POLYGONAL INTERSECTION SEARCHING

H. EDELSBRUNNER and H.A. MAURER

Institut für Informationsverarbeitung, Technical University of Graz, Steyrergasse 17, 8010 Graz, Austria

D.G. KIRKPATRICK

Department of Computer Science, University of British Columbia, 2075 Wesbrook Mall, Vancouver BC V6T 1W5, Canada

Received 30 December 1980; revised version received 12 November 1981

Geometric searching, multi-dimensional searching, polygon intersection, triangular range searching, geometric transforms, concrete complexity

1. Introduction

Recent developments concerning intersection searching involving special kinds of geometric objects have motivated us to investigate intersection searching with general polygons in the plane. The papers of Edelsbrunner [10] and Edelsbrunner and Maurer [12] can be seen as concluding a long period in which various special cases of essentially one problem were examined, namely, the intersection searching problem for orthogonal objects. A geometric object is called *orthogonal,* if every one of its edges are parallel to one of the coordinate axes. Two objects are said to *intersect* if they have at least one point in common. Special cases of intersection searching for orthogonal objects considered prior to [10] and [12] include problems such as the range searching problem (e.g. Bentley and Maurer [2], Bentley [1], Willard [21], and others), the line intersection searching problem with vertical and horizontal line segments (e.g. Vaishnavi and Wood [20]), and the inverse range searching problem (e.g. Vaishnavi [19]). The paper of Bentley and Wood [4] initiated a series of papers concerning orthogonal rectangle intersection, the most important of which are due to Six and Wood [17,18]. Edelsbrunner [8,9] gave the ingredients for a uniform treatment of orthogonal intersection searching.

The successful treatment of general orthogonal objects in [10] and [12] suggested a general treatment of geometric intersections involving simple polygons with a bounded number of edges, leading to what we call the *polygon intersection searching problem*: Given a set of simple plane polygons with a bounded number of edges and a query polygon of the same type, determine all polygons intersecting the query object.

Fig. 1 illustrates this problem be depicting eight simple polygons, each of which consists of at most five edges. (Notice that degenerate polygons such as points are also allowed.) The query polygon consists of four edges and intersects four of the eight objects.

As will be explained in more detail in the next section, the polygon intersection searching problem can be solved by reducing it to three simpler searching problems which are of independent interest. The first of these involves a set of polygons (again with a bounded number of edges as will be assumed throughout the paper) and asks for the identification of all polygons containing a specified query point. The second asks for all line segments of a given set which intersect a given query line segment. The third, finally, requires the determination of all points of a set which lie in a given query polygon. These problems are generalizations of their counterparts in orthogonal object searching, namely of the inverse range searching problem, of the rectilinear line segment intersection searching problem, and of the range searching problem.

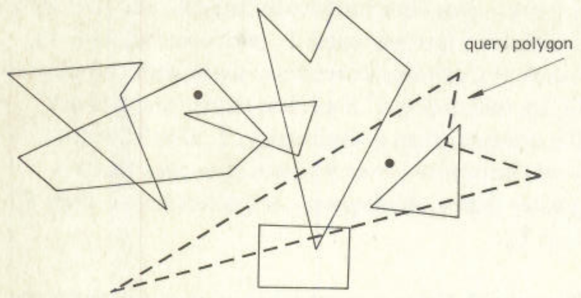

Fig. 1. Polygon intersection searching.

The main result is the following: Let S denote a set of n simple polygons in the plane such that each polygon is bounded by at most a constant number of edges. A polygon intersection query asks for the t polygons in S that intersect the query polygon which is again a simple polygon in the plane bounded by at most a constant number of edges. With suitable preprocessing, such a polygon intersection query can be answered in time O(log n + t).

