半动态矩形交查询算法*

高静波 李新友 唐泽圣 周晓辉

(清华大学计算机科学与技术系CAD中心 北京 100084)

摘要 本文讨论了动态矩形交查询算法.文中介绍了两个半动态矩形查询的新算法,它们分别基于一维数据结构和二维数据结构.一维查询算法的查询时间复杂度是$O(\log M+k')$,更新时间复杂度是$O(\log M\log n)$,空间复杂度是$O(n\log M)$.二维查询算法的查询时间复杂度是$O(\log^2 M+k)$,更新时间复杂度是$O(\log^2 M\log n)$,空间复杂度是$O(n\log^2 M)$.本文分别实现了这两个算法,通过对它们的性能进行比较,发现一维查询算法是一种高效、实用的算法.

关键词 计算几何,矩形交查询,动态查询.

中图法分类号 TP391

矩形交查询问题可以这样描述:已知平面上n个轴向安置矩形的集合$\{P_1,P_2,\ldots,P_n\}$(轴向安置矩形是指边平行于坐标轴的矩形).以一个轴向安置矩形Q作为查询矩形,报告集合中所有与查询矩形Q相交的矩形.它属于计算几何中的一类问题——正交形体交查询问题.正交形体是指n维空间中轴向安置的点、直线段、矩形、立方体、超立方体等形体.常见的问题有:

· 范围搜索:n维空间中有一组点,以一个轴向安置的超立方体作为查询形体,报告超立方体所包含的所有的点.

· 点包含:n维空间中有一组轴向安置的超立方体,以一个点作为查询形体,报告所有包含该点的超立方体.

· 正交线段交查询:在平面中有一组水平(或垂直)的直线段,以一个垂直(或水平)的直线段作为查询形体,报告与该线段相交的所有线段.

· 矩形交查询:在平面中有一组轴向安置的矩形,以一个轴向安置的矩形作为查询形体,报告与该矩形相交的所有矩形.

上述这些问题之间有十分密切的关系.它们可以用相似的方法解决.这类问题之间可以相互转换.例如:范围搜索问题和点包含问题是对偶问题;矩形交查询问题可以转换成优势查询问题(属于范围搜索问题)[1];矩形交查询问题还可以转换成3个子问题——范围搜索、

* 本文研究得到国家863高科技项目基金资助.作者高静波,1970年生,博士生,主要研究领域为工程图自动识别,计算几何.李新友,1962年生,博士,副教授,主要研究领域为计算机图形学,工程图自动识别.唐泽圣,1932年生,教授,博士导师,主要研究领域为计算机图形学,计算几何,工程图自动识别,科学计算可视化.周晓辉,1974年生,本科生,主要研究领域为计算机图形学.

本文通讯联系人:高静波,北京 100084,清华大学计算机科学与技术系CAD中心

本文1996-09-16收到修改稿

点包含、正交线段交查询.[2,3]由于这些问题之间的关系是如此密切,目前有许多算法都能同时解决它们中的若干个问题.

根据处理的问题方式,矩形交查询问题可以分为批处理方式[4,5]和查询方式两类.批处理方式的矩形交查询问题可以用平面扫描方法[4]或分治方法[5]解决.目前已有的算法可以用$O(n\log n+k)$的时间和$O(n)$的空间解决该问题.

查询方式的矩形交查询问题又可以进一步分为静态问题和动态问题两类.静态问题是指矩形集合在查询过程中不发生变化.静态算法主要支持两个操作:矩形集合的初始化操作和查询操作.动态问题是指矩形集合在查询过程中会发生变化——增加新的矩形或者删除已有的矩形.动态算法主要支持3个操作:增加矩形、删除矩形和查询.

