   Point pj = sagement.End;
double sx = pi.X;double sy = pi.Y;
double tx = pj.X;double ty = pj.Y;

//点与线段顶点重合
bool psTf =(px == sx && py == sy);
bool ptTf =(px == tx && py == ty);
if(psTf || ptTf)
{
return PointWithLineSagementState_Enum.VertexOverlap;
}
switch(sagement.FunType)
{
case LineType_Enum.XX:
if(px == pi.X &&((py <= sy && py >= ty)||(py >= sy && py <= ty)))
return PointWithLineSagementState_Enum.OnLineSagement;
break;
case LineType_Enum.YY:
if(py == pi.Y &&((px >= sx && px <= tx)||(px <= sx && px >= tx)))
return PointWithLineSagementState_Enum.OnLineSagement;
break;
case LineType_Enum.XnXYnY:
default:
break;
}
//判断线段端点是否在射线两侧
if((sy < py && ty >= py)||(sy >= py && ty < py))
{
//线段上与射线 Y 坐标相同的点的 X 坐标
double x = sx +(py - sy)*(tx - sx)/(ty - sy);
```

```
//点在线段上
if(x == px)
{
return PointWithLineSagementState_Enum.OnLineSagement;
}
//射线穿过线段
if(x > px)
{
return PointWithLineSagementState_Enum.Cross;
}
}
    return PointWithLineSagementState_Enum.UnCross;}
```

附录 8

```
publicbool CheckIfInArea(Area theArea)
{
int cnt =0;
foreach(LineSagement lineSagement in theArea.LineSagements)
{
switch(CheckPointInLineSagement(lineSagement))
{
case PointWithLineSagementState_Enum.Cross:
             cnt +=1;
break;
case PointWithLineSagementState_Enum.OnLineSagement:
returntrue;
case PointWithLineSagementState_Enum.VertexOverlap:
returntrue;
case PointWithLineSagementState_Enum.UnCross:
default:
break;
}
}
//射线穿过多边形边界的次数为奇数时点在多边形内
if(cnt %2==1)
{
returntrue;//点在多边形内
}
else
{
```

```
returnfalse;//点不在多边形内
}
}
```

附录 9

```
publicbool CheckIfInChildArea(Area theArea)
{
int cnt =0;
foreach(LineSagement lineSagement in theArea.LineSagements)
{
switch(CheckPointInLineSagement(lineSagement))
{
case PointWithLineSagementState_Enum.Cross:
             cnt +=1;
break;
case PointWithLineSagementState_Enum.OnLineSagement:
returnfalse;
case PointWithLineSagementState_Enum.VertexOverlap:
returnfalse;
case PointWithLineSagementState_Enum.UnCross:
default:
break;
}
}
//射线穿过多边形边界的次数为奇数时点在多边形内
if(cnt %2==1)
{
returntrue;//点在多边形内
}
else
{
returnfalse;//点不在多边形内
}
}
```

附录 10

```
public bool WhetherPolygonIntersection(Area father)
{
   List<LineSagement> childEdgeXfatherEdge_List =new List<LineSagement>();
foreach(LineSagement edg inthis.LineSagements)
```

```
{
        Point a = edg.Start;
        Point b = edg.End;
foreach(LineSagement fatherEdge in father.LineSagements)
{
            Point c = fatherEdge.Start;
            Point d = fatherEdge.End;

double denominator =(b.Y - a.Y)*(d.X - c.X)-(a.X - b.X)*(c.Y - d.Y);
// 如果分母为 0 则平行或共线, 不相交
if(denominator ==0)
{
//竖线
if(edg.FunType == LineType_Enum.XX)
{
//共线
if(edg.Start.X == fatherEdge.Start.X)
{
//不重叠
if(b.Y > c.Y || a.Y < d.Y)
{
continue;
}
//完全重叠
if(a.Y == c.Y && b.Y == d.Y)
{
continue;
}
//上跨立（包含两线相接）
if(a.Y > c.Y && b.Y <= c.Y && b.Y >= d.Y)
{
                        edg.Intersection.Add(c);
continue;
}
//下跨立（包含两线相接）
if(a.Y <= c.Y && a.Y >= d.Y && b.Y < d.Y)
{
                        edg.Intersection.Add(d);
continue;
}
```