This result is obtained by transforming several subproblems by means of a geometric transform to planar point location searching which can be defined as follows: Suppose we have a subdivision of the plane defined by n straight-line edges which do not intersect except possibly at their endpoints. A point location query asks for the region of the subdivision a given query point is in. This problem has been thoroughly examined in the past. Dobkin and Lipton [7] were the first to obtain a solution that answers a query in time O(log n). However, their method is based on a rather space consuming data structure. Preparata [15] refined their method and reduced the space requirements to O(n log n) without worsening the query time. Finally, Kirkpatrick [13] presented a solution optimal in time and space, i.e. the data structure requires O(n) space and allows the determination of the region a given point is in in O(log n) time. Only a few problems remain open in this area and are those mainly involving arbitrary regions, as considered by Edelsbrunner and Maurer [11].

In the next section, the main theorem is established. Before doing so, the three subproblems are examined to which the main result can be reduced. Finally, the contribution of this paper is discussed and some directions for further research are pointed out.

2. Polygon intersection searching

We are interested in finding a fast solution for the polygon intersection searching problem. We will show that this problem (and hence many simpler problems) can be solved in a time proportional to the logarithm of the number of polygons involved plus the time to write the answer. Unfortunately, our method will be rather costly in space and preprocessing time. The polygons considered are assumed to be simple (i.e., the edges of one polygon do not intersect except possibly at their endpoints, and each edge intersects its two 'neighbor' edges only) and bounded by at most a constant number, say K, edges.

Suppose a set of points, lines, and polygons is given. We determine the objects which intersect a query polygon by solving three subproblems:

(1) given a point, determine all the polygons it is in,

(2) given a line segment, determine all the line segments it intersects,

(3) given a polygon, determine all the points it contains.

We are done if we can show that each one of these subproblems can be solved in O(log n + t) time, while the methods report the same polygon intersection no more than a constant number of times.

The following three theorems will be devoted to the above three subproblems. They are obtained by transforming the respective problems to planar point location searching.

We are especially interested in solutions that allow a point location query to be answered in O(log n) time.

Since our structures will require much space anyway, any of the methods due to Dobkin and Lipton [7], Preparata [15], or Kirkpatrick [13] will serve in the subsequent discussion.

Theorem 1. Given a set of n simple polygons in the plane with a bounded number of edges, there is a data structure such that the t polygons that contain a given point can be located and reported in O(log n + t) time.

Proof. We will reduce this problem to planar point location searching.

The n polygons together consist of at most Kn edges. The collection of edges establishes a straight-line subdivision of the plane. Each region of this subdivision is covered by some of the original polygons. Our intention is to precompute for each region of the subdivision the set of covering polygons. To determine all the polygons a point is in, it is sufficient to determine the region it lies in. The polygons covering this region will be precisely the polygons that enclose the point and must be reported.

Observe that the subdivision consists of $O(n + s)$ vertices, where s denotes the number of intersections occurring among the various edges. A single edge can intersect at most $K(n - 1)$ edges, hence, s is bounded above by $K^2 n(n - 1)$. This implies that the subdivision comprises no more than $O(n + s)$ regions, each of which has to be assigned the list of all covering polygons.

The subdivision can be set up in $O(n \log n + s \log n)$ time (see Nievergelt and Preparata [14]), and all the lists of covering polygons can be assigned in additional $O(n + sn)$ time, using $O(n + sn)$ space. Any of the structures developed by Dobkin, Lipton [7], Preparata [15], and Kirkpatrick [13] will serve to establish the final structure which allows a query to be answered in $O(\log n + t)$ time.

In order to solve the subproblems (2) and (3), we need a geometric transform T to reduce them to planar point location searching as well. One such transform, T, proposed in Brown [5] transforms a point p with Cartesian coordinates (a, b) into the line T_p of points (x, y) satisfying $y = ax + b$. Conversely, a line ℓ described by $y = kx + d$ is transformed into the point T_ℓ with coordinates $(-k, d)$.

Lemma 1. Let ℓ be a non-vertical line in the plane and let ℓ^+ denote the halfplane above ℓ, and ℓ^- the halfplane below ℓ. If a point p with coordinates (a, b) is in ℓ^+ then T_ℓ is in T_p^-, and if p is in ℓ then T_ℓ is in T_p^+.

