Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques

L. Guibas, M.H. Overmars, M. Sharir

RUU-CS-88-26
August 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques

L. Guibas, M.H. Overmars, M. Sharir

Technical Report RUU-CS-88-26
August 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

Intersecting Line Segments, Ray Shooting, and Other Applications of Geometric Partitioning Techniques

Leonidas Guibas[1,2], *Mark Overmars*[3] *and Micha Sharir*[4,5]

[1] Computer Science Department, Stanford University
[2] DEC Systems Research Center
[3] Department of Computer Science, University of Utrecht
[4] Courant Institute of Mathematical Sciences, New York University
[5] School of Mathematical Sciences, Tel Aviv University

ABSTRACT

We present a variety of applications of certain techniques, based on partition trees, that were originally developed for range searching problems. Our results are obtained by enhancing and extending these techniques, and include: (i) An $O(n^{4/3+\delta} + k)$-time (for any $\delta>0$), $O(n)$-space randomized algorithm for finding all k intersections of n line segments in the plane (we can count the number of these intersections in $O(n^{4/3+\delta})$ time and linear space). (ii) Preprocessing a collection of n (possibly intersecting) segments in the plane so that, given any query ray, we can find quickly the first segment it hits. Other applications concern "implicit" point location, hidden surface removal in three dimensions, polygon placement queries, and problems involving overlapping planar maps. We also present several efficient algorithms involving the analysis of the connectivity and other useful properties of arrangements of line segments.

1. Introduction

In this paper we present efficient algorithms for solving a variety of problems in computational geometry, using several enhancements and extensions of techniques that have been developed originally for half-plane or polygon range searching (see [E], [EW], [HW]). These techniques are based on constructing a *partition tree* T for a given set G of n points (here we consider only the planar case, although partition trees have also been constructed for points in higher dimensions), so that each node v of T represents a subset G_v of these points, with the following properties:

(i) The root of T represents the entire set G.

(ii) Each node v of T has a constant number of children $w_1, \ldots, w_M$ so that $\{G_{w_1}, \ldots, G_{w_M}\}$ is a partition of G_v.

(iii) For any line l that cuts the convex hull $conv(G_v)$, for some node v, there exist only a few of the children w_i of v for which l cuts $conv(G_{w_i})$; moreover, the total size of the sets G_{w_i} for such children w_i, is relatively small compared with the size of G_v.

Note that partition trees have only $O(n)$ nodes, so that they require only linear storage. Even when one wishes to store auxiliary information in these nodes (as will be the case in some of our applications), one can still attin $O(n)$ storage, using some tricks proposed in [DE]. The exact requirements in condition (iii) vary from one kind of partition tree to another. Roughly speaking, this condition facilitates the following "propagation" procedure for a "query" line l. We begin by passing l to the root of T. Whenever l reaches a node v of T (which means that l cuts $conv(G_v)$), we pass l down the tree only to children w of v for which l intersects $conv(G_w)$. For other children w we know that all the points in G_w lie on the same side of l, and we take advantage of this knowledge for the particular application at hand.

The name of the game in partition trees is to construct trees satisfying as sharp a version of condition (iii) as possible, so as to ensure that a line l visits only a small number of nodes of T, thereby obtaining efficient query time. Various versions of partition trees have been recently developed, yielding a query time of the form $O(n^\gamma)$, for various $\gamma<1$. There is a lower bound of 1/2 for γ which is actually attainable (see [W], [Ch2], [Ch3]). The second best known value for γ is $2/3+\delta$, for any $\delta>0$ [HW], and the third prize goes to $\gamma \approx 0.695$ [EW]. In our solutions we will mostly use the second kind of trees. This kind (with $\gamma \approx 2/3$) is based on random sampling techniques and thus yields a randomized procedure for constructing the desired tree (the procedure will always produce a partition tree, and, with high probability, it will be a tree which yields the asserted query time for any query line). The third kind of tree, known as *conjugation tree, ham-sandwich tree* ([EW], [E]) is constructed deterministically, and can be used as a slightly weaker alternative, should randomized preprocessing be undesired. The first (and best) kind of tree seems unfortunately to be too unstructured to be amenable to some of the enhancements that our solutions require. In what follows we will state our results using only the second kind of tree.