本文将详细讨论动态矩形交查询问题.第1节对现有的算法进行总结.第2、3节介绍2种不同的半动态算法.第2节的算法是基于一维查询的,它的查询时间复杂度是$O(\log M+k')$,更新时间复杂度是$O(\log M\log n)$,空间复杂度是$O(n\log M)$.其中k'不是满足查询条件的矩形数目,而是一维查询时在一个坐标轴上满足查询条件的矩形的数目.第3节的算法是基于二维查询的,它的查询时间复杂度是$O(\log^2 M+k)$,更新时间复杂度是$O(\log^2 M\log n)$,空间复杂度是$O(n\log^2 M)$.最后通过实验方法给出这两种算法的性能比较.

1 已有的算法

首先要说明的是矩形之间相交的含义.矩形A与B相交包括如下3种情况:A包含B,B包含A,A和B的边相交.查询时应该把这3种情况都报告出来.然而在目前的文献中,一些文章介绍的算法只能报告矩形包含的情况.[6]

矩形交查询问题的解决思路可以分为两类:第1,直接设计新的算法和数据结构;第2,将矩形交查询问题转换成其它的问题.矩形交查询问题可以转换成四维空间中的范围查询问题,也可以转换成平面中的范围查询、点包含和正交线段交查询问题.所以,范围查询、点包含和正交线段交查询问题与矩形交查询问题有密切的关系,这3个问题的任何高效的算法都可以用于矩形交查询问题中.表1总结了这4种问题的最优算法复杂度.

由于矩形交查询问题可以转化成范围查询、点包含、正交线段交查询3个子问题,因此可以将每个子问题分别用各自的最优算法求解,最后将它们的解合并起来.这就是表1中列出的最优合成算法.显然,整个算法的复杂度取决于3个子算法中最坏的一个.从表中可以看出,点包含问题是一个关键问题,它的复杂度决定了整个问题的复杂度.基于Edelsbrunner[7]的2个点包含算法,我们可以得到2个最优合成算法.它们在空间复杂度和更新时间复杂度上各有优势.第1个算法的空间复杂度较高,第2个算法的更新时间复杂度较高.从表中可以看出,最优合成算法是目前最好的算法.然而每个子问题的最优算法都要采用十分复杂的设计,所以最优合成算法实现起来比较复杂,不是一个实用的算法.而且各个最优子算法设计思想各异,使用不同的数据结构,难以统一到一个整体的框架中去,不能算是一个完美的算法.

Chazelle[3]的算法基于与最优合成算法同样的思路,把矩形交查询问题分解为范围查询、点包含、正交线段交查询3个子问题.然而他使用了一种新的思路(功能方法)来解决3

个子问题.他得到动态矩形并查询算法的复杂度如下:空间复杂度为 $O(n)$,查询时间复杂度为 $O(\log^3 n+k(\log^2 n/k)^2)$,最坏的更新时间复杂度为 $O(\log^2 n)$.与最优合成算法相比,该算法将空间复杂度降到了线性的复杂度,但是它的查询复杂度比较高.虽然 Chazelle 用一种统一的思想设计 3 个子算法,但是 3 个子算法还是有很大的差异.与最优合成算法一样,该算法也有实现复杂、形式不统一的缺点.

