Algorithms for Reporting and Counting Geometric Intersections

JON L. BENTLEY, MEMBER, IEEE, AND THOMAS A. OTTMANN

Abstract—**An interesting class of "geometric intersection problems" calls for dealing with the pairwise intersections among a set of N objects in the plane. These problems arise in many applications such as printed circuit design, architectural data bases, and computer graphics. Shamos and Hoey have described a number of algorithms for detecting whether any two objects in a planar set intersect. In this paper we extend their work by giving algorithms that *count* the number of such intersections and algorithms that *report* all such intersections.**

Index Terms—**Computational geometry, geometric intersection problems.**

I. Introduction

MANY fascinating aspects of "geometric intersection problems" have been brought to light in the recent study of Shamos and Hoey [7]. They investigated many different problems defined on sets of planar objects such as "do any two objects intersect?".

They pointed out that such problems arise in printed circuit design (do any conductors cross?), architectural data bases (are two items in one spot?), and operations research (linear programming can be reduced to an intersection problem). Shamos and Hoey have given many optimal algorithms in their paper both for *detecting* and for *forming* intersections of many different classes of objects.

In this paper we answer some of the open questions raised by Shamos and Hoey by solving problems of the form "*report* all intersecting pairs of objects" and "*how many* pairs intersect?". For example, we will give a fast algorithm for reporting all intersecting pairs among a set of N line segments in the plane. This problem arises in integrated circuit design, for crossovers must be placed at all such intersecting points (see, for example, [1] or [4]). In this application (and many others) it is critical that all such pairs be reported.

In Section II of this paper we will study an algorithm due to Shamos and Hoey for determining whether any pair of a set of line segments intersect, and then generalize their algorithm to report all intersecting pairs. In that section and the following we will assume that the reader is familiar with Shamos and Hoey [7]. In Section III we will see how to modify the algorithm of Section II to solve many other problems calling for reporting all intersecting pairs of planar objects. In Section IV we return to a special case of planar line segments, namely when all such segments are either horizontal or vertical. This case does arise in applications, and our algorithm for reporting all intersecting pairs of such segments is faster than for the general case (indeed, it is optimal). We also solve the problem of counting how many intersections there are in such a set. We give directions for further work and conclusions in Section V.

Manuscript received August 21, 1978; revised February 21, 1979. This research was supported in part by the U.S. Office of Naval Research under Contract N00014-76-C-0370.

J. L. Bentley is with the Departments of Computer Science and Mathematics, Carnegie-Mellon University, Pittsburgh, PA 15213.

T. A. Ottmann is with the University of Karlsruhe, Karlsruhe, Germany.

II. Intersection of Line Segments

In this section we will examine the problem of "given N line segments in the plane, report all intersecting pairs." We will investigate this problem by first describing an algorithm due to Shamos and Hoey [7] for detecting *whether* any of the segments intersect, and then we will modify that algorithm to *report* all intersecting pairs. In this paper we will not carefully describe certain important points such as the representation of line segments and algorithms for deciding if a point is above or below a given segment; we assume that the reader is familiar with Shamos and Hoey [7], where these details are discussed. Throughout this section we will make the assumptions that no segments in the set we are to process are vertical and that no three segments meet at any one point—to confront the details for handling these situations is cumbersome and not particularly illuminating.[1]

We will now briefly review Shamos and Hoey's algorithm for determining if any two of a set of N line segments in the plane intersect. The basic process of their algorithm "draws vertical lines" through the endpoints of segments in the set. They make the crucial observation that the positions at which the different segments intersect a given vertical line define a total ordering on those segments (the "above-below" ordering), and if the segment set is free of intersections then the relative ordering of any particular pair of segments will be the same at all vertical lines. (Note that if a line segment A is above segment B at one vertical line and B is above A at another, then they must have crossed, or intersected, somewhere between the two vertical lines.)

