Ray shooting, implicit point location, and related queries in arrangements of segments

L. Guibas, M. Overmars, M. Sharir

RUU-CS-89-11
May 1989

Rijksuniversiteit Utrecht

Vakgroep informatica

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

Ray Shooting, Implicit Point Location, and Related Queries in Arrangements of Segments

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

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

ABSTRACT

We extend and modify recent efficient techniques for half-plane range searching, to obtain algorithms for answering efficiently several types of queries on planar arrangements of segments and certain related structures. For example, we show that, given n segments in the plane, we can preprocess them in $O(n \log n)$ randomized expected time and $O(n)$ space, so that, given a query ray, emanating from an arbitrary point in an arbitrary direction, we can determine the first segment (if any) hit by the ray, in time $O(n^{2/3+\epsilon})$, for any $\epsilon>0$. Other results involve "implicit" point location in an arrangement of (possibly intersecting) triangles, polygon placement queries, and certain restricted types of ray shooting and hidden surface removal for 3-dimensional polyhedral scenes.

1. Introduction

We begin with the following somewhat unusual opening remarks. The results of this paper have been conceived and developed in the summer of 1987, but for various reasons were left unpublished for almost two years. Meanwhile, there has been significant progress on the problems studied here. In particular, a recent work by Agarwal [Ag] present techniques that achieve query time roughly $O(\sqrt{(n)})$, and involve deterministic, rather than randomized, preprocessing (however, the preprocessing time no longer remains close to linear). Agarwal's results are based on a decomposition technique that is different from the one used in this paper. Still, many of Agarwal's arguments are taken from this paper. As a public service, to make our results somewhat more accessible, we have decided to elevate our paper from the status of "unpublished manuscript" to that of a technical report.

Let G be a collection of n segments $e_1, \ldots, e_n$ in the plane, and let $A = A(G)$ be the *arrangement* of these segments, that is the planar map whose vertices are the endpoints and the intersection points of the segments, whose edges are maximal connected subsegments of the segments not containing any vertex, and whose faces are the connected components of the

Work on this paper by the third author has been supported by Office of Naval Research Grant N00014-82-K-0381, by National Science Foundation Grant No. NSF-DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation, and by a research grant from the NCRD - the Israeli National Council for Research and Development.

complement of the union of the segments. In this paper we consider a variety of problems in which we wish to process such an arrangement A so as to be able to answer certain queries involving A.

Typically, the queries we are interested in are trivially answered in $O(\log n)$ time, if we are willing to calculate and store A explicitly. Since the combinatorial complexity of A can be quadratic in n in the worst case, this might be too expensive, and we therefore seek alternative methods of more efficient but implicit representation of A, which still facilitates reasonably efficient query handling, although not as efficient as the (optimal) $O(\log n)$ time that can be achieved with explicit representation.

As a typical example, consider the following implicit point location problem. Given such an arrangement A, preprocess it so that, given any query point x, we can calculate quickly certain properties of the face of A containing x. For example, suppose A is formed as the overlay of n (possibly intersecting) triangles, and the property in question is how many triangles contain the query point x, or more generally, if each triangle is associated a certain weight, what is the sum of the weights of the triangles covering x, etc. Clearly, by calculating the arrangement A induced by these triangles, and by further preprocessing it for efficient point location, it is easy to answer such a query in (optimal) $O(\log n)$ time. Can we obtain faster preprocessing and more compact representation of A that will still enable us to answer these queries reasonably efficiently?

In this paper we show that this is indeed possible. We extend and modify several recent techniques, based on partition trees for sets of points in the plane, which have been originally developed for half-plane and polygon range searching [HW], [EW], [DE]. Using such partition trees, we are able to solve the implicit point location problem, as well as many other related ones, using $O(n \text{ polylog}(n))$ preprocessing time and space, and $O(n^{2/3+\epsilon})$ query time, for any $\epsilon>0$.

Perhaps the most interesting application that we have is that of *ray shooting* in non-simple polygons, or, more generally, in an arbitrary arrangement of (intersecting) segments. In this problem we wish to preprocess the given segments so that, given any query ray ρ, emanating from an arbitrary point in an arbitrary direction, we can determine the first segment (if any) hit by ρ. Using a somewhat complex partition-tree technique, and an interesting development of a linear ordering of the segments which is consistent with the order in which they are hit by certain types of rays, we obtain an algorithm that uses $O(n \text{ polylog}(n))$ preprocessing time and space, and achieves $O(n^{2/3+\epsilon})$ query time, for any $\epsilon>0$. In the case of a simple polygon, Chazelle and Guibas [CG] have developed a ray shooting procedure that uses $O(n \log\log n)$ preprocessing time and $O(n)$ space, and achieves optimal $O(\log n)$ query time; however, no efficient solution was known for the non-simple case (see some further discussion on this problem in Section 3).

We also present additional applications of our technique, including

(i) *Polygon placement queries*. Here, given an arbitrary polygonal object P and a collection of polygonal obstacles, we wish to determine whether a specific query translation of P will move it to a free placement, in which it does not intersect any obstacle.

(ii) *Implicit hidden surface removal in three dimensions*. Here, given certain collections of triangles in three dimensions, we wish to preprocess them so as to be able to answer efficiently ray shooting queries, each asking for the first triangle (if any) hit by a given query ray. This is a central subproblem of the ray-tracing problem in computer graphics [SML].

An important variant of these problems is when all the queries are known in advance, as is the case in many applications. We call this variant the "batched" version of these problems. We show that batched problems can be solved considerably more efficiently than unbatched ones, by constructing "customized" partition trees, which are tailored to handle efficiently only the given queries, disregarding efficiency considerations for any other query.

An important theme of this paper is to point out the versatility of the partition tree techniques, and their applicability to problems which do not always seem directly related to range searching. In particular we note that the batching techniques introduced in [EGSh] can be applied to a large variety of problems, much beyond the specific applications studied in [EGSh]. This theme has also been observed and exploited by Dobkin and Edelsbrunner [DE], where a weaker form of partition trees (that is, "conjugation trees") is used for space intersection queries, in a manner similar to the one used here.