表 1　4 种问题的算法复杂度总结

问题及算法	空间	查询时间	增加元素时间	删除元素时间
范围查询				
2 维 Overmarse[8]	$O(n\log n)$	$O(\log^2 n+k)$	$O(\log^2 n)$	$O(\log n)$
k 维 Willard[9]	$O(n\log^{k-1} n)$	$O(\log^{k-1/2} n)$	$O(\log^{k-1/2} n)$	$O(\log^{k-1/2} n)$
点包含				
Edelsbrunner[7]	$O(n\log^2 n)$	$O(\log^2 n+k)$	$O(\log^2 n)$	$O(\log^2 n)$
Edelsbrunner[7]	$O(n\log n)$	$O(\log^2 n+k)$	$O(\log^3 n)$	$O(\log^3 n)$
Chen[10]	$O(n)$	$O(\log^3 n+k)$	$O(\log^2 n)$	$O(\log^2 n)$
正交线段交查询				
Chen[10]	$O(n)$	$O(\log^2 n+k)$	$O(\log n)$	$O(\log n)$
矩形交查询				
Bistiolas[6]	$O(n\log^2 n)$	$O(\log^3 n+k)$	$O(\log^3 n)$	$O(\log^3 n)$
Lee[1]	$O(n\log^3 n)$	$O(\log^3 n+k)$	$O(\log^4 n)$	$O(\log^4 n)$
Chazelle[3]	$O(n)$	$O(\log^3 n+k(\log^2 n/k)^2)$	$O(\log^2 n)$	$O(\log^2 n)$
最优合成算法 1	$O(n\log^2 n)$	$O(\log^2 n+k)$	$(\log^2 n)$	$O(\log^2 n)$
最优合成算法 2	$O(n\log n)$	$O(\log^2 n+k)$	$O(\log^3 n)$	$O(\log^3 n)$
半动态一维查询算法	$O(n\log M)$	$O(\log M+k')$	$O(\log M\log n)$	$O(\log M\log n)$
半动态二维查询算法	$O(n\log^2 M)$	$O(\log^2 M+k)$	$O(\log^2 M\log n)$	$O(\log^2 M\log n)$

矩形交查询问题还可以转换成范围搜索问题,Lee 的算法[1]就是基于这个思路.该算法可以解决批处理方式的矩形交查询问题、静态的和动态的矩形交查询问题.对于动态的矩形交查询问题,该算法的空间复杂度为 $O(n\log^3 n)$,查询时间复杂度为 $O(\log^3 n+k)$,最坏的更新时间复杂度为 $O(\log^4 n)$.这是一个简洁的算法,但是它的复杂度太高.

Bistiolas[6]的算法是最新的用一个专门的数据结构解决矩形包含查询的算法.这个算法把范围搜索树和优先搜索树结合起来,构造了一种 3 层的数据结构.它可以解决静态的和动态的矩形包含查询问题.该算法的空间复杂度为 $O(n\log^2 n)$.对于静态问题的查询时间复杂度为 $O(\log^2 n\log\log n+k)$.对于动态问题的查询时间复杂度为 $O(\log^3 n+k)$,平均的更新时间复杂度为 $O(\log^3 n)$.这个算法的复杂度虽然不是最好的,但是它具有简洁、易实现、效率高的优点.然而它只解决了矩形包含查询问题.

表 1 最后列出了本文要介绍的半动态一维查询算法和半动态二维查询算法.与 Bistiolas 的算法相比,这两个算法的优点是简洁、高效.半动态二维查询算法与 Bistiolas 的算法相比有两个优点:第 1,Bistiolas 的算法只解决了矩形包含查询问题,而半动态二维查询算法解决了矩形交查询问题.第 2,Bistiolas 的算法的查询时间复杂度比半动态二维查询算法高一个 $\log n$ 因子.要注意的是,半动态二维查询算法复杂度中包含了 $\log M$ 因子,而 Bistiolas 的算法复杂度中则是 $\log n$ 因子.但是在实际应用中矩形个数较多时,可以认为 $\log M$ 和 $\log n$ 是一个量级的.

半动态一维查询算法比较特殊,它的复杂度与当前已有的算法不好比较.关键在于它的查询时间复杂度$O(\log M+k')$中的k'因子.其它算法的查询复杂度中的k是满足查询条件的矩形数目,而半动态一维查询算法中的k'是在一个坐标轴上满足查询条件的矩形的数目.k和k'的关系不能确定,它依赖于矩形在平面上的分布情况.在理论分析上$O(\log M+k')$不是一个很好的复杂度.但是由于实现的简单性,而且空间复杂度和更新时间复杂度很好,半动态一维查询算法在实际使用中是一个很好的算法.

2 一维查询算法

矩形相交的条件可分解为一维线段相交的条件:两个矩形相交当且仅当它们在X轴上的投影相交且它们在Y轴上的投影相交.因此我们首先研究一维上的区间交查询算法.