The results of this paper thus aim to support the observation that partition trees are much more versatile and have a wider range of applications than their traditional use in range searching. Observations of this kind have already been made earlier; see [DE], [CSY], [Ch], [E]. Our results extend the domain of such applications, promote the benefits of using query batching (a technique that has been very little applied before), and develop some extra special tools needed to handle some of our applications.

The results of this paper include:

(i) An $O(n^{4/3+\delta} + k)$ time (for any $\delta>0$), $O(n)$ space randomized algorithm for reporting all k intersections of n line segments in the plane; we can count the number of these intersections in $O(n^{4/3+\delta})$ time and linear space.

(ii) Preprocessing a collection of n (possibly intersecting) segments in the plane so that, given any query ray, we can find quickly the first segment it hits. The preprocessing is randomized and requires $O(n \log^3 n)$ time and linear space, and allows a ray-shooting query to be answered in time $O(n^{2/3+\delta})$, for any $\delta>0$.

Other applications involve certain ray-shooting problems in three dimensions, implicit point location in planar maps produced by overlapping figures, polygon placement queries, and solving various optimization problems involving two overlapping planar maps.

Here is a bird's-eye view of our techniques. The problems mentioned above involve various interactions between a collection of (generally intersecting) line segments (or lines) and points in the plane. These problems can generally be solved in quadratic time and space by constructing the arrangement of the given lines or segments and then searching over it to produce the desired information (sometimes the solution involves arrangements in the dual plane). For example, problem (i) is trivially solved this way. To improve upon this quadratic cost, we use plane partitioning techniques (i.e. partition trees) as a divide-and-conquer device which allows us to partition the problem recursively into subproblems of smaller size.

However, to make these trees work for our applications, we need to modify and enhance them with certain new features. The two main enhancements that we use are *query batching* and *multi-level structures*. Query batching is applicable when the query lines to be processed through our partition tree are known in advance. In this case we can construct a "customized" version of the tree which achieves superior performance for these lines (without guaranteeing any efficiency for other lines); batching is used in our segment intersection algorithm. Multi-level versions of partition trees are required when the objects to be stored in the tree, or the queries to be processed through the tree are more complex than just points or lines. For example, one way to store n segments in a partition tree might be to construct a primary tree in the dual plane for the points dual to the lines containing the segments, and then attach to each node of the tree an auxiliary partition tree in the primal plane for the endpoints of the segments (this is how our ray shooting problem (ii) is handled). Beyond the specific applications presented in this paper, we advocate the use of these enhancements as general paradigms for solving other problems using partition trees.

These paradigms are of course well known in other contexts; e.g. the multidimensional range and segment trees [M], [Sa] are multilevel structures; batching has been used in [Pr] for certain point location problems. The use of query batching in partition trees has been fairly limited (see [CSY] for a technique somewhat resembling query batching in partition trees). The use of multi-level structures for partition trees has been initially proposed by Dobkin and Edelsbrunner [DE]. We use a somewhat similar approach, and extend it to many other applications.

While a main theme of this paper concerns applications of partition trees, some of these applications require additional technical tools which are non-trivial and are of independent interest; see for example the linear ray-shooting ordering of a collection of segments analyzed in Section 3.

Another theme that is addressed in this paper is the analysis of various structural properties of arrangements of segments, and the development of new tools to investigate such properties. The intersection reporting and counting problem (i) mentioned above typifies this kind of problems. We also study several other problems, related to the connectivity of arrangements of segments, and obtain efficient algorithms for their solution. Arrangements of segments have received considerable attention recently (see [CE], [Cl2], [EGS], [EGHSSSW], [PSS], [GSS] for example), and the results of this paper contribute to this growing body of research.

The paper is organized as follows. In section 2 we describe our solution to the segment intersection problem, in section 3 we present the ray-shooting algorithm, in section 4 we provide some results involving analysis of the connectivity of an arrangement of line segments, which are related to the segment intersection problem, and in section 5 we list other applications of our techniques.