The paper is organized as follows. In section 2 we present efficient techniques to handle (certain variants of) the implicit point location problem. Section 3 analyzes the planar ray shooting problem. Section 4 presents techniques for handling certain three-dimensional ray shooting problems. Section 5 handles polygon placement queries. Concluding comments and discussion is given in Section 6.

2. Implicit Point Location

The planar point location problem is one of the most fundamental problems in computational geometry. It was studied in many papers, culminating in several optimal algorithms [Ki], [EGS], [ST]. In this problem one is given a planar subdivision (or map) M, consisting of n faces (and $O(n)$ edges and vertices), and the goal is to preprocess M into a data structure which supports fast point location queries, in which, given a query point x, we wish to determine the face of M containing x. The algorithms in [Ki], [EGS], [ST] achieve this goal in $O(n \log n)$ preprocesing time, $O(n)$ storage, and $O(\log n)$ query time; if the given map has some favorable properties (e.g. is a convex subdivision) then preprocessing can be done in linear time [Ki], [EGS].

A related (and simpler) problem is when the points to be located in M are all known in advance. In this case, if m points are to be located, one can use a simpler sweeping algorithm whose complexity is $O((m+n)\log n)$ [Pr]. This complexity is comparable with what one could get using the preprocessing approach, but the algorithm is much simplified. We call this version of the problem the "batched" version.

In this section we study the following generalization of the problem. Suppose the map M is not given to us explicitly, but rather it is defined as the *arrangement* of n (usually intersecting) geometric objects of some simple shape. We will assume that the given objects are line segments (or sometimes full or half lines); this will also enable us to handle general polygonal objects. As will be demonstrated below, this situation arises in many applications, including several visibility and placement problems. A naive approach to this modified problem is of course to calculate the arrangement M formed by the given segments, and then preprocess it for point location as above. Calculating the arrangement can be accomplished in time $O(n^2)$ and space $O(n^2)$, e.g. by calculating the arrangement of the lines containing the segments as in [EOS]. However, since the input size is only $O(n)$, we face the challenge to improve upon this naive approach, and obtain subquadratic solutions.

This goal is not always possible. For example, if the number m of points eventually to be located in M is $\geq n^2$, the above method is optimal, and no subquadratic performance is

possible. Even in this case, however, there still remains the problem of reducing the space requirement of the algorithm to subquadratic.

Another issue at hand is why point location in M is desired. Often this is needed for determining some "local" property of the query point. For example, if M is defined as the arrangement of n triangles, then we might want to determine, for a query point x, whether it is contained in the union of these triangles, or how many triangles contain x, or, if each triangle is assigned some weight, what is the sum of the weights of the triangles containing x, or the minimum-weight triangle containing x, etc. Similarly, for an arrangement of segments, we might seek, for a query point x, which segment lies directly below x. We think of these problems as being local, because they do not seek precise knowledge about the position of the query point x in the entire arrangement, but rather confine themselves to combining pieces of data, each obtained by "confronting" x with one of the objects, in some associative manner (which can be done naively in linear time).

This section assumes that point location in M is indeed required for such a "local" task. We show that in many cases these problems can be solved efficiently in subquadratic time and space, by avoiding explicit calculation of M, and by storing instead some implicit representation of it.

2.1. Statement of the Problem

For the sake of exposition, we will study the following specific problem. Given a collection $G = \{e_1, \ldots, e_n\}$ of n segments in the plane. With each segment $e \in G$ we associate a function ϕ_e defined on the entire plane which assumes values in some associative and commutative semigroup S (whose operation will be conveniently denoted by "+"). We assume that evaluation of $\phi_e(x)$ at a point x, as well as addition of two elements in S, takes constant time. Two versions of the problem concern us. In the *preprocessing* version, we want to preprocess the data into a data structure so that, given any query point x, we can calculate

$$\phi(x) = \sum_{e \in G} \phi_e(x)$$

efficiently. In the *batched* version, we are given m points $p_1, \ldots, p_m$, and we want to calculate $\phi(p_1), \ldots, \phi(p_m)$ efficiently.

The functions ϕ_e and ϕ are assumed to satisfy certain properties, which make their calculation closely related to point location. One assumption is that ϕ is constant on each face of the arrangement A formed by the segments in G (or at least that ϕ is constant on each subface obtained by some standard triangulation of other decompositions of A). We also assume that these constant values of ϕ can be obtained at constant cost per face as this arrangement is calculated.

To facilitate the following algorithms, we also need to make certain additional assumptions concerning the function ϕ. Roughly speaking, these assumptions require that the resulting data structure can facilitate especially fast calculation of $\phi(x)$ for points x which either lie above all the lines containing the segments in G, or lie below all these lines. See Condition A below for a more precise statement.

2.2. The Preprocessing-and-Query Version

We begin by describing the data structure to be used. Our description follows closely the one given in [EGSh], which in turn is an adaptation of the partition-tree approach to half-plane range searching [HW] and its extensions in [DE]. The reader is referred to these papers for more information concerning the construction.

Specifically, let l_i be the line containing e_i, for $i=1, \ldots, n$. Dualize these lines (in the manner used in [HW], [EGSh]), to obtain a collection G^* of points $\{l_1^*, \ldots, l_n^*\}$. Each query point p then becomes a dual query line p^*. We construct a *partition-tree* T for G^* as follows. With each node v of T we associate a convex subset Q_v of the dual plane, and the subset G_v^* of the dual points lying in Q_v. Let n_v denote the cardinality of G_v^*. If $n_v=1$, then v is a leaf of T. Otherwise the construction of T below v continues as follows. Take the set G_v^* and apply to it some partitioning scheme, consisting of a convex subdivision H_v of Q_v having certain useful properties for the desired point location. We will use here the scheme of [HW] which is based on ϵ-nets, and which yields the second-best query performance known to date. (The best is a recent technique of Chazelle and Welzl [We], [CW]. This technique does not seem at present amenable to the modifications we need to introduce for our applications, but has been successfully applied in [Ag] using different methods.) Specifically, for some arbitrary fixed $\epsilon>0$, the partition $H_v = H_v^\epsilon$ is a convex planar subdivision of Q_v having $M=O((\frac{1}{\epsilon}\log\frac{1}{\epsilon})^4)$ faces, such that any line l cuts at most $c=O((\frac{1}{\epsilon}\log\frac{1}{\epsilon})^2)$ faces of H_v, and the total number of points of G_v^* in the faces cut by l is at most ϵn. Such a subdivision always exists, and can be constructed in constant randomized time [HW]. We then create M children $w_1, \ldots, w_M$ of v, one per face of H_v, associate with each child w_j the corresponding j-th face Q_{w_j} of H_v, and the subset $G_{w_j}^*$ of the points of G_v^* in Q_{w_j}, and continue this partitioning process recursively at each w_j, thereby obtaining the desired tree T.