两个区间A和B相交的条件可以划分为两种情况:A的左端点在B中;B的左端点在A中.这两种情况完全覆盖了两个区间相交的条件,并且它们没有重复的情况.为了简化问题,我们在文章中暂时不讨论各种边界情况,因此我们假设区间的端点不会重合.在算法完成后,经过仔细的考虑,就可以容易地处理好边界情况.

基于区间相交的两种情况,我们可以把区间交查询问题分解为两个子问题:第1,报告查询区间的左端点在哪些区间内;第2,报告查询区间包含哪些区间的左端点.第1个子问题可以用线段树来解决.[4]第2个子问题可以用平衡二叉树解决.在静态问题中,这2个子问题都可以在$O(\log n+k)$的时间内完成一次查询.在动态问题中,由于线段树在插入和删除线段时涉及到树的重新平衡操作,因此更新的复杂度要超过$O(\log n)$;而平衡二叉树则可以在$O(\log n)$的时间内完成更新操作.

为了避免线段树的重新平衡操作,我们使用了一种半静态的数据结构.假设线段端点坐标的取值范围是$[1..M]$.我们首先建立一棵骨架树T_b,该树的树根代表整个区间$[1..M]$.树中的每个结点代表的区间都均分为两个子区间,分别赋给该结点的两个子结点.骨架树的叶子结点代表最小的区间$[i..i+1]$.在线段的插入和删除过程中,骨架树的结构保持不变.显然,骨架树的高度为$\log M$.树中有$2M-1$个结点.当把线段存放到骨架树中时,每个线段最多出现在$2\log M-1$个结点中.所以空间复杂度为$O(n\log M+M)$.这个结构中,有一些树的结点是浪费的,因为有些结点不包含任何线段.因此,如果一个结点及其子结点不包含任何线段,就删除该结点及其对应的子树.这样,空间复杂度可以降到$O(n\log M)$.

基于骨架树T_b,我们定义一个统一的数据结构来完成两个子问题的查询.我们在骨架树的每个结点上增加两个表:A表存放的线段完全包含了结点对应的区间,但是不完全包含它的父结点对应的区间;B表存放了左端点落在结点对应区间的线段.我们称这种数据结构为一维查询树T_1.图1是查询过程和插入线段过程的描述.假设查询或插入的线段是Q,当前树结点是T,当前树结点对应的区间是P.

显然,查询时最多访问$2\log M-1$个结点.即查询的时间复杂度为$O(\log M+k)$.同样,进行更新操作时最多访问$2\log M-1$个结点,每个结点上的A表和B表用树结构组织,可以在$O(\log n)$的时间将一个线段加入到树结点的A表和B表中或从中删除.因此更新操作的时间复杂度为$O(\log M\log n)$.现在考虑空间复杂度.第1,由于线段的左端点只会位于一个最小区间内,它必定在树的每一层上出现一次,所以一个线段最多位于$\log M$个结点的B表

中.第 2,由于一个线段一旦完全包含了一个树结点对应的区间,它就存放在该结点中而不会出现在该结点的子结点中,所以一个线段最多出现在 $2\log M-1$ 个树结点的 A 表中.总之,空间复杂度为 $O(n\log M)$.

A 表

```
过程 查询(T,Q)
开始
    如果 P 和 Q 不相交,则返回
    如果 Q 的左端点落在 P 内,
      则报告 T 的 A 表存放的线段
    如果 Q 完全包含 P,
      则报告 B 表存放的线段;
    否则
      开始
        查询(T 的左子树,Q)
        查询(T 的右子树,Q)
      结束
结束
```

B 表

```
过程 插入(T,Q)
开始
    如果 P 和 Q 不相交,则返回
    如果 Q 的左端点落在 P 内,
      则将 Q 存放到 T 的 B 表中
    如果 Q 完全包含 P,
      则将 Q 存放到 T 的 A 表中;
    否则
      开始
        插入(T 的左子树,Q)
        插入(T 的右子树,Q)
      结束
结束
```

图 1 查询算法和插入算法

我们可以直接利用一维查询树 T_1 进行二维的矩形交查询.首先将矩形集合在一个坐标轴方向投影,得到一个线段集合.用一维查询树 T_1 组织这些线段.然后利用查询矩形在同一坐标轴上的投影进行查询,得到一个矩形集合 R^* 作为中间结果.最后检查集合 R^* 中的每个矩形,将满足相交条件的矩形报告出来.显然,这种算法的查询复杂度是 $O(\log M+k')$.其中 k' 是中间结果集合 R^* 的大小.