```
//父包子
if(c.Y >= a.Y && d.Y <= b.Y)
{
continue;
}
//子包父
if(a.Y >= c.Y && b.Y <= d.Y)
{
                        edg.Intersection.Add(c);
                        edg.Intersection.Add(d);
continue;
}
}
//平行
else
{
continue;
}
}

//横线
elseif(edg.FunType == LineType_Enum.YY)
{
//共线
if(edg.Start.Y == fatherEdge.Start.Y)
{
//不重叠
if(b.X < c.X || a.X > d.X)
{
continue;
}
//完全重叠
if(a.X == c.X && b.X == d.X)
{
continue;
}
//左跨立（包含两线相接）
if(a.X < c.X && b.X >= c.X && b.X <= d.X)
{
                        edg.Intersection.Add(c);
```

```
continue;
}
//右跨立（包含两线相接）
if(b.X > d.X && a.X >= c.X && a.X <= d.X)
{
                        edg.Intersection.Add(d);
continue;
}
//父包子
if(c.X <= a.X && d.X >= b.X)
{
continue;
}
//子包父
if(a.X <= c.X && b.X >= d.X)
{
                        edg.Intersection.Add(c);
                        edg.Intersection.Add(d);
continue;
}
}
//平行
else
{
continue;
}
}
//斜线
elseif(edg.FunType == LineType_Enum.XnXYnY)
{
//共线
if(edg.Params.First().Equals(fatherEdge.Params.First())&&
edg.Params.Last().Equals(fatherEdge.Params.Last()))
{
//不重叠
if((a.X < c.X && b.X < c.X)
||(a.X > d.X && b.X > d.X))
{
continue;
}
```

```
//完全重叠
if(a.X == c.X && a.Y == c.Y && b.X == d.X && b.Y == d.Y)
{
continue;
}
//跨立（包含两线相接）
if(a.X < c.X && b.X >= c.X && b.X <= d.X)
{
                        edg.Intersection.Add(c);
continue;
}
//跨立（包含两线相接）
if(a.X >= c.X && a.X <= d.X && b.X > d.X)
{
                        edg.Intersection.Add(d);
continue;
}
//父包子
if(a.X >= c.X && a.X <= d.X && b.X >= c.X && b.X <= d.X)
{
continue;
}
//子包父
if(a.X <= c.X && b.X >= d.X)
{
                        edg.Intersection.Add(c);
                        edg.Intersection.Add(d);
continue;
}
}
//平行
else
{
continue;
}
}
}
// 线段所在直线的交点坐标 (x , y)
double x =((b.X - a.X)*(d.X - c.X)*(c.Y - a.Y)
+(b.Y - a.Y)*(d.X - c.X)* a.X
```

```
-(d.Y - c.Y)*(b.X - a.X)* c.X)/ denominator;
double y =-((b.Y - a.Y)*(d.Y - c.Y)*(c.X - a.X)
+(b.X - a.X)*(d.Y - c.Y)* a.Y
-(d.X - c.X)*(b.Y - a.Y)* c.Y)/ denominator;
// 判断交点是否在两条线段上
if(
// 交点在线段 1 上
(x - a.X)*(x - b.X)<=0&&(y - a.Y)*(y - b.Y)<=0
// 且交点也在线段 2 上
&&(x - c.X)*(x - d.X)<=0&&(y - c.Y)*(y - d.Y)<=0)
{
             edg.Intersection.Add(new Point(x, y));
}
else
{
continue;
}
}
}
if(this.LineSagements.Where(m => m.Intersection.Any()).Any())
{
returntrue;
}
else
{
returnfalse;
    }
}
```