UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

Finding Rectangle Intersections by Divide-and-Conquer

Ralf Hartmut Güting
Derick Wood

Data Structuring Group
CS-83-03

March, 1983

FINDING RECTANGLE INTERSECTIONS BY DIVIDE-AND-CONQUER[1]

Ralf Hartmut Güting[2]
Derick Wood[3]

ABSTRACT

In this paper we reconsider three geometrical problems for which we develop divide-and-conquer algorithms. The first problem is to find all pairwise intersections among a set of horizontal and vertical line segments. The second is to report all points enclosures occurring in a mixed set of points and rectangles and the third is to find all pairwise intersections in a set of iso-oriented rectangles. We derive divide-and-conquer algorithms for the first two problems which are then combined to solve the third. In each case a space- and time-optimal algorithm is obtained, that is $O(n)$ space and $O(n \log n + k)$ time, where n is the number of given objects and k the number of reported pairs.

These results show that divide-and-conquer can be used in place of line sweep, without additional asymptotic-cost, for some geometrical problems. This raises the natural question: For which class of problems are the line-sweep and divide-and-conquer paradigms interchangeable?

1. INTRODUCTION

We reconsider the Rectangle Intersection Problem, that is finding all intersecting pairs among a set of n iso-oriented rectangles. This is an

[1] The work of the first author was carried out under a DAAD (Deutscher Akademischer Austauschdienst) Grant, while visiting McMaster University, and that of the second author under Natural Sciences and Engineering Research Council of Canada Grants Nos. A-5692 and A-7700.

[2] Lehrstuhl Informatik VI, Universität Dortmund, D-4600 Dortmund 50, West Germany.

[3] Data Structuring Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

important step in checking the design rules of VLSI-circuitry (see Baird [B], Lauther [L1] or Mead and Conway [MC]) and has other applications as well, for instance in architectural data bases [EL]. For more details and motivation see [BW].

The first time-optimal solution for the problem was given by Bentley and Wood [BW], that is $O(n\log n+k)$ time, where k is the number of intersecting pairs. Almost immediately Edelsbrunner [E] and McCreight [M] independently found space- and time-optimal solutions. However, each of these algorithms is based on the line-sweep paradigm. A natural question is whether or not this problem can be solved efficiently using divide-and-conquer (see for instance [AHU] for an introduction to this paradigm). Lee [L2] the first to investigate this approach produced a divide-and-conquer algorithm requiring $O(n\log n\log^* n+k)$ time.

In this paper we develop a space- and time-optimal divide-and-conquer algorithm for the rectangle intersection problem. This raises the question: for which class of problems the line sweep and divide-and-conquer paradigm are interchangeable with the same space and time bounds.

In Section 2 we recall how the rectangle intersection problem can be divided into two subproblems, the line segment intersection problem and the point enclosure problem. In Section 3 we describe a divide-and-conquer algorithm for the line segment intersection problem, while in Section 4 a divide-and-conquer algorithm for the point enclosure problem follows. In Section 5 we put the pieces together to solve the original problem and finally discuss the results and give some directions for further work in Section 6 .

2. THE SUBPROBLEMS

The *Rectangle Intersection Problem* is:

Given a set of n iso-oriented rectangles, report all pairwise intersections.

Two rectangles intersect if and only if either their edges intersect or one encloses the other one completely.

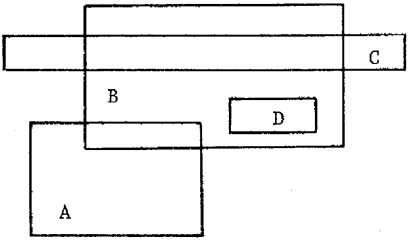

Figure 1

In Figure 1 the pairs of rectangles (A,B) and (B,C) intersect by edge intersection, while (B,D) constitutes a rectangular enclosure. The two types of intersections can be found efficiently by treating them separately. We follow [BW] in this approach. The result is a 2-stage algorithm which finds all edge intersections in the first step and all rectangular enclosures in the second.

In the first step, in Section 3, we consider the set of all edges composing the rectangles, that is $4n$ edges and solve the line segment intersection problem for them. In the second, in Section 4, we extend the set of rectangles to include an interior point for each rectangle. We then solve the point enclosure problem for the resulting set. Letting R be a rectangle and p_R be the chosen interior point for R, then it is important to realize that a report *A encloses* p_B holds not only when B is enclosed by A, but may also hold when B is not enclosed by A. However, in this latter case B and A have a point in common and hence intersect (see [BW] for more details). Finally in

Section 5 we combine the solutions to the two subproblems to solve the rectangle intersection problem.