2. Efficient calculation of the intersections of line segments

Let $G = \{e_1, \ldots, e_n\}$ be a collection of line segments in the plane. We derive in this section a (randomized) algorithm for reporting all k intersections of the given segments in time $O(n^{4/3+\delta}+k)$ and $O(n)$ space, for any $\delta>0$. Recently, Chazelle and Edelsbrunner [CE] have obtained a time-optimal solution to this problem; their algorithm runs in $O(n \log n + k)$ time but uses $O(n+k)$ space. Our algorithm is space optimal (and also time optimal for $k \gg n^{4/3}$). After the original version of this paper, Clarkson [Cl2] has obtained an improved randomized algorithm for reporting all k intersections in expected time $O(n \log n + k)$ and in linear space. A variant of our algorithm can count the number of intersections between the given segments in (randomized) time $O(n^{4/3+\delta})$ and $O(n)$ space, for any $\delta>0$ (such an output size independent performance does not appear to be possible for Clarkson's algorithm). The best previous general solution to this problem is due to Chazelle [Ch] and runs in time $O(n^{1.695})$; an improved $O(n^{1.5}\log n)$ algorithm for a special case is given in [MS]. The solution presented in this section exemplifies the use of query batching in partition trees.

Let $p_1, \ldots, p_m$ $(m \leq 2n)$ be the endpoints of the given segments. We construct a partition-tree T for this set of points, with a predetermined set of query lines, consisting of the lines $l_1, \ldots, l_n$ containing the segments $e_1, \ldots, e_n$. The tree T is constructed top-down in a recursive manner. At each node v of T there is associated a convex polygonal region Q_v (obtained from the plane partitionings done at the ancestors of v), the subset P_v of the endpoints p_i which lie inside Q_v, and the subset L_v of the query lines l_j that intersect Q_v. The recursive goal at v is to report all intersections of the segments e_i which lie within Q_v. We put $n_v = |P_v|$, $m_v = |L_v|$.

The root of T corresponds to the entire plane. Let v be a node of T. If $m_v \geq n_v^2$ we do not continue the construction of T below v (so v is a leaf of T), and instead apply the procedure described below to obtain all intersections within Q_v. Otherwise we partition Q_v into a collection of subregions, using a technique akin to the ϵ-net approach of Haussler and Welzl [HW] or the random sampling technique of Clarkson [Cl]. Thus we fix some integer r, draw a random sample of r of the lines in L_v, clip each sample line to obtain its portion within Q_v, construct the arrangement of these clipped portions, and triangulate each face of this arrangement. By the ϵ-net theory, with high probability, each of the resulting $M = O(r^2)$ triangles will be cut by at most

$O(\frac{cm_v}{r} \log r)$ lines of L_v, for some constant c. We then create M children $w_1, \ldots, w_M$ of v; each w_i is assigned one of the triangles Q_{w_i} of this "sample arrangement", the corresponding subset P_{w_i} of the points of P_v within Q_{w_i}, and the subset L_{w_i} of the lines in L_v which cut Q_{w_i}. In addition, we also record the (at most two) intersection points of each line l_j in L_{w_i} with ∂Q_{w_i}; if the portion of l_j within Q_{w_i} is disjoint from e_j we exclude l_j from L_{w_i}. The overall cost of this expansion step at v is $O(m_v+n_v)$ (randomized) time.

To achieve space efficiency, the tree T is constructed in a depth-first manner. Thus at any given time we maintain only a single path within T. Moreover, in passing from a node v to one of its children w, the lines in L_w are taken away from L_v, and are placed back there upon returning from w. It is easy to see that in this way only linear space is needed to maintain T.

The heart of our procedure is the processing of nodes v of T that lie at the bottom of the recursion. Let v be such a node (so $m_v \geq n_v^2$). Let A_v denote the set of segments cutting Q_v but having no endpoint within Q_v, and let B_v be the set of segments having an endpoint within Q_v. We have $|B_v| \leq n_v$, $|A_v| \leq m_v$.

The intersection-reporting procedure at v consists of three substeps: intersections between pairs of segments of A_v, intersections between pairs of segments of B_v, and intersections between segments of A_v and segments of B_v. We show that one can report all k_v intersections (of any of these three kinds) within Q_v in time $O(m_v \log n_v + k_v)$ and space $O(m_v)$. Since $m_v \geq n_v^2$, we are allowed quadratic time and space in n_v to find the last two kinds of intersection, which makes these tasks fairly simple. The first kind of intersection is computed by a circular scan of the boundary of Q_v, noting that two segments in A_v intersect within Q_v if and only if their endpoints along ∂Q_v interleave. More details are given in the full version of the paper.