Once we have observed that there is a natural order

[1] The problem of vertical line segments can be solved by rotating the segment set a few degrees, ensuring that no line is vertical. This can be accomplished in linear time. The problem of many lines intersecting at one point is more subtle. In Algorithm 2.1 it is immaterial; that algorithm is concerned only with detecting *whether* there is an intersection. In Section IV we discuss how Algorithm 4.1 can be modified to deal with the problem. Algorithm 2.2 can also be so modified by noticing that if many segments intersect at one point, then they will all be adjacent in R when that point is scanned.

0018-9340/79/0900-0643$00.75

relation on sets of line segments with respect to any given vertical line it is easy to describe Shamos and Hoey's algorithm. The main loop of the algorithm "sweeps" a vertical line left-to-right through the set of segments, stopping at each endpoint (this is implemented algorithmically by sorting the $2N$ endpoints in an array and then sequentially scanning through it). At each point during this sweep we maintain the segments which intersect the vertical line defined by the current x-value, stored *in the order relation of the segments with respect to the vertical line.* As a left endpoint is encountered during the sweep we insert the segment into the ordering and as a right endpoint is encountered we delete it from the ordering. Whenever we insert a segment into the ordering we compare it against both of its "top" and "bottom" neighbors in the relation and when we delete a segment we compare the newly adjacent segments—if a given segment intersects any segment then it intersects one of those. Once such an intersection is found the algorithm reports it and halts; if no such intersection is found then none exists among the segments.

The correctness of this algorithm has been proved by Shamos and Hoey. They showed that if the order relation R among line segments is maintained as a balanced tree, then the running time of the algorithm is $0(N \lg N)$. We include a pseudo-Algol description of their procedure as Algorithm 2.1.

```
Q ← the set of all endpoints of segments, stored in order by
    x-values
R ← φ     // R is the order relation of segments currently
          examined
FOREACH endpoint p in Q (in ascending x-order) DO
    IF p is the left endpoint of segment s THEN insert s in R;
          check if s intersects the segments directly above
          or below it, and return that pair if it does
    ELSE // p is the right endpoint of s
          check if the segments directly above and below
          s intersect, if so return that pair; delete s from R.
```

Algorithm 2.1: Determine *if* N planar segments intersect.

We will now examine the more general problem of reporting all intersecting pairs, rather than just saying whether or not there is at least one such pair. This problem was posed by Shamos and Hoey, who asked if there exists an algorithm to do this in time $0(N \lg N + k)$, where k is the number of intersecting pairs. They showed this time complexity to be a lower bound on the problem. We cannot answer their question directly, but we can modify their algorithm to solve this problem in time $0(N \lg N + k \lg N)$.

The correctness of Shamos and Hoey's algorithm is due to the fact that if two segments intersect then at some time they must become adjacent in the vertical ordering. We will now show how this fact allows us to construct an algorithm for reporting all intersecting pairs of segments. We do this by "sweeping" a line through the point set (as before), always maintaining the correct vertical ordering in set R (as before), and then checking whenever modifying R to see if newly adjacent segments ever intersect (as before). Thus the algorithm we will present is substantially the same as Shamos and Hoey's original, but modified to maintain the correct total ordering on the segments even after an intersection is found. Note that if two segments are determined to intersect (somewhere to the right of the current scan position), then they will be in the correct order up to the intersecting point and at that point they should be "swapped" in the order. Having made this observation it is trivial to modify Shamos and Hoey's algorithm. They updated the order R at the "critical" times of entering and leaving the line segments. We will additionally update R at the "critical" time of segment intersection. Our modified version of Shamos and Hoey's algorithm is presented in pseudo-Algol as Algorithm 2.2.

```
Q ← the set of all endpoints of segments, stored in order by
    x-values;
R ← φ;   // The order relation among segments
FOREACH point p in Q (in ascending x-order) DO
    IF p is the left endpoint of segment s THEN
          insert s in R;
          check if s intersects the segments immediately
              above and below it and if it intersects segment
              t then insert the intersection point of s and t
              into Q (in x-order)
    ELSE IF p is the right endpoint of segment s THEN
          IF the intersection point of the pair of segments
              directly above and below s is not in Q THEN
              check them for intersection and if they meet
              then add their intersection point to Q (in
              x-order);
          delete s from R
    ELSE // p is the intersection of segment s and t
          report the pair as intersecting; swap the
          positions of s and t in R
              (notice that they were and still are
              adjacent);
          check the upper segment (say s) for intersec-
              tion with the segment above it, the lower
              (t) with segment below it, and add any
              intersection points to Q
```

Algorithm 2.2: Report *all* intersections among N planar segments.