This lemma can be established by straightforward calculation. We leave the details to the reader. Since lines parallel to the y axis must be treated separately, we mention a possibility to deal with such lines. First, we assume that the left halfplane of such a vertical line v lies 'above' it and that the right halfplane lies 'below' it. Let v be specified by the equation $x = c$. Due to our assumption, v resembles the line satisfying $y = mx - cm$, for a sufficiently large m, and thus we transform v into the point T_v with coordinates $(-m, -cm)$. If p denotes an arbitrary point with coordinates (a, b), then p is in v^+ if a is less than c, and p is in v^- if a is greater than c. Conversely, T_v is in T_p^+ if $(-m, -cm)$ lies above $y = ax + b$, which implies that a is greater than c. Analogously, if a is less than c, then T_v is in T_p^-.

Theorem 2. Given a set of n line segments in the plane, there is a data structure such that those t line segments that intersect a given query line segment can be located and reported in $O(\log n + t)$ time.

Proof. Let s denote the number of intersections among the n line segments. First, the n segments are cut into pieces such that no pair of these pieces intersects except possibly at their endpoints. As can be readily seen, s is bounded above by $n(n - 1)$. The cutting process results in a set of $n + 2s$ new line segments with $2n + 4s$ endpoints. Transform the endpoints into lines by means of the transform T. This gives a planar subdivision comprising $O((n + s)^2)$ regions.

Let ℓ denote the query line segment and let ℓ be part of the line L specified by the equation $y = kx + d$. All original line segments whose endpoints lie in different halfplanes L^+ and L^- intersect L and thus potentially intersect ℓ. In order to obtain these line segments, we transform L into $T_L = (-k, d)$ and look for the region of the subdivision T_L is in. This region uniquely determines the original segments intersecting L.

Since we consider sets of non-intersecting line segments, the segments intersecting L are totally ordered w.r.t. their intersections with L. Note that Lemma 1 implies that this total order does not change for any M with T_M in the same region as T_L. Thus, the totally ordered set of line segments may be preprocessed for each region and those t of them which intersect ℓ can be determined by binary search. The time needed to answer a query is therefore $O(\log n + t)$, using a structure requiring $O(n(n + s)^2)$ space and time to construct.

Similarly, the third subproblem is solved with the techniques of geometric transformation and planar point location searching. However, before discussing the actual problem of determining the points lying in a specified polygon, we focus on a more elementary problem.

Lemma 2. Given n points and a halfplane, there is a data structure that permits us to locate and report those t points that lie in the halfplane in time $O(\log n + t)$.

Proof. As was previously carried out, the given points are transformed by means of T into lines to establish a planar subdivision. The line ℓ bounding the given halfplane is transformed into the point T_ℓ and the region to which T_ℓ belongs is determined. The lines above the region correspond to original points in ℓ^+, and the lines below the region correspond to points in ℓ^-. If we associate with each region two lists containing the points corresponding to lines above and below the region, respectively, $O(\log n)$ time for locating the desired points suffices. The data structure requires $O(n^3)$ space and time to construct, since the subdivision comprises $O(n^2)$ regions, to each of which two linear lists (with at most n points each) are associated.

Theorem 3. Given n points in the plane, there is a data structure that permits us to locate and report those t points that lie in a specified simple polygon that is bounded by at most a constant number of edges in time $O(\log n + t)$.

Proof. Note that the query polygon comprises at most K edges, and hence, can be decomposed in O(1) time into K – 2 disjoint triangles. The problem is solved by determining for each constituent triangle the points that lie in it.

Once more, the n points are transformed by means of T into n lines to establish a subdivision of the plane consisting of $r = O(n^2)$ regions R_i, with i ranging from 1 to r. A query triangle is interpreted as the intersection of three halfplanes. A query asking for those points that lie in a specified triangle is called a *triangular range query*. Such a query can thus be accomplished by, first, transforming the three lines bordering the halfplanes into three points, and second, searching for the regions of the subdivision the three points lie in. The triple of regions obtained uniquely determines the set of original points in the triangle.