在矩形查询的一维查询算法中,k' 是影响查询时间复杂度的关键因素.k' 一般要比最终的查询结果个数 k 大很多.这取决于矩形集合在平面上的分布情况以及查询矩形的大小和位置.假设矩形集合在平面上均匀分布,查询矩形的宽和高分别为 w 和 h.一维查询涉及的平面区域是整个平面的一个横条($M\times h$ 大小)或纵条($w\times M$ 大小),而最终的查询结果应该只涉及平面中的一个区域($w\times h$ 大小).显然 k' 和 k 的比应该与两个面积的比有关.即 $k'/k=O(M/w)$(沿高度方向查询)或 $k'/k=O(M/h)$(沿宽度方向查询).

一维查询算法的查询复杂度是 $O(\log M+k')$.它包含两部分.$\log M$ 代表树查询时间,它比其它算法要好.这是该算法的优点,使得算法易于实现、效率高.k' 代表报告时间,它比其它算法要差.很难判断这两个因素哪个起主要作用,只能在实际问题中通过实践来检验.我们将在第 4 节中介绍一个实际的应用问题,并根据该问题对一维查询算法的性能进行评价.

3 二维查询算法

在一维查询树 T_1 中,每个树结点上有两个表:A 表和 B 表.对这两个表的直接查询导致了查询复杂度中的 k' 因子.一维查询算法相当于在一个坐标轴方向进行折半查询,在另一个坐标轴方向进行线性查询.现在我们在 T_1 的每个结点中,用同样的一维查询树组织 A 表和 B 表,形成一个两层的树结构.这样在两个坐标轴方向都可以进行折半查询.从而降低了查询的时间复杂度.

我们在一维查询树 T_1 的基础上定义二维查询树 T_2.二维查询树是一个两层的数据结

构.外层树与就是一个一维查询树.只是每个树结点上的A表和B表被两个一维查询树代替.这些一维查询树被称为内层树.外层树按矩形在X方向投影所得的线段来组织,内层树按矩形在Y方向投影所得的线段来组织.

查询时先按矩形X方向的投影在外层树用一维查询算法进行查询,最多可以得到$3\log M-1$个相关的内层树(A表中最多有$\log M$项;B表中最多有$2\log M-1$项).然后按矩形在Y方向的投影在这些内层树中用一维查询算法进行查询.显然,查询操作的时间复杂度为$O(\log^2 M+k)$.

插入操作与查询操作类似,在外层树上按X方向的投影进行插入操作,最多涉及到$3\log M-1$个相关的内层树(A表中最多有$2\log M-1$项;B表中最多有$\log M$项).然后在这些内层树上按Y方向的投影进行插入操作.显然,插入操作的时间复杂度为$O(\log^2 M\log n)$.

现在考虑空间复杂度.一个矩形最多出现在$O(\log M)$个内层树中,在每个内层树中的空间复杂度为$O(\log M)$.所以总的空间复杂度为$O(n\log^2 M)$.

4 算法应用及性能比较

在一个工程图自动识别系统中我们使用了矩形交查询的算法.这个系统通过分析工程图中连通域的大小和位置将字符提取出来并进行识别.[11]同时也可以将工程图中的直线、圆弧等图素识别出来.通过进一步分析字符、直线、圆弧等图素的位置关系,还可以进行更高层次的图素的识别,最终达到工程图理解的目的.在这里,图素之间的位置关系对工程图的自动识别和理解十分重要.