The correctness of Algorithm 2.2 follows from the fact that the total ordering R of segments is correctly maintained at all times. The details of the proof are analogous to the argument in Theorem 2 of Shamos and Hoey [7]. To implement the algorithm efficiently we can store R as a balanced tree and Q as a heap. As before, we assume that there are k intersecting pairs. The number of times the FOREACH loop of Algorithm 2.2 is executed is exactly $2N + k$. Since the total order R can never contain more than N segments each operation on R within the FOREACH loop can be performed in $0(\lg N)$ time. The cost of each priority queue operation on Q is $0(\lg [2N + k]) = 0(\lg N)$ (since $k \leq N^2$) if Q is represented by a heap. We thus see that a cost of $0(\lg N)$ is incurred at each of the $0(N + k)$ iterations through the loop, so the total running time of the algorithm is $0(N \lg N + k \lg N)$. Note that if k is very close to N^2 then the running time of our algorithm is actually greater than the

$0(N^2)$ time of the "naive" algorithm that checks all $\binom{N}{2}$ pairs for intersection.

III. A General Algorithm

In their paper Shamos and Hoey showed how the algorithm they give for detecting intersection among line segments can be modified to detect intersection among sets of many different kinds of objects. In this section we will show how our algorithm for reporting all intersections among line segments can be modified to report all intersections among sets of many different kinds of objects. We will explore this facet of our algorithm by first mentioning general properties of objects sufficient for the correctness of our algorithm when applied to a set of such objects, and then use the general construction to solve a particular problem.

Algorithm 2.2 depended on three properties of line segments for its correctness. It can also be used to solve intersection problems on other objects as long as those objects display the following three properties.

Property P1: Any vertical line through the object intersects the object exactly once.

Property P2: For any pair of objects intersecting the same vertical line it is possible to determine algorithmically (at constant cost) which is above the other at that line.

Property P3: Given two objects it is possible to determine algorithmically if they intersect, and if so to compute their leftmost intersection point after some fixed vertical line.

Property P1 ensures that an order relation R will exist for any vertical line and Property P2 ensures that the relation can be computed. Property P3 is used by Algorithm 2.2 as it adds intersection points to Q; the leftmost intersection point after the current scan position is the one which should be added. So we see that if a class of objects C has Properties P1–P3, then we can report all intersections among a set of C's by modifying Algorithm 2.2 to read "C" whenever it reads "segment." The running time of the modified algorithm is still $0(N \lg N + k \lg N)$.

An example of a class of objects displaying the above three properties are circular arcs in the plane, restricted to exclude arcs which include a point with no derivative. (Note that an arc with such a point can be represented by two arcs without this property by "breaking" the arc into two at the place where the slope becomes vertical.) Such arcs can be described mechanically by giving a circle (center and radius), two x-values defining the endpoints of the arc, and one bit saying whether we are considering the upper or lower part of the circle within the specified x-slab. We have guaranteed that Property P1 is satisfied by excluding arcs that are both concave up and concave down. It is a trivial programming problem to design an algorithm showing that Properties P2 and P3 are both satisfied. This establishes the fact that $0(N \lg N + k \lg N)$ time is sufficient for reporting all k intersecting pairs among a set of N circular arcs in the plane.