In the preprocessing phase, a three-dimensional array A(i, j, k) (with i, j, and k ranging from 1 to r) is set up. A triple of regions R_i, R_j, R_k corresponds to the element A(i, j, k) of the array. The $r^3 = O(n^6)$ elements of the array that correspond to triples of regions each contain the answer. Since each answer may comprise up to n of the original points, the above structure requires $O(n^7)$ space and time to construct. A query can be answered in time $O(\log n + t)$, where t denotes the number of points inside the query triangle.

We now demonstrate how the general problem can be treated by solving several instances of the three subproblems, thus proving our Main Theorem.

Main Theorem. Suppose n simple polygons in the plane, bounded by at most a constant number of edges, are given. With suitable preprocessing, each polygon intersection query can be answered in time $O(\log n + t)$, where t denotes the number of polygons that intersect the query polygon which is of the same type as the other polygons.

Proof. Let p be any point in the interior of the query polygon. The solution to subproblem (1) can be used to report the polygons that enclose this point, including all polygons which enclose the query polygon. The solution to subproblem (2) can be used to report all occurring edge intersections. This enables us to report those polygons which intersect the query polygon by at least one edge. A point in the interior of each polygon is chosen and checked to see if it is in the interior of the query polygon. If it is, the associated polygon potentially lies entirely in the query polygon. Hence, the solution to subproblem (3) is used to detect those polygons that lie entirely within the query polygon.

A polygon that intersects the query polygon will be detected potentially once with subproblem (1), no more than K^2 times with subproblem (2) and potentially once with subproblem (3). Hence, the reduction method reports one and the same polygon intersection at most $K^2 + 2$ times.

The reason for restricting our polygons to comprise at most a constant number of edges has become obvious in the proof of the above Main Theorem: One and the same intersection could be reported too often, otherwise. Additionally, we would be forced to measure the required resources in terms of the total number of edges (which is proportional to the number of polygons in our cases) instead of the number of polygons.

Let us now demonstrate how multiple answers (i.e. one and the same intersecting polygon is reported more than once but at most a constant number of times) can be avoided without affecting the stated complexities. A bit-vector with a bit for each stored polygon is used. We assume that prior to a polygon query each bit equals 0. Now instead of reporting each polygon immediately after having detected it, we just mark the intersection by setting the appropriate bit to 1. After having marked the occurring intersections, the indicated polygons are reported and the associated bits are reset to 0.

A challenging task is the reduction of space requirements with or without increasing the time needed to answer a query. Note that, first of all, space reductions for the triangular range searching problem would improve the space requirements for polygon intersection searching. However, the problems considered in Theorem 1 and 2 are of independent interest and it seems worthwhile to think about space reductions in these cases as well.

Note that our methods, as presented, fail to determine the number of polygons intersecting a query polygon. This rather unusual effect occurs due to the fact that one intersection is in general reported more than once (but at most a constant number of times). Although the method described in the last section cannot determine the number of intersections, it can be modified to do so. The main problem has been reduced to three subproblems which solve the main problem via independent processing. By performing the subproblems dependently (as is done above for the triangular range searching problem), we can determine the number of intersections in O(log n) time. However, the space requirements are even worse than for the structures that report intersections.

3. Discussion

We have shown that O(log n + t) time suffices to answer a polygon intersection query involving n simple (and potentially non-convex) polygons with a bounded number of edges, where t denotes the number of polygons intersecting the query polygon, provided the polygons are stored in a proper data structure. The result has been obtained by reducing the problem to three simpler searching problems which are of independent interest. The three subproblems are solved by means of a geometric transform introduced by Brown [5] and by solutions of the point location searching problem.

Perhaps the most urgent question is to decrease the prohibitive space requirements of our data structures. This may also be of interest for the design of dynamic structures supporting polygonal intersection searching.