The tree T is used to process a query point p, or rather its dual line p^*, as follows. We propagate p^* through T, starting at the root of T. For each node v of T, let $\phi^v(p) = \sum_{e\in G_v} \phi_e(p)$. Suppose p^* reaches some node v of T during propagation. If v is a leaf, then $\phi^v(p)$ is simply $\phi_e(p)$, where e is the single element of G_v. Otherwise, by the properties of H_v mentioned above, p^* cuts at most c faces of H_v, and "misses" all the other faces. Let w_j be a child of v. If p^* cuts the face Q_{w_j} of H_v associated with w_j (this will be reflected by saying that p^* "reaches" w_j), then we obtain $\phi^{w_j}(p)$ recursively, by continuing the propagation of p^* past w_j. If p^* misses Q_{w_j}, we need an ad-hoc procedure, to be discussed below, for obtaining that value. Then we obtain $\phi^v(p)$ by straightforward summation

$$\phi^v(p) = \sum_{j=1}^{M} \phi^{w_j}(p) .$$

Note that when p^* misses the face Q_w of a node w, then the point p must lie either above all the lines l_i containing the segments e_i in G_w, or below all these lines. Thus we will assume that, using appropriate additional preprocessing, there is a fast way to compute $\phi^w(p)$ for such points p. This special-purpose procedure will vary from one application to another, but in all applications described below we will be able to carry out this computation in $O(\log n_w)$ time (as noted below, we can also tolerate here a much slower computation). Moreover, the additional preprocessing needed at w to facilitate this fast computation will usually be either $O(n_w)$ or $O(n_w \log n_w)$, and will consume only linear space. To be able to state our results in general but precise terms, we formalize these conditions as follows.

Condition A: Given a set G of n segments in the plane, one can preprocess it in time $O(n \log^r n)$ (for some small fixed integer r) into a linear-size data structure, so that, for any query point p lying either above all the lines containing the segments in G, or below all these lines, one can calculate $\phi(p)$ in $O(\log n)$ time.

Assuming Condition A, we can easily bound the query time $Q(n)$ as follows (see [HW] for more details).

$$Q(1) = O(1) ,$$

$$Q(n) = O(\log n) + \sum_{i=1}^{c} Q(n_i) , \quad n>1 ,$$

where $\sum_{i=1}^{c} n_i \le \epsilon n$.

We claim that $Q(n) = O(n^\gamma)$ for any $\gamma>2/3$. This is proven inductively, as in [HW], using the inequality

$$\sum_{i=1}^{c} n_i \le c^{1-\gamma} \cdot (\sum n_i)^\gamma \le \epsilon^\gamma \cdot c^{1-\gamma} \cdot n^\gamma$$

and the fact that $\log n = O(n^\gamma)$. Since $\gamma>2/3$ and $c = O((\frac{1}{\epsilon} \log \frac{1}{\epsilon})^2)$, it follows that if ϵ is chosen sufficiently small, $Q(n) = O(n^\gamma)$ will satisfy the required inequality. (Note that the same argument still applies if we replace $\log n$ by any function which is $O(n^\gamma)$. This allows us to relax Condition A accordingly, if necessary.)

As to the preprocessing time and space of the algorithm, it is easily checked that the space used by the data structure is $O(n \log n)$ (the partition tree T itself has only linear size, and the auxiliary structures at each node of T, as given by Condition A, require total additional $O(n \log n)$ storage). Similarly, the time needed to construct the tree and the auxiliary structures is easily seen to be $O(n \log^{r+1} n)$. Note however that since we use the ϵ-net construction of [HW], processing of each node of T also involves a (constant-time) randomized selection of an appropriate ϵ-net, which, with high probability, produces a set with the desired properties. See [EGSh] for more details concerning this issue, and also for an alternative fully deterministic construction of a similar partition tree, following the construction in [EW], which yields only slightly inferior query time.

Moreover, at the expense of slightly complicating the query processing, we can reduce the space to $O(n)$, with a similar saving of $O(\log n)$ in the preprocessing time, following the technique of [DE]. SPecifically, we store the auxiliary data structures only every $\sigma \log n$ levels. To compute $\phi(p)$ at a node v that p misses, we explore a complete subtree rooted at v and having depth at most $\sigma \log n$ until we reach a level where the auxiliary data structures are stored. We then compute $\phi(p)$ at each node of the subtree at that level (obviously, p misses those nodes as well), and sum up the results to obtain $\phi(p)$ at v. The cost of this extended operation is $O(\log n \cdot 2^{\sigma \log n}) = O(n^\delta)$, for any $\delta>0$, provided σ is chosen appropriately small. A slight modification of the above analysis shows that the query time is still $O(n^\gamma)$, for any $\gamma>0$, and the space requirement is now only $O(n)$, because the auxiliary structures are stored at only a constant number of levels. For the same reason, the preprocessing time also decreases by a factor of $O(\log n)$.

In summary, we have shown

Theorem 2.1: Given a collection of n segments in the plane, and an associated function ϕ as above which satisfies Condition A, one can preprocess these segments in (randomized) time $O(n \log^r n)$ into a data structure of size $O(n)$, so that, given any query point p, one can calculate $\phi(p)$ in time $O(n^\gamma)$, for any $\gamma > 2/3$.

2.3. The Batched Version

We now turn to consider the batched version of our implicit point location problem. The set-up is the same as assumed in the preceding subsection, except that now we know in advance the m points $p_1, \ldots, p_m$ at which we want to calculate the function ϕ. We use the approach of [EGSh], in which the partition tree T is constructed in a "customized" style, catering only to the query lines $p_1^*, \ldots, p_m^*$.