为了利用图素之间的位置关系,就要经常做按位置查询的操作:给定一个图素,找出与该图素邻近的其它图素.我们用图素的外接矩形表示图素的位置,这个操作就可以转化成一个矩形交查询问题.把一个图素的外接矩形向四周扩展一定的尺寸,以扩展后的矩形作为查询矩形进行矩形交查询,得到的矩形就对应着与该图素邻近的图素.

我们利用从工程图自动识别系统中产生的实际数据对一维矩形查询算法和二维矩形查询算法进行了性能测试.在工程图自动识别系统运行时会不断地进行插入矩形、删除矩形和查询的操作,我们把这些操作记录在一个文件中.为了便于测试,我们把矩形的更新操作和查询操作分开测试.因此在数据文件中只包含插入和删除操作.测试时首先根据数据文件进行矩形的插入和删除操作,建立一个矩形集合,同时记录操作时间和最终占用的空间.然后在矩形集合上进行查询操作.查询矩形的大小根据对实际查询矩形的统计采用一个固定的值.查询矩形的位置在测试时随机产生.最后记录查询操作所用的时间.

表2 性能测试—问题1

$M=700, N_a=689, N_b=60$

	空间(KB)	更新时间(ms)	查询时间(ms)
线性算法	16.5	0.160	6.168
一维算法	110.0	0.726	1.424
二维算法	230.7	2.888	1.461

表3 性能测试—问题2

$M=4\,000, N_a=7\,459, N_b=3\,130$

	空间(KB)	更新时间(ms)	查询时间(ms)
线性算法	180.0	0.240	32.360
一维算法	717.5	0.477	1.129
二维算法	8 814.5	2.022	3.177

我们测试了3个查询算法的性能.包括一维查询算法、二维查询算法和一个线性的查询算法.我们用线性的查询算法作为参照的基准.表2~4总结了在3个实际问题中各种算法

所占用的空间 S、一次查询操作的时间 Q 和一次更新操作的时间 U. 在表中 M 代表整数坐标的范围,N_a 代表增加操作的数目,N_b 代表删除操作的数目,查询矩形的大小为 200×200,位置随机生成.

表 4　性能测试—问题 3

$M=13\ 000, N^a=257\ 895, N_b=157\ 272$

	空间(KB)	更新时间(ms)	查询时间(ms)
线性算法	5 164.5	2.054	725.560
一维算法	11 204.7	0.524	5.140
二维算法	*	*	*

表 5　一维算法的查询效率

问题	M	w	k'/k	$(k'/k)/(M/w)$
一	700	100	78.53	11.22
		200	2.75	0.79
二	4 000	100	75.44	1.89
		200	16.46	0.82
三	13 000	100	147.89	1.14
		200	87.63	1.35

从表中可以看出,一维查询算法的性能是最好的. 从占用的空间看,线性算法占用的空间最小;二维查询算法占用的空间最大,甚至在处理问题三时,由于空间不够而没有完成测试;一维查询算法占用的空间处于二者之间. 从查询时间看,一维查询算法远远优于线性算法,而且比二维查询算法稍好. 从更新时间看,一维查询算法比二维查询算法好,而且在大规模的问题中要比线性算法好. 经过实验验证,我们可以得出结论,一维查询算法要比二维查询算法优越.

从理论上分析,一维查询算法的查询时间复杂度是 $O(\log M+k')$;二维查询算法的查询时间复杂度是 $O(\log^2 M+k)$. 查询时间复杂度难以比较. 我们可以把查询的时间分为搜索时间和报告时间. 一维算法的搜索时间 $O(\log M)$ 小于二维算法的搜索时间 $O(\log^2 M)$;而一维算法的报告时间 $O(k')$ 大于二维算法的报告时间 $O(k)$. 因此 k' 的大小是决定一维算法效率的关键. 我们可以用比值 k'/k 衡量一维查询算法的效率. 在第 2 节中我们已得出结论,k'/k 取决于矩形集合在平面上的分布情况以及查询矩形的大小和位置. 假设矩形集合在平面上均匀分布,查询矩形的宽和高分别为 w 和 h,则 $k'/k=O(M/w)$ 或 $k'/k=O(M/h)$. 从表 5 列出的数字可以看出,k'/k 和 M/w 是比较接近的.