3. LINE SEGMENT INTERSECTION

The first subproblem is:

Given n horizontal and vertical line segments, report all pairwise intersections.

A line sweep algorithm for this problem was found by Bentley and Ottmann [BO] and used in [BW]. In this section we first give an algorithm *LINE SEGMENT INTERSECTION* which makes use of a second algorithm *LINSECT*, then explain it and finally analyze its time- and space-requirements. *LINSECT* uses divide-and-conquer to accomplish its task.

For the sake of simplicity and clarity we initially assume all line segments to have distinct x- and y-coordinates.

Algorithm *LINE SEGMENT INTERSECTION*

Input: A set of horizontal line segments H and a set of vertical line segments V.

Output: The set of all intersecting pairs (h,v) where $h \in H$ and $v \in V$.

Step 1: Let $\overline{H}$ be the set of endpoints defined by H. Let S be $\overline{H} \cup V$ sorted by x-coordinate. (Observe that sorting is possible because all objects in $\overline{H} \cup V$ are characterized by a single x-coordinate. For simplicity we assume at present all x-coordinates to be distinct.)

Step 2: LINSECT (S, LEFT, RIGHT, VERT).

end of algorithm *LINE SEGMENT INTERSECTION*.

Algorithm *LINSECT(S , LEFT, RIGHT, VERT)*

Input: An x-ordered set of objects S where each object is either a point (a left or a right endpoint of a horizontal line segment) or a vertical line segment.

Output: Three sets *LEFT*, *RIGHT* and *VERT*. *LEFT* and *RIGHT* contain the y-projections of left and right endpoints in S, respectively, whose partner is not in S. *VERT* is a set of intervals. It contains the y-

projections of all vertical line segments in S.

Recursive Invariant: On exit *LINSECT* has reported all edge intersections within S, that is all pairs (h,v) where h is a horizontal line segment in S (represented by its left or right endpoint) and v is a vertical line segment in S.

Case 1: S consists of one element p.

a) $p=(p_x,p_y)$ is the left endpoint of a horizontal line segment:

$LEFT \leftarrow \{p_y\}$, $RIGHT \leftarrow \emptyset$, $VERT \leftarrow \emptyset$ and return .

b) $p=(p_x,p_y)$ is the right endpoint of a horizontal line segment:

$LEFT \leftarrow \emptyset$, $RIGHT \leftarrow \{p_y\}$, $VERT \leftarrow \emptyset$ and return .

c) $p=(p_x,p_{y1},p_{y2})$ is a vertical line segment:

$LEFT \leftarrow \emptyset$, $RIGHT \leftarrow \emptyset$, $VERT \leftarrow \{[p_{y1},p_{y2}]\}$ and return .

Case 2: S contains more than one element.

Divide: Choose an x-coordinate $\tilde{x}$ dividing S into two subsets S_1 and S_2 of approximately equal size.

Conquer: *LINSECT*$(S_1, LEFT_1, RIGHT_1, VERT_1)$;
LINSECT$(S_2, LEFT_2, RIGHT_2, VERT_2)$.

Merge: (Let $LR = LEFT_1 \cap RIGHT_2$)

$$output((LEFT_1 \setminus LR) \otimes VERT_2)) \quad (1)$$

$$output((RIGHT_2 \setminus LR) \otimes VERT_1)) \quad (2)$$

{ the operator $\otimes$ is defined below }

$$LEFT \leftarrow (LEFT_1 \setminus LR) \cup LEFT_2 \quad (3)$$

$$RIGHT \leftarrow RIGHT_1 \cup (RIGHT_2 \setminus LR) \tag{4}$$

$$VERT \leftarrow VERT_1 \cup VERT_2 \text{ and return .} \tag{5}$$

end of *LINSECT*.

The merge step needs some explanation. First we have to define the operator $\otimes$:

Let P be a set of points on the line and I a set of intervals on the line. Then

$$P \otimes I := \{(p, i) \mid p \in P, i \in I \text{ and } i \text{ contains } p\} .$$

In the merge step intersections between horizontal and vertical line segments are reported. Since horizontal line segments are represented by their endpoints we have to examine the interaction between points and vertical line segments.