Specifically, we construct T as follows. As above, let l_i denote the line containing e_i, for each $i = 1, \ldots, n$. We construct recursively a partition tree T each of whose nodes v represents a convex polygonal region Q_v of the dual plane, the corresponding subset G_v of the n_v segments e_i for which the dual points l_i^* lie in Q_v (we denote this set of dual points by G_v^*), and the set L_v of the m_v points p_j whose dual lines p_j^* cut Q_v (we denote the set of these dual lines by L_v^*). If $m_v < n_v^2$, we expand the tree T at v as follows. We choose a random sample R of r of the lines in L_v^* (where r is some constant parameter to be chosen later), and draw and triangulate the arrangement A_v of these sample lines; we consider only the portion of A_v lying within Q_v. For each of the $M = O(r^2)$ faces f of this portion of A_v, we create a child w of v, take Q_w to be f, and obtain G_w, L_w according to the above definition. The ϵ-net theory of [HW] is easily seen to imply that, with high probability, the number m_w of lines p_j^* in L_w^* is at most $O(\frac{m_v}{r} \log r)$ for all children w. If $m_v \geq n_v^2$ we do not continue the construction of T below w, pass back to the primal plane, construct there the arrangement $A(G_v)$ of the n_v segments e_i in G_v, and assign to each face (or subface) of $A(G_v)$ the constant value of ϕ^v over that face. It is then straightforward to collect the faces of $A(G_v)$ that contain the m_v points p_i that have reached v, and obtain the value of ϕ^v at each of these points. All this can be easily done in time $O(n_v^2 + m_v \log n_v) = O(m_v \log n_v)$, and space $O(m_v)$.

For each child w of v, we collect all the points $p_i \in L_v$ whose dual lines have missed the face Q_w, and place them in a special "bucket" B_w associated with w. As above, if $p_1, \ldots, p_t$ are in B_w then, for each i, the point p_i lies either above all the lines l_j containing the segments $e_j \in G_v$, or below all these lines. Condition A implies that the values $\phi^v(p)$, for $p \in B_w$, can all be calculated in total time $O(b_w \log n_w)$, where $b_w = |B_w|$.

For an inner node v of T, we obtain $\phi^v(p_j)$ for all the points $p_j \in L_v$, by propagating these lines to the children $w_1, \ldots, w_M$ of v, obtaining $\phi^{w_i}(p_j)$ either recursively, if $p_j \in L_{w_i}$, or from the bucket B_{w_i} otherwise. Then the values of ϕ^v at all these points is readily obtained by summing these partial results, as above.

It follows that the time $T(m,n)$ needed to calculate the value of ϕ, for a given collection of n segments, at m given points, satisfies the following recurrence (in which we assume that the random sampling at each node of T does indeed produce an ϵ-net of the desired type, which will be the case with high probability):

$$T(m,n) = O(m \log n), \quad \text{if } m \geq n^2$$

$$T(m,n) = \sum_{i=1}^{M} T(m_i,n_i) + O(m \log n + n \log^r n), \quad \text{otherwise}$$

where m_i, n_i satisfy the following conditions.

(a) $\sum_{i=1}^{M} n_i = n$.

(b) $m_i = O(\frac{m}{r} \log r)$ for each i.

Hence, as in [EGSh], we obtain

Theorem 2.2: The batched version of the implicit point location problem, as formulated above, can be solved in (randomized) time

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

for any $\delta > 0$.

Remark: As already noted in [EGSh], Theorems 2.1 and 2.2 together show the power of prior knowledge of the points p_i to be "located". Indeed, when m is reasonably large, the above procedure spends roughly $O((n^2/m)^{1/3+\delta} + \log n)$ time per point, which tends to $O(\log n)$ as m approaches n^2.

2.4. An example - point-triangle containment queries

Let $\Delta_1, \ldots, \Delta_n$ be n (possibly intersecting) triangles in the plane. We wish to preprocess them so that, given any query point x, we can determine quickly whether x lies in the union $\bigcup_{i=1}^{n} \Delta_i$ of these triangles.

This containment problem fits nicely into the implicit point location mold presented above, as follows. Let Δ be a triangle; for each edge e of Δ let $B(e)$ denote the semi-infinite trapezoidal strip of points lying directly below e. Define a function ϕ_e in the plane so that $\phi_e \equiv 0$ outside $B(e)$, and $\phi_e \equiv +1$ on $B(e)$ if Δ lies below the line containing e, otherwise $\phi_e \equiv -1$ on $B(e)$.

It is easily checked that if e_1, e_2, e_3 are the edges of Δ then

$$\sum_{j=1}^{3} \phi_{e_j}(x) = \begin{cases} +1 & \text{if } x \in \Delta \\ 0 & \text{otherwise} \end{cases}$$

In other words, if we put $\phi(x) = \sum_e \phi_e(x)$, where the summation extends over all the edges of the triangles Δ_i, then $\phi(x) > 0$ if and only if x is contained in $\bigcup_{i=1}^{n} \Delta_i$. As a matter of fact, $\phi(x)$ gives the number of triangles Δ_i containing x. We can therefore reformulate our problem as that of calculating $\phi(p)$ at a query point p, and testing whether $\phi(p) > 0$. We thus want to apply the previous analysis to the collection G of edges $e_1, \ldots, e_{3n}$ of the triangles Δ_i.

To do so, we need to show, for any given subcollection G_ν of G, of cardinality n_ν, that

(i) ϕ^ν (as defined in Section 2.2) is readily obtainable during calculation of the entire arrangement of the segments in G_ν;

(ii) ϕ^ν satisfies Condition A.

To establish the first property, we calculate a modified arrangement formed by the n_ν segments in G_ν and of the $2n_\nu$ vertical rays extending downwards from their endpoints. Calculation of this arrangement still takes $O(n_\nu^2)$ time, and by definition ϕ is constant on each of

its faces, and can be obtained, in constant time per face, as the arrangement is being constructed.