We can now analyze the time performance of the entire algorithm. Arguing as in [EGS], it follows that the time $T(m_v,n_v)$ needed to process recursively a node v of the tree obeys the following recurrence relationship (where k_v is the number of intersections within the corresponding region Q_v).

$$T(m_v,n_v) = O(m_v \log n_v + k_v), \quad \text{if } m_v \geq n_v^2,$$

$$T(m_v,n_v) = \sum_{i=1}^{M} T(m_{w_i},n_{w_i}) + O(m_v+n_v), \quad \text{if } m_v < n_v^2,$$

where $M = O(r^2)$, $m_{w_i} = O(\frac{m_v}{r} \log r)$ for all i, and $\sum_{i=1}^{M} n_{w_i} = n_v$. One can then show, as in [EGS], that the solution of this recurrence satisfies

$$T(m_v,n_v) = O(m_v^{2/3-\delta} n_v^{2/3+2\delta} + (m_v+n_v) \log n_v + k_v)$$

for any $\delta > 0$. Substituting the root of the tree in this formula, we obtain

Theorem 2.1. One can report all k intersections between n given segments in (randomized) time $O(n^{4/3+\delta} + k)$ and in $O(n)$ space, for any $\delta > 0$.

By a fairly simple modification of these procedures, we also show

Theorem 2.2. The number of intersections between n given line segments can be calculated in (randomized) time $O(n^{4/3+\delta})$ and in $O(n)$ space, for any $\delta > 0$.

3. Ray Shooting in Non-simple Polygons

We now consider another application of the partition tree technique, this time illustrating the second enhancement, namely using a multilevel tree structure (see [DE] for a similar development with different applications). Let P be a (connected) polygonal region having n edges. We wish to preprocess P so that, given any query ray r emerging from any point inside P, we can determine quickly the first intersection of r with ∂P (or determine that r does not intersect ∂P, which might be the case when P is unbounded). If P is simply connected, there exist efficient techniques for

solving this problem [CG], [GHLST] which require $O(\tau(n))$ preprocessing time, $O(n)$ space, and have $O(\log n)$ query time (here $\tau(n)=O(n \log\log n)$ is the time needed to triangulate an n-sided simple polygon [TV]; see also [CTV]). However, if P is not simply connected, the problem appears to be more difficult. For example, Suri and O'Rourke [SO] have studied the restricted problem of ray shooting into a non simply connected polygon P (or actually testing visibility inside P) from a given edge of P. They show that the portion of P visible from an edge is a polygonal region that can be described as the union of $O(n^2)$ triangles, and which, in the worst case, can have $\Omega(n^4)$ edges on its boundary. We show, nevertheless, that the analysis of [SO] is over-pessimistic, and that the more general version of the problem can be attacked using the partition-tree approach, and can thus be solved relatively efficiently.

For technical reasons, and without loss of generality, we will consider only rays that are rightward-directed. Let $e_1, \ldots, e_n$ be the edges of P. We use a multi-level data structure, such that each level is based on a partition-tree for a different type of data points and query lines. The primary level is a partition-tree T for the set $G^*=\{l_1^*, \ldots, l_n^*\}$ of dual points to the lines l_i containing the edges e_i of P. The dual q^* of the origin q of the query ray r will serve as a query line to be propagated through T. At each node v of T there is associated a subset G_v of the edges of P (so that the set G_v^* of the duals of the lines containing the segments in G_v, is the set of points l_i^* lying in the convex face of the partitioning that v represents), and the recursive goal at v is to find the nearest intersection of r (if any) with any of the segments in G_v. If v is a leaf of T this task is trivially done in constant time, and if v is an inner node of T, then the answer to the query at v is easily obtained in constant time, provided we already know the answer at each child of v.

There are two ways to get the desired solution at each child w of v. If the query line q^* reaches w, then r and q^* are propagated to w and the answer is obtained recursively. Otherwise, q^* "misses" w, so the point q must either lie above all the lines containing the segments in G_w, or lie below all these lines. Assume without loss of generality the first case.

We define a linear ordering $<$ on the segments in G_w which is consistent with our ray-shooting problem, in the sense that if our ray r (emerging from a point q as above) hits two segments e,e' in G_w so that the intersection of r with e lies to the left of its intersection with e', then $e<e'$. This will enable us to apply binary search (in a non-trivial way, though) to find the first intersection of r with these segments.