One application of the above algorithm for determining arc intersection is the problem of "Euclidean fixed-radius near neighbors." In this problem we are given N points in the plane and then asked to report all pairs of points within some distance d of one another by the Euclidean metric. Notice that two points are within distance d of one another if and only if two circles, each centered at one of the points and both with radius $d/2$, intersect. Thus, we can find all near neighbors by considering each point in the set to be the center of a circle of radius $d/2$ and then reporting all intersecting circles. (Note that we will have to "break" each circle into its top and bottom halves, however.) This gives an $0(N \lg N + k \lg N)$ solution to the near neighbor problem. This same algorithm can also be used to report all intersections among a set of circles of varying radii.

IV. Intersections of Horizontal and Vertical Line Segments

In this section we consider the special case of planar line segment intersections in which each of the N given line segments is either vertical or horizontal. The problem of finding all intersecting pairs in a set of horizontal and vertical line segments arises in many applications. When designing an integrated circuit conductors are often restricted to horizontal and vertical lines; detecting all crossings of conductors calls for finding all intersecting pairs of vertical and horizontal line segments. Rectilinearly oriented squares and rectangles are built from vertical and horizontal line segments. Thus, an algorithm for finding intersecting pairs of vertical and horizontal line segments can be used to detect all (properly) intersecting pairs of squares and rectangles. Finally, we will show how the "L_∞ fixed radius near neighbors" problem (which asks for all pairs of N points in the plane within some fixed distance d of one another) can be solved efficiently by this algorithm, if the distance is measured by the L_∞ metric.

In order to simplify the presentation of the algorithm and to clarify the discussion we first restrict the input to the case where no line segments overlap. All x-values of vertical lines and left and right endpoints of horizontal lines are pairwise distinct. We will later mention how to handle these cases. In the problem of interest we are given N vertical or horizontal line segments in the plane. Each vertical line segment A is specified by its x-coordinate $x(A)$ and the y-values of its lower and upper endpoints $\text{bot}(A)$ and $\text{top}(A)$. Each horizontal line segment is similarly specified by its y-coordinate $y(B)$ and by the x-values of its left and right endpoints $\text{left}(B)$ and $\text{right}(B)$. We will let M be the set of x-coordinates

$$M = \{x(A) \mid A \text{ vertical}\} \cup \{\text{left}\,(B) \mid B \text{ horizontal}\} \cup \{\text{right}\,(B) \mid B \text{ horizontal}\}.$$

Note that M has at most $2N$ elements.

The main loop of our algorithm sweeps a vertical line from left to right through the set M. We use a data structure R to store horizontal line segments currently intersecting the vertical lines ordered by their y-coordinates. Initially R is empty. Whenever a left (respectively right) endpoint of a horizontal line segment S is scanned S is inserted into (respectively deleted from) the structure R. When a vertical segment is encountered during the sweep we check for

intersection with horizontal line segments in R. We describe this algorithm in pseudo-Algol as Algorithm 4.1.

$Q \leftarrow$ the set M in ascending x-order (stored in such a way that for each p in Q we can recognize to which line segment p belongs and whether p is the x-value of a vertical line segment or the left or right endpoint of a horizontal segment).
$R \leftarrow \phi$ // The order relation among horizontal line segments (ordered by y-values)
FOREACH p in Q (in ascending x-order) DO
  IF p is the x-value of the left endpoint of a horizontal line segment S THEN insert S in R
  ELSE IF p is the x-value of the right endpoint of a horizontal line segment S THEN delete S from R
  ELSE // p is x-value of a vertical line segment S
    determine A = successor (bot (S), R), the least y-value greater than or equal to bot (S) in R;
    determine B = predecessor (top (S), R), the greatest y-value less than or equal to top (S);
    FOREACH horizontal line segment T occurring in R between A and B do RETURN (S, T) as an intersecting pair.

Algorithm 4.1: Report *all* intersecting pairs of N planar horizontal and vertical line segments.