As to the second property, let p be a point lying above all the lines containing the segments in G_v; then $\phi^v(p) = 0$ by definition. On the other hand, if p lies below all these lines, we can obtain $\phi^v(p)$ as follows. We project the $2n_v$ endpoints of the edges in G_v onto the x-axis and sort them in increasing order. Now, given such a point p, it is easily checked that

$$\phi^v(p) = \sum \{\epsilon_j : e_j \in G_v,\ p \in e_j^*\}$$

where e_j^* is the x-projection of e_j and where ϵ_j is the non-zero value of ϕ_{e_j}. Note that the sum on the right-hand side is constant over each interval between two successive projected endpoints, and we can obtain these values of ϕ^v by maintaining a running value Σ of this sum as we scan the list of projected points from left to right. Initially, $\Sigma = 0$. When we scan the left endpoint of some e_j^*, we add ϵ_j to Σ, and when we scan the right endpoint of e_j^* we subtract ϵ_j from Σ. We store the sorted list of projected endpoints, together with the associated values of ϕ_v. Thus, given a query point p lying below all the lines containing the segments of G_v, we can obtain $\phi^v(p)$ by a binary search through this list. This establishes Condition A (with preprocessing time $O(n_v \log n_v)$).

Remark: In general, the three edges of a triangle do not "stay together" as their dual points are partitioned among the nodes of the tree T. As a result, the values of the "intermediate" functions ϕ^v at inner nodes v of T need not bear any relationship to the original containment problem.

We thus obtain

Theorem 2.3: (i) The preprocessing version of the point-triangle containment problem for a collection of n triangles in the plane can be solved in $O(n \log n)$ randomized expected preprocessing time, $O(n)$ space, and $O(n^\gamma)$ query time, for any $\gamma > 2/3$.

(ii) The batched version of this problem for n triangles and m points, can be solved in randomized expected time

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

for any $\delta > 0$.

Remark: By straightforward modifications of the above procedure, we can obtain similar procedures for counting how many triangles Δ_i contain a query point x (this number is simply $\phi(x)$), or, assuming each triangle is assigned some weight w_i, for determining the sum of the weights of the triangles that contain a query point x (this is achieved by an appropriate modification of the edge functions ϕ_e), etc.

3. Ray Shooting in Non-simple Polygons

We now consider another major application of the techniques of Section 2. Let P be a (connected) polygonal region having n edges. We wish to preprocess P so that, given any query ray ρ emerging from any point inside P, we can determine quickly the first intersection of ρ with ∂P (or determine that ρ 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]). 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 of finding the visible portion of 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, which, in the worst case, can have $\Omega(n^4)$ edges on its boundary. We will show, nevertheless, that the analysis of [SO] is over-pessimistic, and that this more general version of the problem can be attacked using our 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 ρ 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 Q_v of the partitioning that v represents), and the recursive goal at v is to find the nearest intersection of ρ (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, because G_v contains just one segment. 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 (in the sense that it intersects the corresponding face Q_w) then ρ and q^* are propagated to w and the answer is obtained by recursive processing of w. Otherwise, q^* "misses" w, and a special procedure is needed to obtain the answer to the query at w.

Note that if q^* misses w, then 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 (the second case is handled in a completely symmetric fashion).

To handle this special situation, 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 ρ (emerging from a point q as above) hits two segments e,e' in G_w so that the intersection of ρ 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 ρ with these segments.

Similar orderings for other classes of planar objects have been investigated by Guibas and Yao [GY] and by Rosenstiehl [Ro]. Some of the proof techniques used below are adapted from [GY].

3.1. Linear Ordering of Segments for Ray Shooting

Let $e_1, \ldots, e_n$ be n segments in the plane with pairwise disjoint (relative) interiors. 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 (if e_i is vertical, we define its top side to be its left side, and take its bottom side to be undefined).

Definition: A (non-vertical) line l is said to *hit* e_i at its top side, or simply to hit e_i^+, if l intersects e_i transversally at some point x, and, slightly to the left of x, l lies above e_i. l is said to hit e_i^- when a symmetric condition holds. (If e_i is vertical, any line l intersecting e_i is said to hit e_i^+.)

Definition: $e_i <_s e_j$ 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.

Definition: $e_i <_v e_j$ 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.

Lemma 3.1: $<_v$ is acyclic.

Proof: This is also proved in [EGS], by showing that any cycle can be shortened by omitting the segment with the leftmost left endpoint. We will show, using a different proof, that in any finite collection E of segments, there always exists a segment that is minimal under $<_v$. This segment e is taken to be the one whose right endpoint (or top endpoint if it is vertical) can "see the sky" (i.e. the upward vertical ray from that endpoint does not hit any segment of E), and which is such that its right endpoint is leftmost among all segments in E satisfying this condition. To see that e is minimal, suppose to the contrary that there exists some $e' <_v e$ in E. Then it is easily checked that the right endpoint of e' lies above e. Let e'' be the segment whose right endpoint lies above e and is rightmost among all segments in E with this property. Then it is easily checked that the right endpoint of e'' can see the sky (and lies to the left of the right endpoint of e), a contradiction which completes the proof. □

Definition: $e_i <_x e_j$ 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.

Definition: $e_i <_{v^*/x} e_j$ (this is our main order relationship which will also be 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 $e_i <_x e_j$.

Lemma 3.2: $<_{v^*/x}$ is a linear order.

Proof: First observe that every pair e_i, e_j are related by this relationship. Indeed, if e_i and e_j are not related by $<_{v^*}$, then their x-projections are non-overlapping, which implies that they are related by $<_x$.

It remains to be shown that $<_{v^*/x}$ is acyclic. Let us first define yet another relationship: $e_i <_{(\neg v^*)\wedge x} e_j$ if e_i and e_j are not related by $<_{v^*}$ and $e_i <_x e_j$; thus $e_i <_{v^*/x} e_j$ if and only if either $e_i <_{v^*} e_j$ or $e_i <_{(\neg v^*)\wedge x} e_j$. By definition, $<_{v^*}$ is transitive: $e <_{v^*} e'$ and $e' <_{v^*} e''$ implies $e <_{v^*} e''$. We claim that $<_{(\neg v^*)\wedge x}$ is also transitive. Suppose $e <_{(\neg v^*)\wedge x} e'$ and $e' <_{(\neg v^*)\wedge x} e''$. Clearly $e <_x e'$ and $e' <_x e''$, and thus $e <_x e''$. The claim will thus follow if we show that e and e'' are not related under $<_{v^*}$. Suppose to the contrary that $e <_{v^*} e''$. Thus there is a chain of relationships $e = e_{i_0} <_v e_{i_1} <_v \cdots <_v e_{i_k} <_v e_{i_{k+1}} = e''$. Without loss of generality we can assume this chain to be maximal, i.e. that there is no longer chain of such inequalities between e and e''. Since $e_{i_p} <_v e_{i_{p+1}}$ in this maximal chain, there exists a vertical segment δ_p connecting a point a_p on e_{i_p} to a point b_p on $e_{i_{p+1}}$, with a_p lying above b_p, so that δ_p does not intersect any segment of our collection. Consider the path γ obtained by the concatenation

$$\gamma = \delta_0 || b_0 a_1 || \delta_1 || b_1 a_2 || \cdots || b_{k-1} a_k || \delta_k .$$