Any set S_α to which *LINSECT* is applied has an associated rectangular area $A(S_\alpha)$ which is defined by the minimal and maximal x- and y-coordinates of objects in S_α. For a left or right endpoint $p \in S_\alpha$ let l_p be the horizontal line segment from which it is taken. Then the *partial segment* $PS(p)$ is defined by

$$PS(p) := l_p \cap A(S_\alpha) .$$

Example:

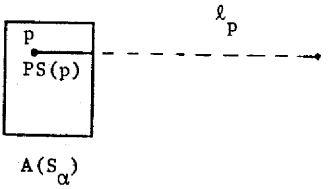

Figure 2

We know from the recursive invariant that after execution of *LINSECT* $(S_\alpha, \cdots)$ intersections between partial segments and vertical line segments in S_α have been reported. Hence in the merge step of *LINSECT*$(S, \cdots)$ we only have to find intersections between partial segments in S_1 and vertical line segments in S_2 and vice versa. In the merge step of *LINSECT*$(S, \cdots)$ the following things can happen to a partial segment $PS(p)$ (let us assume p is a left endpoint; the other case is symmetric):

a) p is in S_2

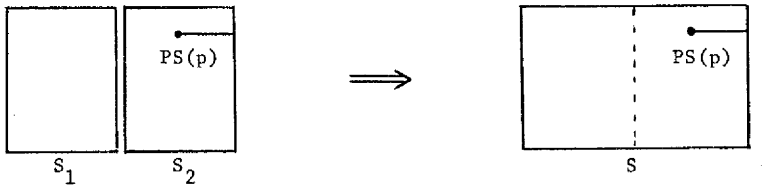

Figure 3

It *remains* as it is; no new intersections occur.

b) p is in S_1 and its partner is in S_2

Figure 4

It has *completed* meeting its partner partial segment; no new

intersections occur; both partial segments (or the complete segment) are removed from S.

c) p is in S_1 and its partner is not in S_2

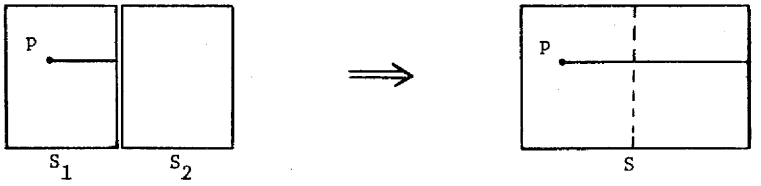

Figure 5

It is *extended* across S_2. The crucial point is that *all* vertical line segments in S_2 which contain p'sy coordinate intersect l_p regardless of their x-coordinate. In other words

> l_p intersects a vertical line segment $v_i \in S_2$ if and only if p's y-projection (a point) is contained in v_i's y-projection (an interval).

Hence, since Case (c) describes the only way intersections can occur, we have reduced the two-dimensional line segment intersection problem to the one-dimensional point enclosure problem. The task of finding all intersections between extended partial line segments from S_1 and vertical line segments from S_2 is exactly the computation of $(LEFT_1 \setminus LR) \otimes VERT_2$. Therefore the two output statements (1) and (2) report exactly the intersections that occur by combining S_1 and S_2. Hence the recursive invariant holds once more. Lines (3)-(5) construct the sets *LEFT*, *RIGHT* and *VERT* in the obvious manner; subtraction of *LR* yields the removal of complete (horizontal) segments from S.

Time:

Step 1: Constructing $\overline{H}$ takes linear time, sorting $O(n\log n)$ time, hence Step 1 takes $O(n\log n)$ time.

Step 2: Let $T(n)$ be the time complexity of applying *LINSECT* to a set S of cardinality n, that is *LINSECT(S, LEFT, RIGHT, VERT)*. Then

we have:

Case 1: $n=1$
All actions take only constant time, establishing

$$T(1) = O(1) \quad . \tag{6}$$

Case 2: $n>1$

Divide: The x-ordered set S can be given as an array-subrange. Then dividing takes only *constant time.*

Conquer: The recursive calls yield *two terms* $T(\frac{n}{2})$.