We believe that this linear ordering may be of independent interest and find applications in other algorithms. Similar orderings for other classes of planar objects have been investigated by Guibas and Yao [GY] and by Rosenstiehl [Ro].

3.1. Linear Ordering of Segments for Ray Shooting

Let $e_1, \ldots, e_n$ be n disjoint (and non-vertical) segments in the plane. For each e_i, its *top side* e_i^+ is that side of e_i visible from a point at $y=+\infty$, and its *bottom side* e_i^- is the opposite side.

Definition. (i) A (non-vertical) line l is said to *hit* e_i at its top (resp. bottom) side, or simply to hit e_i^+ (resp. e_i^-), if l intersects e_i transversally at some point x, and, slightly to the left of x, l lies above (resp. below) e_i.

(ii) $e_i <_s e_j$ if there exists a (non-vertical) line l hitting both e_i^+ and e_j^+ such that its intersection x_i with e_i lies to the left of its intersection x_j with e_j, and such that l does not hit any bottom side of any other e_k at a point between x_i and x_j.

(iii) $e_i <_v e_j$ if there exists a vertical line intersecting both e_i and e_j such that its intersection with e_i lies above its intersection with e_j.

(iv) $e_i <_x e_j$ if e_i and e_j have non-overlapping x-projections and the projection of e_i lies to the left of that of e_j.

(v) $e_i <_{v^*/x} e_j$ (also denoted shortly as $e_i < e_j$) if either e_i precedes e_j in the transitive closure $<_{v^*}$ of $<_v$, or e_i and e_j are not related by $<_{v^*}$ and $e_i <_x e_j$.

We show

Theorem 3.1. $<_s$ is a partial order, and $<_{v^*/x}$ is a linear order which extends $<_s$. Moreover, $<_{v^*/x}$ can be calculated in $O(n \log n)$ time.

Proof. Omitted in this version.

3.2. The Ray Shooting Procedure Continued

In the special case of our shooting problem, the rays that we consider cannot hit the bottom side of any segment in G_w. We thus obtain

Lemma 3.2. Suppose $e, e' \in G_w$ and $e <_s e'$. If a rightward-directed ray r emerges from a point lying above all the lines containing the segments in G_w and hits e, then either r does not hit e' or it hits it at a point lying to the right of the intersection of r with e.

Proof. Simple, and omitted in this version.

Lemma 3.2 implies that if our query ray r is known to hit every segment in some subset of G_w, then the first of these segments in the order $<_{v^*/x}$ is the first that r hits. Moreover, it is easily verified that r intersects a given segment e_i if and only if the line l containing r hits the top side of e_i, which implies that the left endpoint of e_i must lie below l whereas the right endpoint of e_i lies above l. We thus use a second-level partition tree T'_w for the set H_w of the left endpoints of the segments in G_w (for which the line l serves as a query object), and further attach to the nodes of T'_w similar third-level partition trees for the corresponding sets of right endpoints of the segments associated with these nodes. It is not difficult to verify (but we omit details here) that for every node z of a tertiary tree that l misses (but does not miss the parent of z), l hits either all or none of the segments associated with z. Using the above observations concerning our ordering, we can thus collect the first of the segments in each such node, and thus readily obtain the desired segment first hit by r. By using the space-saving trick of [DE], we can reduce the space requirement for all our auxiliary trees to linear size, without substantially degrading query performance.

Theorem 3.3. Given a polygonal region P with n edges, one can preprocess it in time $O(n \log^3 n)$ into a data structure of linear size, using which one can answer ray shooting queries in time $O(n^{2/3+\delta})$, for any $\delta > 0$.

As we show in the full version, this technique can be extended to solve ray shooting queries for an arbitrary collection of n (possibly intersecting) segments. Specifically, we show

Theorem 3.4. Given a collection of n segments in the plane, one can preprocess it in time $O(n\alpha(n) \log^3 n)$ into a data structure of size $O(n\alpha(n))$, using which one can answer ray shooting queries (in which we seek, for a given query ray, the first segment it hits) in time $O(n^{2/3+\delta})$, for any $\delta > 0$ (here $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function).