γ is a simple path that starts at a point to the left of e', ends at a point to the right of e', and does not intersect e'. Therefore γ must pass either above e' or below e'. But by definition of $<_v$ it is easily checked that in the first case we have $e <_{v^*} e'$, and in the second case $e' <_{v^*} e''$, a contradiction. A similar contradiction is obtained by a completely symmetric argument in the case $e'' <_{v^*} e$.

Now we can prove the acyclicity of $<_{v^*/x}$. Suppose to the contrary that there is a cycle

$$e_{i_1} < e_{i_2} < \cdots < e_{i_m} < e_{i_1}$$

and that this cycle is of minimal length m. First of all, it is easily seen that $m=3$. Indeed, $m=2$ is ruled out by definition, and any cycle of length >3 can be shortcut. Thus we can take our cycle to be $e < e' < e'' < e$. Now each of the three relationships along the cycle is either $<_{v^*}$ or $<_{(\neg v^*)\wedge x}$. Not all three relationships can be $<_{v^*}$, because that would imply that $<_v$ is cyclic, contrary to Lemma 3.1. Also, not all three can be $<_{(\neg v^*)\wedge x}$, because that would imply in particular $e <_x e' <_x e'' <_x e$, which is clearly impossible. Suppose two of the links are $<_{v^*}$ and one is $<_{(\neg v^*)\wedge x}$. Without loss of generality we can write $e <_{v^*} e' <_{v^*} e'' <_{(\neg v^*)\wedge x} e$. But by transitivity we obtain $e <_{v^*} e''$ which contradicts, by definition, the relationship $e'' <_{(\neg v^*)\wedge x} e$. Similarly, if $e <_{(\neg v^*)\wedge x} e' <_{(\neg v^*)\wedge x} e'' <_{v^*} e$, then by transitivity of $<_{(\neg v^*)\wedge x}$ proved above, we again reach a contradiction

$$e <_{(\neg v^*)\wedge x} e'' <_{v^*} e.$$

This shows that $<_{v^*/x}$ is total and acyclic, and thus a linear order. □

Lemma 3.3: If $e <_s e'$ then $e <_{v^*/x} e'$.

Proof: By definition, there exists a line l hitting both e and e' on their top side, so that its intersection point x with e is to the left of its intersection x' with e', and such that l hits no bottom side between x and x'. Without loss of generality, we can also assume that l hits no top side of a segment between x and x'. Indeed, if l hits the top sides of $e_{j_1}, \ldots, e_{j_k}$ between x and x' in this order from left to right, then it suffices to prove our lemma for each of the pairs $e<e_{j_1}, e_{j_1}<e_{j_2}, \ldots, e_{j_k}<e'$, and then obtain our claim from the transitivity of $<_{v^*/x}$.

If e and e' have overlapping x-projections, then they are related by $<_v$ and it is easy to check that $e <_v e'$, which implies $e <_{v^*/x} e'$. Otherwise we must have $e <_x e'$. If e and e' are not related by $<_{v^*}$, or if $e <_{v^*} e'$, then again $e <_{v^*/x} e'$. Thus we can assume $e' <_{v^*} e$, i.e. we have a chain of the form

$$e' <_v e_{i_1} <_v e_{i_2} \cdots <_v e_{i_k} <_v e ,$$

where again we assume this chain of inequalities to be maximal.

Let π denote the unbounded polygonal curve obtained by the concatenation of the downward-directed vertical ray r emerging from the left endpoint a of e, the segments ax, xx', $x'b'$, where b' is the right endpoint of e', and the upward-directed vertical ray r' emerging from b'. π partitions the plane into two regions R_π, L_π, lying respectively to its right and to its left (as we traverse π from r to r').

We claim that none of the segments $e_{i_1}, \ldots, e_{i_k}$ intersects π. Indeed, none of them can intersect ax, xx', or $x'b'$. If some segment e_{i_p} intersects r, then by definition we have $e <_v e_{i_p}$, implying a cycle in $<_v$ and contradicting Lemma 3.1. A symmetric argument implies that no segment can intersect r'.

As in the proof of Lemma 3.2, let γ be the simple directed polygonal path obtained by following the (maximal) chain of inequalities $e' <_v e_{i_1} <_v \cdots <_v e$. Since the first link of γ lies below e' (thus in R_π) and the last link of γ lies above e (thus in L_π), it follows that γ must intersect π. As argued above, none of the segments e_{i_p} can intersect π. Consider the first intersection z, encountered along γ, of this curve with π. Then z lies along one of the

vertical subsegments of γ, connecting a point $a \in e_{i_p}$ with a point $b \in e_{i_{p+1}}$ for some p, so that a lies above b. But the only portion of π that ab can intersect is the segment xx', and, by assumption, a lies in R_π, hence below xx'. Thus b also lies below xx', so the intersection between ab and xx' is impossible, a contradiction which completes the proof of the lemma. □

Corollary: $<_s$ is a partial order, and $<_{v^*/x}$ is a linear extension of it.

We now turn to describe an algorithm for calculating $<_{v^*/x}$. The idea is quite simple. Using a vertical line-sweep algorithm, it is easy to obtain all $O(n)$ pairs of segments (e,e') related by $<_v$, such that there exists a line l meeting both e and e' and the portion of between these segments does not meet any other segment. Clearly, the collection E of such pairs is a subset of $<_v$ whose transitive closure coincides with $<_{v^*}$. We next apply a topological sorting procedure to E, so as to obtain a linearization of $<_{v^*}$, and adapt it so that the linearization it produces is actually $<_{v^*/x}$. The standard topological sort procedure [Kn] maintains a subset I of the given segments, all of whose predecessors (if any) have already been placed in the output linear order, and then selects at each step an element of I as the next element in the linear order. Note that each pair of segments in I have non-overlapping x-projections. We modify the procedure so that it maintains the set I as a priority queue (ordered by x-coordinate), and always selects from it the smallest element as the next element to be placed on the output list.