Merge: Finally we have to choose some representation of the sets *LEFT*, *RIGHT* and *VERT*. Surprisingly, simple y-ordered linked lists are sufficient to obtain an optimal solution. For *VERT*, each list item contains one interval $[y_b,y_t]$ and the list is ordered by the y_b-coordinates. Now all operations in the merge step can be realized by scanning some of the given lists in parallel. For instance, a first scan may operate on the lists (representing) $LEFT_1$,$RIGHT_1$,$LEFT_2$ and $RIGHT_2$ and construct lists *LEFT* and *RIGHT*. At the same time the elements of *LR* are removed from lists $LEFT_1$ and $RIGHT_2$. (This can be done by removing elements occurring in both $LEFT_1$ and $RIGHT_2$, because we assumed all y-coordinates to be distinct. If we permit multiple y-coordinates we use a different technique, see below. A second scan operates on lists $VERT_1$ and $VERT_2$ and the reduced lists representing $LEFT_1 \setminus LR$ and $RIGHT_2 \setminus LR$. During this scan *VERT* is constructed and the sets $(LEFT_1 \setminus LR) \otimes VERT_2$ and $(RIGHT_2 \setminus LR) \otimes VERT_1$ are computed and reported.

The computation of $P \otimes I$ where P and I are given as linked lists L_P and L_I is the only nontrivial (but not difficult) task. L_P and L_I are scanned in parallel. For each interval $i=[y_b,y_t]$ encountered in L_I this scan (called the *main scan*) pauses. From the current position in L_P (which corresponds to y_b) a *report scan* is started which reports a pair (p,i) for each point $p=y'$ encountered in L_P. The report scan terminates as soon as a y-coordinate $y'>y_t$ is encountered. Then the main scan is resumed.

Since the size of all lists is $O(n)$ the total time for scanning them (excluding report scans) is also $O(n)$. Hence the merge step contributes an $O(n)$ term if we count the time for reporting separately. Thus we have

$$T(n) \leq 2 \cdot T(n/2) + O(n) \qquad (7)$$

It is well known that the recurrence equations (6) and (7) have the solution

$$T(n) = O(n \log n)$$

(see for instance [AHU]). Obviously the report scans require time linear in the number of reported pairs (which correspond to intersections). Hence if k is the number of pairwise intersections between H and V, the total worst-case time required by the algorithm is $O(n \log n + k)$.

Space: The space required is $O(n)$ since each of the six lists used by the algorithm contains at most n elements.

Multiple x- or y-Coordinates

We now drop the restriction that the x- and y-coordinates of all line segments have to be pairwise distinct. The algorithm has to find all intersections between horizontal and vertical line segments. Finding intersections of type horizontal/horizontal or vertical/vertical is no problem (it amounts, for example, to sorting the horizontal line segments by y-coordinate and then for each subset with equal y-coordinate to find the intersections among a set of intervals) and can be done in separate step in $O(n \log n)$ time (the line sweep algorithm of Bentley and Ottmann [BO] reports only intersections of type horizontal/vertical and would also require a separate step to report the other types).

To accommodate multiple x- or y-coordinates we modify the algorithm slightly. We first deal with multiple x-coordinates. After the initial sorting of all objects in $H \cup V$ (points and vertical line segments) we are left with a set of *groups* $g_1, \ldots, g_r$ of objects, where each group g_i contains one or more objects with the same x-coordinate.

Example:

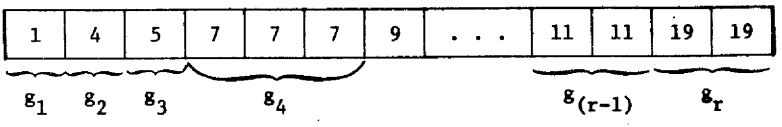

Figure 6

For each group g_i we construct the sets *LEFT*, *RIGHT* and *VERT* in advance and report the point enclosures (intersections) *LEFT*⊗*VERT* and *RIGHT*⊗*VERT*. Since these are one-dimensional problems the task only involves sorting the objects in the group by y-coordinate and can be done in $O(s \log s)$ time for a group with s elements. The total time taken for this step is $O(n \log n)$.

We now apply divide-and-conquer in the following way: Let $S = \bar{H} \cup V$ contain the elements $x_1, \ldots, x_m$ ($m \leq 2n$). We select the median object $x_{\lfloor (n+1)/2 \rfloor}$. It belongs to some group g_i. We divide S into three subsets S_1, S_c and S_2 where:

$$S_1 = \bigcup_{j=1}^{i-1} g_i \ , S_c = g_i \ , S_2 = \bigcup_{j=i+1}^{r} g_i \ .$$

We recursively apply the unmodified algorithm to S_1 and S_2, constructing the sets *LEFT*,*RIGHT* and *VERT* for each of them (remember that for S_c we constructed these sets beforehand).