4. Connectivity and Related Problems for Line Segments

Another class of problems that can be addressed with techniques similar to those presented here and more fully developed in [EGS] concerns connectivity questions in arrangements of line segments. As before, let $G = \{e_1, e_2, \ldots, e_n\}$ denote a collection of n line segments in the plane. The arrangement $A(G)$ of these segments is the subdivision of the plane they define. The segments in G can naturally be partitioned into the *connected components* of their union. Our goal is to compute the connected components of G quickly.

We will call a face of $A(G)$ *interesting* if it contains an endpoint of one of the segments in G on its boundary; otherwise we will call it *dull*. Dull faces are convex polygons while, as is easy to see, interesting faces need not even be simply connected. The boundary of an interesting face will in general consist of one outer and several inner edge loops in $A(G)$. This loop view requires that we regard each arrangement edge as being duplicated, so that each of its sides is counted separately; then each such edge loop can be viewed as a simple polygon whose vertices are either intersection points or endpoints of the segments in G. Notice that there are only $O(n)$ interesting faces in $A(G)$, while the total number of faces can be as high as $\Theta(n^2)$. The edge loops of (the interesting faces of) $A(G)$ are important for the connected components problem because of the

following lemma:

Lemma 4.1: The segment intersections that appear along the edge loops of the arrangement $A(G)$ define a subgraph of the intersection graph of G whose connected components are the same as the connected components of G.

The proof is omitted in this abstract. In [EGS] it is shown that the total combinatorial complexity of any $O(n)$ faces in $A(G)$ is $O(n^{4/3+\delta})$ for any positive δ. It follows that the number of segment intersections along edge loops of $A(G)$ is also at most $O(n^{4/3+\delta})$. The import of the above lemma is then that one can find the connected components of G by looking only at the intersections along edge loops, whose number can be much less than the total number of intersections of the segments in G ($\Theta(n^2)$ in the worst case). In [EGS] it is also shown how one can obtain a randomized algorithm that computes the boundaries of the interesting faces within the same time bound $O(n^{4/3+\delta})$. Our result then is:

Theorem 4.2: It is possible to compute the connected components of a set of n line segments in the plane in $O(n^{4/3+\delta})$ time and space.

In the full version of the paper we the following related results.

Theorem 4.3: Given a collection R of n red segments with disjoint (relative) interiors, and a collection B of n blue segments with disjoint (relative) interiors, then the connected components of $R \cup B$ can be computed (deterministically) in time $O(n \log^2 n)$ and space $O(n)$.

Theorem 4.5: Given n segments with k pairwise intersections such that they form a single connected component and all their endpoints lie on the unbounded face of their arrangement, then if the boundary of that face is given to us (say in the edge-loop format discussed above), we can sweep the rest of the arrangement (going from the outside "inwards") in time $O(n+k)$ and working storage $O(n)$.

See [EG] for description of topological sweeping of arrangements of lines. We leave it as an open problem whether similarly efficient (and simple) sweeping algorithms exist for general arrangements of segments.

5. Further Applications

In this section we list other applications of the partition tree technique, and some other related results.

5.1. Implicit point location:

Let M be a planar map defined by the superposition of n (possibly intersecting) planar figures $B_1, \ldots, B_n$ of some simple shape; in what follows we restrict our attention to polygonal figures, such as triangles, strips, half-planes, etc. In the worst case, M might have quadratic complexity. The goal is to preprocess M (in subquadratic time and space) so that, given any query point x, we can locate it in M and calculate certain information about the interaction between M and x. For example, we may want to find whether x lies in the union of the B_i's, or find how many B_i's contain x, or, assuming each B_i has a weight associated with it, sum up the weights of the B_i's containing x, etc. We show

Theorem 5.1: (i) One can preprocess such a map M in (randomized) time $O(n \log^r n)$ (where r is some small fixed integer depending on the nature of the problem) and space $O(n \log n)$, so that for each query point x, the desired information at x can be obtained in time $O(n^{2/3+\delta})$, for any $\delta>0$.

(ii) Given (in advance) m query points, we can calculate the desired data at each of them in total time

$$O(m^{2/3-\delta} n^{2/3+2\delta} + m \log n + n \log^{r+1} n)$$

for any $\delta>0$.