It is easy to show, using induction on the order in which the segments are being placed on the output list, that the resulting order coincides with $<_{v^*/x}$. To see the general induction step, consider the current value of I, and let e be the smallest segment (in the x-ordering) in I. It suffices to show that e is the smallest element, in the order $<_{v^*/x}$, among all the still unprocessed segments. Suppose the contrary. Then there exists another unprocessed segment e' such that $e' <_{v^*/x} e$. Without loss of generality we can assume that $e' \in I$, because every unprocessed segment must have a predecessor, in the order $<_{v^*}$, in I. But then $e <_x e'$, so that we must have $e' <_{v^*} e$. But since e' has not yet been processed, e cannot belong to I, a contradiction which proves our claim.

The time complexity of the algorithm is clearly $O(n \log n)$, which arises from the cost of the initial sweeping algorithm, and from the maintenance of the priority queue I in the topological sorting. All other steps take linear time. We thus have in summary

Theorem 3.4: Given a collection of n segments in the plane with disjoint interiors, one can compute, in time $O(n \log n)$, a linear order of them which extends their "rightward shooting order" $<_s$ as defined above.

3.2. The Ray Shooting Procedure, Continued

Let us now return to the special case of our shooting problem. The ordering $<$ has been defined above in more generality than we actually need for this special case, because the rays that we consider now cannot hit the bottom side of any segment in G_w. This fact yields the following simple corollary which is essential to our procedure.

Lemma 3.5: Suppose $e,e' \in G_w$ and $e <_s e'$. If a rightward-directed ray ρ emerges from a point lying above all the lines containing the segments in G_w and hits e, then either ρ does not hit e' or it hits it at a point lying to the right of the intersection of ρ with e.

Proof: If ρ were to intersect e' to the left of its intersection with e, then since both these intersections must hit the top sides of e,e', and since ρ cannot hit the bottom side of any segment in G_w, it follows by definition that we would have $e' <_s e$, a contradiction that

completes the proof. □

We will use this result to obtain a divide-and-conquer strategy, based on repeated applications of the following blocking detection procedure:

> We are given n segments $e_1, \ldots, e_n$ and we want to preprocess them so that, given any query (rightward-directed) ray ρ originating at a point lying above all the lines containing the given segments, we wish to determine quickly whether ρ intersects any of the segments $e_1, \ldots, e_n$.

The solution to this problem is a straightforward application of the implicit point location technique described in Section 2. Specifically, let l denote the line containing the query ray ρ. It is easily verified that ρ intersects a given segment e_i if and only if the line l hits the top side of e_i. Moreover, passing to the dual plane, this condition is equivalent to the condition that the dual point l^* lies in the right wedge W_i formed by the two dual lines to the endpoints of e_i. Thus our blocking detection query is equivalent to testing whether l^* is contained in at least one of these wedges, and, as we have seen in Theorem 2.3, this can be determined in time $O(n^\gamma)$ (for any $\gamma > 2/3$), after $O(n \log n)$ randomized expected preprocessing and using $O(n \log n)$ space.

Having this solution at hand, we solve our shooting problem as follows. Let $e_1 < e_2 < \cdots < e_n$ (where $n = n_w$) be the linear ordering $<_{v^*/x}$ of the segments in G_w. The preprocessing for this problem consists of constructing the above data structure for the blocking detection problem for the first half $\{e_1, \ldots, e_{n/2}\}$ of the segments in G_w, followed by recursive preprocessing of each of the two halves $\{e_1, \ldots, e_{n/2}\}$, $\{e_{n/2+1}, \ldots, e_n\}$ of the set G_w. Clearly, this preprocessing requires $O(n \log^2 n)$ time and uses $O(n \log n)$ space.

However, we can apply once again a space-saving trick akin to that of [DE]. That is, we split $(e_1, \ldots, e_n)$ into n^δ consecutive subsequences, each of size $n^{1-\delta}$, apply the preprocessing step of the blocking detection procedure for each subsequence separately (other than the last one), and then preprocess recursively each subsequence for the shooting problem. Moreover, in each recursive application we split the corresponding subsequence into n^δ subsequences, where n is the size of the original set. In this way, there is only a constant number of recursive levels. It is easily verified that the overall cost of this recursive preprocessing is still $O(n \log n)$, and the total space is $O(n)$.

To process a shooting query involving a (rightward-directed) ray ρ emerging from a point lying above all the lines containing the segments of G_w, we first test, in $O(n^\gamma)$ time, whether ρ is blocked by the first subsequence of G_w. If so, then Lemma 3.5 implies that we can ignore in this case the remaining subsequences of G_w, and we thus continue recursively to query the first subsequence. Otherwise, we apply the blocking detection procedure to the second subsequence of G_w, and continue this way until we find a subsequence that blocks ρ, and then continue the query recursively within that subsequence. At the end of this process, we reach a subset of G_w consisting of a single segment, which we return as the output to our shooting query. The overall query time is

$$O(n^{\gamma+\delta} + (\frac{n}{2})^{\gamma+\delta} + (\frac{n}{4})^{\gamma+\delta} + \cdots) = O(n^{\gamma+\delta})$$

which can also be written as $O(n^\gamma)$ for another, yet still arbitrarily small, $\gamma > 2/3$.

Our final stroke is to apply the space-saving trick of [DE] once again, by preprocessing the sets G_w just once every h levels, where h is chosen appropriately as above. The above arguments imply

Theorem 3.6: Given a polygonal region P with n edges, one can preprocess it in time $O(n \log n)$ into a data structure of size $O(n)$, using which one can answer ray shooting queries in time $O(n^{\gamma})$, for any $\gamma > 2/3$.