After this S_1, S_c and S_2 are merged. This can either be done simultaneously or sequentially by first merging S_1 and S_c into S_{left}, then S_{left} and S_2 into S.

It is easy to see that this algorithm still has the same time complexity: By construction, each of the sets S_1 and S_2 contains less than $n/2$ elements. Merging S_1, S_c and S_2 takes $O(n)$ time. Hence the recurrence equations

$$T(1) = O(1)$$
$$T(n) \leq 2T(n/2) + O(n)$$

with solution $O(n \log n)$ holds again.

With multiple y-coordinates it becomes a problem to form the intersection of the set $LEFT_1$ and $RIGHT_2$. We said previously that *LEFT* and *RIGHT* are represented by y-ordered point lists. If a point list contains many elements with identical y-coordinates then it becomes a problem to identify a pair of points in $LEFT_1$ and $RIGHT_2$ stemming from the same horizontal line segment. However, there exists a simple way around this difficulty. Note that any set S to which *LINSECT* is applied contains all objects within a certain x-range. We pass this range as a parameter of *LINSECT*. Furthermore we add to the representation of a left (right) endpoint the x-coordinate of its right (left) partner endpoint. Merging, for instance, sets S_1 and S_c with associated x-intervals (a,b) and $[b,b]$, say, it is then possible to decide for any point in $LEFT_1$ and $RIGHT_c$ whether its partner endpoint is in the other set.

Hence it is once more possible to perform the subtraction of set *LR* ($=LEFT_1 \cap RIGHT_c$, in this case) in linear time.

In this way the time complexity of the algorithm is maintained for the general case which permits multiple x- or y-coordinates.

We summarize the results of this section in

Theorem 3.1 *For a set of n horizontal and vertical line segments in 2-space with k intersections the line segment intersection problem can be solved by divide-and-conquer in* $O(n\log n+k)$ *time and* $O(n)$ *space.*

4. FINDING POINT ENCLOSURES

The *point enclosure problem* is:

> *Given n objects each of which is a point or a rectangle in the plane, report for each rectangle all points that lie within it.*

Remember that the study of this problem is motivated by the fact that we can combine its solution with the solution of the line segment intersection problem to find all intersections among a set of rectangles. Again we use separational representation and divide-and-conquer and assume for simplicity at the moment that all objects have pairwise distinct coordinates.

The Idea

Separational representation applied to the given set reduces each rectangle to its left and right vertical edge and leaves the points unchanged. Hence we obtain a mixed set of points and vertical line segments which we sort by x-coordinate. To the resulting ordered set S the divide-and-conquer algorithm *PENC* is applied.

PENC splits a given set S into two subsets S_1 and S_2 and recursively computes y-ordered interval sets $LEFT_i$ (the y-projections of left rectangle edges), $RIGHT_i$ (the y-projections of right edges) and a set of y-coordinates $POINTS_i$ (y-projections of points) from S_i. The merge step which computes *LEFT*,*RIGHT* and *POINTS* from $LEFT_i$, $RIGHT_i$ and $POINTS_i$ $(i=1,2)$ is once more the crucial step. It makes use of the following observation:

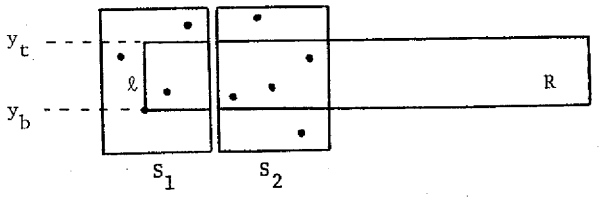

Figure 7

If a left vertical edge l in S_1 of a rectangle $R=(x_l,x_r,y_b,y_t)$ is not

matched by its partner edge in S_2, then the bottom edge and the top edge of R extend through all of S_2. This means all points in the y-ordered set $POINTS_2$ in the range $[y_b,y_t]$ are enclosed by R. $[y_b,y_t]$ is precisely the y-projection of l which is contained in $LEFT_i$.