5.2. Hidden surface removal

Given a collection T of objects in 3-dimensional space, and a viewing point a, we wish to calculate the scene obtained by viewing the objects in T from a. In other words, we want to produce some representation of T, in which we can determine, for each ray emerging from a, the first surface of an object in T hit by that ray (or determine that the ray does not his any object). Known solutions to this problem run in time and space $O(n^2)$ [De], [MK], which is of course worst-case optimal if explicit calculation of the entire image is desired. Our techniques can be used to obtain improved implicit solutions (a similar, and more efficient, solution has been given in [CS] for a very restricted case of this problem). A somewhat related technique (yielding weaker bounds) has recently been described in [SML].

We will consider the following special case of the hidden surface removal problem. Let $T = \{\Delta_1, \ldots, \Delta_n\}$ be a collection of n horizontal triangles in 3-space such that Δ_i lies in the plane $z=i$. The problem is to determine, for each point p in the xy-plane, the first (i.e. lowest) triangle Δ_j hit by the vertical line passing through p (or to determine that no such triangle exists). What distinguishes this special case from the general surface removal problem is that in this case there exists a natural total ordering (by their z-coordinates) of the triangles in T, with the property that if a vertical line hits both Δ_i and Δ_j with Δ_i lying below Δ_j, then $\Delta_i < \Delta_j$. Our results will actually apply to any instance of the surface removal problem in which such an ordering exists. We show

Theorem 5.2: (i) The preprocessing version of the hidden surface removal problem for an ordered collection of n triangles in 3-space can be solved in $O(n \log^3 n)$ (randomized) preprocessing time, $O(n \log^2 n)$ space, and $O(n^{2/3+\delta})$ query time, for any $\delta>0$.

(ii) The batched version of this problem (where m point queries are given in advance) can be solved in (randomized) time

$$O(m^{2/3-\delta} n^{2/3+2\delta} + m \log n + n \log^3 n)$$

for any $\delta>0$.

5.3. Polygon placement queries

Let $A_1, \ldots, A_m$ be m polygonal obstacles in the plane having pairwise disjoint interiors, and let n denote the total number of their edges. Let B be an arbitrary polygonal region defined as the union of k simply-shaped polygons (e.g. line segments or triangles). (Thus the explicit complexity of B might be as high as $\Omega(k^2)$.) We want to preprocess these data so that, given any query displacement x, we can determine whether $B + x$ does not intersect any obstacle, or, in other words, whether translating B by the vector x yields a placement of B which is free of collision with any obstacle. Under further restrictions on the shape of B there exists alternative efficient solutions (see [KLPS], [LS]). However for the general case assumed above, the following solution appears to be the best known.

Theorem 5.3: With (randomized) preprocessing time $O(kn \log^2 kn)$ and space $O(kn \log kn)$, one can obtain a data structure, using which one can determine, for a given query placement of B, whether B is free of collision with the obstacles at that placement, in time $O((kn)^{2/3+\delta})$, for any $\delta>0$.

5.4. Overlapping planar maps

Let M_1, M_2 be two planar maps, each having at most n faces. Suppose each face f of M_1 or of M_2 is assigned some weight $w(f)$. We want to find a point x in the plane which minimizes the sum of the weights of the two faces, one of M_1 and one of M_2, containing x. There is a trivial $O(n^2)$ algorithm. However, using a variant of the segment intersection algorithm, we show

Theorem 5.4: The above overlapping maps problem can be solved in (randomized) time $O(n^{4/3+\delta})$ and linear space, for any $\delta>0$.

References

[Ch] B. Chazelle, Reporting and counting segment intersections, *J. Computer Systems Sciences* 32 (1986) pp. 156-182.

[Ch2] B. Chazelle, Polytope range searching and integral geometry, *Proc. 28th Symp. on Foundations of Computer Science,* 1987, pp. 1-10.

[Ch3] B. Chazelle, Tight bounds on the stabbing number of spanning trees in Euclidean space, in preparation.

[CE] B. Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, Tech. Rept. CS-TR-148-88, Comp. Sci. Dept., Princeton University, April 1988.

[CG] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Proc. 1st ACM Symposium on Computational Geometry* 1985, pp. 135-146.

[Cl] K. Clarkson, New applications of random sampling in computational geometry, *Discrete Comput. Geom.* 2 (1987) pp. 195-222.