5　总　结

本文对动态矩形交查询算法进行了详细的讨论. 首先回顾了当前已有的算法,然后介绍了两个新算法. 它们分别基于一维数据结构和二维数据结构. 一维查询算法的查询时间复杂度是 $O(\log M+k')$,更新时间复杂度是 $O(\log M\log n)$,空间复杂度是 $O(n\log M)$;二维查询算法的查询时间复杂度是 $O(\log^2 M+k)$,更新时间复杂度是 $O(\log^2 M\log n)$,空间复杂度是 $O(n\log^2 M)$. 我们的二维算法在查询复杂度上优于 Bistiolas[6] 的算法,其它方面与它是同一量级.

在理论分析上,一维算法的查询复杂度$O(\log M+k')$不如二维算法.但是实际上一维查询算法实现简单,查询时要做的操作很少;二维算法比较复杂,查询时要做的操作很多.二维算法的查询速度反倒不如一维算法.然而一维算法的空间复杂度和更新时间复杂度很好,所以一维算法要比二维查询算法优越得多.因此可以得出结论,一维查询算法虽然在理论分析上没有很好的查询时间复杂度,但是在实际应用中它是一个很好的算法.

参考文献

1 Lee D T, Wong C K. Finding intersection of rectangles by range search. Journal of Algorithms, 1981, **2**: 337~347.

2 Edelsbrunner H. Intersection problems in computational geometry [Ph. D. thesis]. Tech. Rep. F-93, Technical Univ. Graz, Graz, Austria, 1982.

3 Chazelle B. A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput., 1988, **17**: 427~462.

4 Bentley J L, Wood D. An optimal worst case algorithm for reporting intersections of rectangles. IEEE Trans. on Comput., 1980, **C29**: 571~577.

5 Guting R H, Wood D. Finding rectangle intersections by divide-and-conquer. IEEE Trans. on Comput., 1984, **33**: 671~675.

6 Bistiolas V, sofotassios D, Tsakalidis A. Computing rectangle enclosures. Computational Geometry: Theory and Applications, 1993, **2**: 303~308

7 Edelsbrunner H. Dynamic data structure for orthogonal intersection queries. Tech. Report No. 59, Institute fur Informationsverarbeitung, Technische Universitat Graz, 1980.

8 Overmars M H, Leeuwen J V. Worst-case optimal insertion and deletion methods for decomposable searching problems. Information Processing Letters, 1981, **12**: 168~173.

9 Willard D E. New data structure for orthogonal range queries. SIAM J. Comput., 1985, **14**: 232~253.

10 Chen S W, Janardan R. Efficient dynamic algorithms for some geometric intersection problems. Information Processing Letters, 1990, **36**: 251~258.

11 Gao J, Tang L, Liu W *et al*. Segmentation and recognition of dimension texts in engineering drawings. In: Int. Conf. on Document Analysis and Recognition, 1995. 528~531.

SEMI-DYNAMIC RECTANGLE INTERSECTION SEARCHING ALGORITHM

GAO Jingbo LI Xinyou TANG Zesheng ZHOU Xiaohui

(*CAD Center Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract This paper introduces a solution to dynamic rectangle intersection searching problem. There are two semi-dynamic algorithms which are based on 1-dimensional data structure and 2-dimensional data structure respectively. The computational complexity of 1-D searching algorithm is as follows: query time $O(\log M + k')$, update time $O(\log M \log n)$, space $O(n \log M)$. The computational complexity of 2-D searching algorithm is as follows: query time $O(\log^2 M + k)$, update time $O(\log^2 M \log n)$, space $O(n \log^2 M)$. The two algorithms are implemented respectively. With an experimental comparison, the authors found that 1-D searching algorithm is far better than 2-D searching algorithm.

Key words Computational geometry, rectangle intersection searching, dynamic searching.

Class number TP391