To report all point enclosures within $S=S_1 \cup S_2$ we have to check whether or not the partner of each interval in $LEFT_1$ is in $RIGHT_2$. For each of the remaining intervals all enclosed points in $POINTS_2$ have to be reported (or rather the corresponding point-rectangle enclosure). Obviously the same has to be done for $RIGHT_2$ and $POINTS_1$. Using the notation $P \otimes I$ of Section 3 this is nothing other than computing and reporting $POINTS_2 \otimes (LEFT_1 \setminus LR)$ and $POINTS_1 \otimes (RIGHT_2 \setminus LR)$. Where $LR = LEFT_1 \cap RIGHT_2$. Hence the algorithm is a simple adaptation of *LINSECT*:

Algorithm *POINT ENCLOSURE*(P,R)

Input: A set of points P and a set of rectangles R.

Output: The set of all pairs (p,r) where $p \in P$ and $r \in R$ and p lies inside r.

Step 1: Construct the sets L and R of left and right edges of rectangles in R, respectively. Sort $L \cup R \cup P$ by x-coordinate, resulting in the ordered set S.

Step 2: *PENC*$(S,LEFT,RIGHT,POINTS)$

end of algorithm *POINT ENCLOSURE*.

Algorithm *PENC*$(S,LEFT,RIGHT,POINTS)$

Input: An x-ordered set of points and left and right vertical rectangle edges S.

Output: Three sets *LEFT*,*RIGHT* and *POINT*. *LEFT* and *RIGHT* contain the y-projections of all left and right rectangle edges from S, whose partner is not in S. *POINTS* contains the y-projection of $S \cap P$.

Recursive Invariant: On exit, *PENC*$(S, \cdots)$ has reported all point enclosures occurring within S, that is all pairs (p,r), where $p \in (P \cap S)$ and $r \in R$ and r is represented in S by its left or right vertical edge.

Case 1: S contains only a single object x.

Depending on the type of x (line segment or point) let *LEFT* or *RIGHT* contain a single y-interval or *POINTS* a single y-coordinate,

respectively, and let the other sets be $\varnothing$.

Case 2: S contains more than one object.

Divide: Choose an x-coordinate dividing S into two subsets S_1 and S_2 of approximately equal size.

Conquer: $PENC(S_1,LEFT_1,RIGHT_1,POINTS_1)$;
$PENC(S_2,LEFT_2,RIGHT_2,POINTS_2)$.

Merge: (Let $LR = LEFT_1 \cap RIGHT_2$).

$output(POINTS_2 \otimes (LEFT_1 \setminus LR))$
$output(POINTS_1 \otimes (RIGHT_2 \setminus LR))$
$LEFT \leftarrow (LEFT_1 \setminus LR) \cup LEFT_2$
$RIGHT \leftarrow RIGHT_1 \cup (RIGHT_2 \setminus LR)$
$POINTS \leftarrow POINTS_1 \cup POINTS_2$ and return

end of *PENC*.

Time and Space Complexity

We may choose the same linked-list representation for sets *LEFT*,*RIGHT* and *POINTS* as for *LINSECT*. In the merge step of *PENC* the same operations occur as in the merge step of *LINSECT*. Hence we know that they can be performed in linear time. The analysis is the same as for *LINSECT* and the algorithm may be adapted to multiple x- and y-coordinates by the same techniques. Hence we obtain

Theorem 4.1 *For a mixed set of points and rectangles in 2-space with cardinality n, the point enclosure problem can be solved by divide-and-conquer in $O(n \log n + k)$ time and $O(n)$ space where k is the number of point enclosures.*

5. COMBINING THE ALGORITHMS

We have already outlined in Section 2 how the algorithms of Sections 3 and 5 can be put together to yield a solution for the rectangle intersection problem. Because this has been described in detail in [BW] we add only a few final comments.

The preparatory steps in both algorithms can be replaced by a single scan through the set of rectangles, creating all input sets as needed. Instead of line segment names and point names we store, of course, the names of the corresponding rectangles (in an implementation the "name" is usually the address of some representation of the rectangle).

In an implementation some details have to be taken care of, such as avoiding multiple reporting (two rectangles whose edges intersect, have at least two different edge intersections), and not reporting the intersection of a rectangle with itself (two adjacent edges intersect, of course). However, these details do not affect the asymptotic complexity of the algorithm. They merely require careful programming.