[Cl2] K. Clarkson, Applications of random sampling in computational geometry, II, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[CTV] K. Clarkson, R. Tarjan and C. Van Wyk, A fast Las Vegas algorithm for triangulating a simple polygon, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[CS] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, Tech. Rept. 266, Comp. Sci. Dept., Courant Institute, 1986.

[CSY] R. Cole, M. Sharir and C. Yap, On k-hulls and related problems, *SIAM J. Computing* 16 (1987) pp. 61-77.

[De] F. Devai, Quadratic bounds for hidden line elimination, *Proc. 2nd ACM Symp. on Computational Geometry,* 1986, pp. 269-275.

[DE] D. Dobkin and H. Edelsbrunner, Space searching for intersecting objects, *J. of Algorithms* 8 (1987), pp. 348-361.

[E] H. Edelsbrunner, *Algorithms in Combinatorial Geometry,* Springer Verlag, Heidelberg, 1987.

[EG] H. Edelsbrunner and L. Guibas, Topologically sweeping an arrangement, *Proc. 18th Symp. on Theory of Computing,* 1986, pp. 389-403.

[EGHSSSW] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink and E. Welzl, Implicitly representing arrangements of lines and of segments, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[EGS] H. Edelsbrunner, L. Guibas and M. Sharir, The complexity of many faces in arrangements of lines and of segments, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[EW] H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inform. Process. Lett.* 23 (1986) pp. 289-293.

[GHLST] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, Linear time algorithms for visibility and shortest path problems for triangulated simple polygons, *Algorithmica* 2 (1987) pp. 209-233.

[GSS] L. Guibas, M. Sharir and S. Sifrony, On the general motion planning problem with two degrees of freedom, *Proc. 4th ACM Symp. on Computational Geometry,* 1988.

[GY] L. Guibas and F. Yao, On translating a set of rectangles, in *Advances in Computer Research,* Vol. 1 (F.P. Preparata, ed.), pp. 61-77.

[HW] D. Haussler and E. Welzl, Epsilon nets and simplex range queries, *Discrete Comput. Geom.* 2 (1987) pp. 127-151.

[KLPS] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* 1 (1986) pp. 59-71.

[LS] D. Leven and M. Sharir, Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams, *Discrete Comput. Geom.* 2 (1987) pp. 9-31.

[MS] H. Mairson and J. Stolfi, Reporting and counting intersections between two sets of line segments, *Theoretical Foundations of Computer Graphics and CAD,* (R. Earnshaw, Ed.), NATO ASI Series, Vol. F-40, Springer Verlag, Berlin 1988, pp. 307-325.

[MK] M. McKenna, Worst case optimal hidden surface removal, *ACM Trans. Graphics* 6 (1987) pp. 19-28.

[M] K. Mehlhorn, *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, Springer Verlag, Heidelberg, Germany 1984.

[PSS] R. Pollack, M. Sharir and S. Sifrony, Separating two simple polygons by a sequence of translations, *Disrete Comput. Geom.* 3 (1988) pp. 123-136.

[Pr] F.P. Preparata, A note on locating a set of points in a planar subdivision, *SIAM J. Computing* 8 (1979) pp. 542-545.

[Ro] P. Rosenstiehl, Grammaires acycliques de zigzags du plan, manuscript, 1987.

[Sa] H. Samet, Hierarchical representations of collections of small rectangles, Tech. Rept. CS-TR-1967, Comp. Science Department, University of Maryland, January 1988.

[SML] A. Schmitt, H. Muller and W. Leister, Ray tracing algorithms - Theory and practice, *Theoretical Foundations of Computer Graphics and CAD*, (R. Earnshaw, Ed.), NATO ASI Series, Vol. F-40, Springer Verlag, Berlin 1988, pp. 997-1030.

[SO] S. Suri and J. O'Rourke, Worst case optimal algorithms for constructing visibility polygons with holes, *Proc. 2nd ACM Symp. on Computational Geometry*, 1986, pp. 14-23.

[TV] R. Tarjan and C. Van Wyk, An $O(n \log \log n)$ algorithm for triangulating simple polygons, *SIAM J. Computing* 17 (1988) pp. 143-178.

[W] E. Welzl, Partition trees for traingle counting and other range searching problems, *Proc. 4th ACM Symp. on Computational Geometry*, 1988.