It is easy to see that Algorithm 4.1 correctly finds all intersecting pairs of line segments: Whenever a vertical line segment S is considered R contains exactly the horizontal line segments crossing the vertical line $x = x(S) = p$ (ordered by y-values). The algorithm then reports all pairs (S, T) where T crosses S. The data structure R can be implemented as a balanced tree. Hence, the time to perform the operations insert, delete, successor and predecessor is $0(\log N)$. Reporting the intersecting pairs can be done in time proportional to their number if we use an appropriate implementation of balanced trees. Brother (leaf-search) trees of Ottmann and Six [6] (see van Leeuwen [5] for an English exposition), for example, can be used for this task where the leaves of the tree are kept in a doubly linked list. The structure Q can be implemented as a sorted linear list. Sorting the elements of M and storing them in increasing order takes time $0(N \log N)$. Hence, the total performance time of the algorithm is $0(N \log N + k)$ where k denotes the number of intersecting pairs.

To simplify the presentation of Algorithm 4.1 we assumed that no pair of line segments share endpoints or lie on the same line, but these assumptions can be removed by increasing the "bookkeeping" performed by the algorithm. To handle the case that endpoints of segments might intersect other segments we must be careful to process "multiple events" at a vertical line during the sweep in the order "insert new horizontal endpoints," "check vertical segments," "delete old horizontal endpoints." We must also report any pair of vertical (likewise horizontal) segments that meet—this can be accomplished before the main body of Algorithm 4.1 is ever invoked. We will sketch the procedure for detecting overlap among pairs of horizontal lines; the case of vertical lines is exactly the same. We first sort all horizontal segments by y-value and then consider only "clusters" of segments sharing the same y-value. Within each cluster we sort the endpoints by x-value, and a scan through the resulting list can report all overlapping pairs in time proportional to their number. Neither of the special cases we have just sketched alters the $0(N \log N + k)$ running time of Algorithm 4.1.

We now briefly describe how to modify Algorithm 4.1 for counting the *number* of intersecting pairs of N horizontal or vertical line segments (without reporting them). We associate a counter with each element A in R which indicates the number of elements preceding A in R. Whenever the sweeping vertical line encounters a vertical line segment S we determine A = successor (bot (S), R) and B = predecessor (top(S), R); the *number* of horizontal line segments intersected by S is the difference between the counter of B and the counter of A. This number is added to the total number of intersecting pairs found so far. In a balanced tree counters can be updated after an insertion or deletion in time $0(\log N)$.[2] The modified algorithm will therefore report the total number of intersecting pairs after $0(N \log N)$ steps, which is optimal.

Finally, we apply Algorithm 4.1 to solve the "L_∞ fixed radius near neighbors" problem. In this problem we are given N points in the plane and asked for all pairs of points within some distance d of one another by the L_∞ metric (i.e., the maximum coordinate metric). The crucial observation is that two points are within distance d of one another if and only if two squares of side length d and with sides parallel to the coordinate axes, each centered at one of the points, intersect. Hence, for finding all pairs of near neighbors we surround each point with a square centered at that point (with side length d and sides parallel to the coordinate axes). Then we use Algorithm 4.1 for reporting all pairs of intersecting squares. This gives an $0(N \log N + k)$ solution to the "L_∞ fixed radius near neighbors" problem. The same result was obtained in a totally different manner by Bentley *et al.* [2]. This algorithm can also be used to find all edge intersections among any set of rectilinearly oriented rectangles.

V. Conclusions

We will now briefly summarize the research described in this paper. In Section II we showed how Shamos and Hoey's algorithm for detecting *whether* any two of N planar line segments intersect can be modified to *report* all such intersecting pairs; the running time of the resulting algorithm was $0(N \lg N + k \lg N)$. In Section III we modified the algorithm of Section II so it can report all intersections in planar sets composed of more complicated objects than line segments; the running time of the algorithm was not changed. We then examined a special case of line segments (when each is either horizontal or vertical) in Section IV and

[2] This involves storing in each node P a RANK field telling P's rank in its subtree (that is, one plus the number of nodes in P's left subtree). For the details of this scheme see Knuth's discussion in [3, pp. 463 ff.].

showed how all intersecting pairs in such a set could be *reported* in $0(N \lg N + k)$ time and *counted* in $0(N \lg N)$ time. Both of those performances are optimal. The existence of these algorithms partially answers a number of questions posed by Shamos and Hoey. These algorithms are particularly interesting because they are among the first geometric algorithms with complexity described not only as a function of the problem *input* size, but jointly as a function of problem *input* and *output* sizes.