The time and space requirements of the algorithm follow immediately from those of the component algorithms:

Let n be the number of rectangles, k the number of intersecting pairs of rectangles, k' the number of edge intersections and k'' the number of point enclosures. The algorithm *LINE SEGMENT INTERSECTION* requires $O(n\log n+k')$ time and *POINT ENCLOSURE* requires $O(n\log n+k'')$ time. Since $k'=O(k)$ and $k''=O(k)$ we have a total time bound of $O(n\log n+k)$. This time is known to be optimal, see [BW]. The space bound is $O(n)$ for both algorithms and hence also for their combination; this is also optimal. Thus we have proved:

Theorem 5.1 *For a set of n iso-oriented rectangles in 2-space the rectangle intersection problem can be solved by divide-and-conquer in $O(n\log n+k)$ time and $O(n)$ space where k is the number of pairwise rectangle intersections.*

6. CONCLUSIONS

We have shown the existence of a space- and time-optimal divide-and-conquer algorithm for the rectangle intersection problem. This can be viewed as a step towards investigating the relation between different algorithmic paradigms and their respective power.

We achieved this by the use of an idea which we wish to emphasize: the "separational representation" of planar objects. Applying divide-and-conquer to a planar object has the inherent difficulty that a dividing line may interest some of the objects. This yields immediately a partition into three sets *LEFT*,*RIGHT* and *MIDDLE* where *MIDDLE* interacts with both *LEFT* and *RIGHT*. "Separational representation" is applied to orthogonal objects in the following way: First, all horizonal parts are deleted. This leaves us with a number of fragments which are characterized by a single x-coordinate. A dividing line therefore splits the set of fragments into only two subsets *LEFT* and *RIGHT*, which can be "conquered" independently. In the merge step the horizontal parts are reconstructed though only on a conceptual level. Details can be found in Section 3. This idea might prove useful for the development of further divide-and-conquer algorithms.

We suggest the following open problems for further work:

(1) Generalize this algorithm to higher dimensions ($d>3$).

(2) Examine other problems based on sets of rectangles, like the closure problem [SSW], or the connectivity problem [EvLOW], for divide-and-conquer solutions.

(3) Characterize the class of problems for which line-sweep and divide-and-conquer are interchangeable.

(4) Is it possible to extend the applicability of divide-and-conquer (possibly using separational representation) to non-orthogonal planar objects, for instance arbitrarily oriented line segments?

REFERENCES

[AHU] Aho, A.V., Hopcroft, J.E., and Ullman, J.D.: *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[B] Baird, H.S., Fast Algorithms for LSI Artwork Analysis, *Journal of Design Automation and Fault-Tolerant Computing 2*, (1978), 179-209.

[BO] Bentley, J.L. and Ottmann, Th., Algorithms for Reporting and counting Geometric Intersections, *IEEE Transactions on Computers C-28*, (1979), 643-647.

[BW] Bentley, J.L. and Wood, D., An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles, *IEEE Transactions on Computers C-29*, (1980), 571-577.

[E] Edelsbrunner, H., A Time- and Space-Optimal Solution for the Planar All Intersecting Rectangles Problem, Technical University Graz, Institut für Informationsverarbeitung, Report 50, 1980.

[EvLOW] Edelsbrunner, H., van Leeuwen, J., Ottmann, Th., and Wood, D., Computing Connected Components of Orthogonal Geometric Objects. *RAIRO* (1983), to appear.

[EL] Eastman, C.M., and Lividini, J., Spatial Search, Institute of Physical Planning, Research Report 55, Carnegie-Mellon University, 1975.

[G1] Güting, R.H., Optimal Divide-and-Conquer to Compute Measure and Contour for a Set of Iso-Rectangles, Technical Report, Lehrstuhl Informatik VI, University of Dortmund, 1982.

[G2] Güting, R.H., Doctoral Disertation, Lehrstuhl Informatik VI, University of Dortmund, 1983.

[L1] Lauther, L., 4-Dimensional Binary Search Trees as a Means to Speed Up Associative Searches in Design Rule Verification of Integrated Circuits, *Journal of Design Automation and Fault-Tolerant Computing 2*, (1978), 241-247.

[L2] Lee, D.T., Algorithms for Rectangle Intersection Problems, unpublished manuscript, 1982.

[M] McCreight, E.M., Efficient Algorithms for Enumerating Intersecting Intervals and Rectangles, XEROX Palo Alto Research Center, Report CSL-80-9, 1980.

[MC] Mead, C., and Conway, L., *Introduction to VLSI-Systems*. Reading, MA: Addison-Wesley, 1980.

[SSW] Soisalon-Soininen, E., and Wood, D., An Optimal Algorithm to Compute the Closure of a Set of Iso-rectangles, *Journal of Algorithms* (1983), to appear.