Finally, we would like to draw attention to an interesting phenomenon occurring in 'non-orthogonal' geometric intersection searching. Shamos and Hoey [16] were the first to consider the problem of determining whether a set of line segments in the plane is free of intersections or not. Bentley and Ottmann [3] generalized their algorithm to report all intersecting pairs of such a set in O(n log n + s log n) time and O(n + s) space, where s denotes the number of intersections occurring among the line segments. Recently, Brown [6] improved this algorithm to use only O(n) space. Anyway, in the formula for the time-complexity the number of intersections occurs implicitly with the number of line segments. In other words, the algorithms cannot apparently manage in time dependent only on the input size plus time linear in the output size. The same situation appeared in Nievergelt and Preparata [14] who succeeded in cleaning up the case of the intersection of two convex subdivisions of the plane, but were not able to do so for the general case.

The same phenomenon occurs with the searching problems considered in this paper. The time- and space-bounds grow with the number of intersections among the objects of the set. It seems worthwhile to investigate this phenomenon in 'non-orthogonal' intersection more closely.

References

[1] J.L. Bentley, Multidimensional divide-and-conquer, Comm. ACM 23 (1980) 214–229.

[2] J.L. Bentley and H.A. Maurer, Efficient worst-case data structures for range searching, Acta Inform. 13 (1980) 155–168.

[3] J.L. Bentley and Th. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. 28 (1979) 643–647.

[4] J.L. Bentley and D. Wood, An optimal worst-case algorithm for reporting intersections on rectangles, IEEE Trans. Comput. 29 (1980) 571–577.

[5] K.Q. Brown, Geometric transforms for fast geometric algorithms, Report CMU-CS-80-10, Carnegie-Mellon University, Department of Computer Science (1980).

[6] K.Q. Brown, Comments on 'Algorithms for reporting and counting geometric intersections', IEEE Trans. Comput. 30 (1981) 147–148.

[7] D.P. Dobkin and R.J. Lipton, Multidimensional searching problems, SIAM J. Comput. 5 (1976) 181–186.

[8] H. Edelsbrunner, Dynamic rectangle intersection searching, Report F47, Technical University of Graz, Institut für Informationsverarbeitung (1980).

[9] H. Edelsbrunner, A time- and space-optimal solution for the planar all intersecting rectangle problem, Report 50, Technical University of Graz, Institut für Informationsverarbeitung (1980).

[10] H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, Report F59, Technical University of Graz, Institut für Informationsverarbeitung (1980).

[11] H. Edelsbrunner and H.A. Maurer, A space-optimal solution of general region location, Theoret. Comput. Sci. 16 (1981) 329–336.

[12] H. Edelsbrunner and H.A. Maurer, On the intersection of orthogonal objects, Report F60, Technical University of Graz, Institut für Informationsverarbeitung (1980).

[13] D.G. Kirkpatrick, Optimal search in planar subdivisions, manuscript, University of British Columbia, Department of Computer Science (1979).

[14] J. Nievergelt and F.P. Preparata, Plane sweep algorithms for intersecting geometric figures, Report R-863, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory (1979).

[15] F.P. Preparata, A new approach to planar point location, Report R-829, University of Illinois at Urbana-Champaign, Coordinated Science Laboratory (1978).

[16] M.I. Shamos and D. Hoey, Geometric intersection problems, Proc. 17th Annual Symposium on Foundations of Computer Science (1976) pp. 208–215.

[17] H.-W. Six and D. Wood, The rectangle intersection problem revisited, BIT 20 (1980) 426–433.

[18] H.-W. Six and D. Wood, Counting and reporting intersections of d-ranges, Report 80-CS-6, McMaster University, Unit for Computer Science (1980).

[19] V.K. Vaishnavi, Computing point enclosures, manuscript, Concordia University, Computer Science Department (1980).

[20] V.K. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees and dynamization, Report 80-CS-8, McMaster University, Unit for Computer Science (1980).

[21] D.E. Willard, New data structures for orthogonal queries, Report TR-22-78, Harvard University, Aiken Computer Laboratory (1978).