A number of geometric intersection problems remain unsolved. The most outstanding open problem is Shamos and Hoey's question of whether all k intersections among N line segments can be reported in $0(N \lg N + k)$ time. (Perhaps one reason that our algorithm fails to meet that bound is that it reports the intersection points sorted by x-value; this might in itself require $0(k \lg k)$ time.) Another interesting open problem deals with finding intersections in a set of rectangles; this problem arises in integrated circuit design (see [4]). Although Algorithm 4.1 can be used to find all pairs that have intersecting edges, it will not report as intersecting one rectangle that lies totally within another. It is not clear whether the methods described in this paper can be used to solve this "rectangle inclusion" problem. The reader interested in further open problems may consult the list given by Shamos and Hoey [7]. Perhaps some of the methods we have used in this paper can be applied to those problems.

Acknowledgment

The authors would like to thank Dr. U. Lauther for pointing out a bug in the original presentation of Algorithm 2.2.

References

[1] H. S. Baird, "Fast algorithms for LSI artwork analysis," *J. Design Automat. Fault-Tolerant Comput.*, vol. 2, pp. 179–209, May 1978.

[2] J. L. Bentley, D. F. Stanat, and E. H. Williams, Jr., "The complexity of finding fixed-radius near neighbors," *Inform. Proc. Lett.*, vol. 6, pp. 209–212, Dec. 1977.

[3] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

[4] U. Lauther, "Four-dimensional binary search trees as a means to speed up associative searches in design rule verification of integrated circuits," *J. Des. Autom. and Fault-Tolerant Comput.*, vol. 2, pp. 241–247, July 1978.

[5] J. van Leeuwen, "The complexity of data organization," in *Foundations of Computer Science II*, K. E. Apt, Ed., Mathematisch Centrum Amsterdam, pp. 37–147.

[6] T. Ottmann and H. W. Six, "Eine neue Klass von ausgeglichenen Binärbäumen," *Angewandte Informatik*, vol. 9, pp. 395–400, 1976.

[7] M. I. Shamos and D. J. Hoey, "Geometric intersection problems," in *Proc. 17th Annu. Conf. Foundations of Computer Science*, pp. 208–215, Oct. 1976.

Jon L. Bentley (M'70) was born in Long Beach, CA, on February 20, 1953. He received the B.S. degree in mathematical sciences from Stanford University, Stanford, CA, in 1974, and the M.S. and Ph.D. degrees in computer science from the University of North Carolina at Chapel Hill in 1976.

He worked as a Research Intern at the Xerox Palo Alto Research Center from 1973 to 1974. During the summer of 1975 he was a Visiting Scholar at the Stanford Linear Accelerator Center. From 1975 to 1976 he was a National Science Foundation Graduate Fellow. He was awarded the Second Prize in the 1974 ACM Student Paper Competition. In 1977 he joined the faculty of Carnegie-Mellon University, Pittsburgh, PA, as an Assistant Professor of Computer Science and Mathematics. His primary research interests include the design and analysis of computer algorithms (especially for geometrical and statistical problems) and the mathematical foundations of computation. Other research areas in which he has worked include software engineering tools and novel computer architectures.

Dr. Bentley is a member of the Association for Computing Machinery and Sigma Xi.

Thomas A. Ottmann was born in Germany in 1943. He received the M.S. degree in mathematical and physical sciences in 1969 and the Ph.D. degree in mathematical logic in 1971 both from the University of Münster, Münster, Germany, and the venia legendi for computer science from the University of Karlsruhe, Karlsruhe, Germany, in 1975.

He worked as a Research Assistant at the Institut für Mathematische Logik at the University of Münster. In 1973 he joined the Institut für Angewandte Informatik of the University of Karlsruhe where he is currently Associate Professor for Computer Science. His research interests include mathematical foundations of computer science and the theory and applications of data structures.