3.3. Ray Shooting in an Arbitrary Arrangement of Segments

The preceding techniques can be generalized to handle the more difficult case of ray shooting in an arrangement of an arbitrary collection of (possibly intersecting) line segments. Let $\{e_1, \ldots, e_n\}$ be such a collection. We prepare a data structure similar to the one used above. At the top level we use the same partition tree T as above. In answering a shooting query, the processing of leaves of T, and the merging of query outputs at children of a node v of T to obtain the output at v itself, are the same as above. The part that needs modification is in the processing of a node w of T that the dual line q^* (where q is the origin of the query ray) has missed.

Assume without loss of generality that q lies above all the lines containing the segments in G_w. Let R denote the intersection of the upper half-planes bounded below by these lines. Clearly $q \in R$ and the interior of R is disjoint from any segment in G_w. Let F denote the face of the arrangement of the segments in G_w which contains R (if all segments in G_w are bounded, F is actually the (unique) unbounded face of the arrangement). As shown in [PSS], [EGSh], the boundary of F consists of $O(n_w \alpha(n_w))$ subsegments, and can be calculated in time $O(n_w \alpha(n_w) \log^2 n_w)$. Once F is available, the remainder of the construction proceeds exactly as above, except that this time it is applied to the $O(n_w \alpha(n_w))$ subsegments (which have pairwise disjoint relative interiors) forming the boundary of F.

Embedding the cost of this additional preprocessing into that of the entire algorithm, we easily obtain

Theorem 3.7: Given a collection of n segments in the plane, one can preprocess it in time $O(n\alpha(n) \log n)$ 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 $O(n^{\gamma})$, for any $\gamma > 2/3$.

4. Implicit Hidden Surface Removal Algorithms

As our next application, we study certain cases of the hidden surface removal problem, which can be stated as follows. Given a collection of objects in 3-dimensional space, and a viewing point a, we wish to calculate the scene obtained by viewing these objects from a. In other words, we want to produce some representation of this scene, in which we can determine, for each ray emerging from a, the first surface of an object hit by that ray (or determine that the ray does not his any object). For the sake of simplicity, we will assume that the given objects are all polyhedral, whose boundaries consist of n triangular faces altogether, and that they do not intersect one another. This problem can be easily formulated in the context of implicit point location, as follows. The given triangular faces are projected towards a onto a certain "plane of view", and the arrangement of their projections is calculated. To each face of this arrangement one then assigns the triangle nearest to a whose projection contains that face, and this augmented arrangement constitutes the desired solution to the hidden surface removal problem. To obtain an actual image for display purposes, one can then take each pixel in the plane of view, and locate the face of the arrangement containing that pixel, thereby obtaining the object seen at that pixel. 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. We will use the implicit algorithms of Section 2 to obtain improved solutions. This is not the first instance of an implicit solution to this problem. It was shown recently in [CS] that if the objects to be viewed form a polyhedral terrain (i.e. a surface cut by each vertical line in exactly one point) then we can obtain an implicit representation of its image in preprocessing time and space $O(n\alpha(n)\log n)$ (where $\alpha(n)$ is the extremely slowly growing inverse Ackermann's function), and query time $O(\log n)$.

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 $z=i$ (or, more generally, Δ_i lies in a plane $z=a_i$ where $0<a_1<\cdots<a_n$). 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 hidden surface removal problem in which such an ordering exists.

We also consider the following simpler problem. Given the same set-up as above, we want to determine, for any point p in the xy-plane, whether p is *blocked* by the triangles $\Delta_1, \ldots, \Delta_n$, meaning that the vertical line through p intersects at least one of these triangles. In other words, we want to determine whether p lies in the union of the projections $\Delta_1^*, \ldots, \Delta_n^*$ of the given triangles onto the xy-plane. For this blocking problem, no order among the given triangles need be assumed. In fact, it applies to an arbitrary collection of (even intersecting) triangles in 3-space. By Theorem 2.3, we immediately obtain

Theorem 4.1: (i) The preprocessing version of the visibility blocking problem for a collection of n triangles in 3-space can be solved in $O(n\log n)$ preprocessing time, $O(n)$ space, and $O(n^\gamma)$ query time, for any $\gamma>2/3$.

(ii) The batched version of this problem for n triangles and m points, can be solved in time

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

for any $\delta>0$.

Remark: We leave it as an open problem whether the results of Section 2 can be applied directly to solve the original hidden surface elimination problem. The difficulty we face is that while membership of a query point p in a projected triangle is obtained by adding edge functions, finding the lowest triangle above p involves calculating the minimum of the indices of the triangles lying above p. This mixture of addition and minimum operations does not seem to be amenable to the techniques of Section 2, which tend to "scramble" the order in which these operations are applied.

To solve our original hidden surface removal problem, we continue in much the same way as in Section 3. Consider first the preprocessing version. We split the set T of triangles into two subsets T_1, T_2, so that $T_1=\{\Delta_1, \ldots, \Delta_{n/2}\}$ contains the lower half of the triangles, and $T_2=\{\Delta_{n/2+1}, \ldots, \Delta_n\}$ contains the upper half of the triangles. We apply the blocking detection preprocessing algorithm to the triangles in T_1, and then proceed recursively with preprocessing T_1 and T_2 separately. To process a query point p, we first test whether T_1 blocks p. If so, we continue to process the query within the structure of T_1; otherwise we continue with the structure of T_2. Thus in $\log n$ such steps we find the lowest triangle blocking p (if one exists). This approach uses $O(n\log^2 n)$ preprocessing time and $O(n\log n)$

space, and has query time $O(n^\gamma)$ for any $\gamma>2/3$. Again, the trick of [DE] can be used to reduce the preprocessing time and space to $O(n \log n)$ and $O(n)$, respectively.

The batched version is handled similarly. We split the set T into two subsets T_1, T_2 as above, and then apply the batched version of the blocking detection procedure to the m given points and to T_1. Suppose m_1 of the points are found to be blocked by T_1, and $m_2 = m - m_1$ are not blocked. We then calculate recursively the lowest triangle in T_1 above each of the first m_1 points, and similarly calculate the lowest triangle in T_2 above each of the remaining m_2 points. If during the recursion the number of points or the number of triangles becomes sufficiently small (e.g. ≤ 1), then we solve the problem by brute force, in maximum time $O(m+n)$.

Hence, if $T(m,n)$ denotes the maximum time to solve our problem for m points and n triangles, we have

$$T(m,n) = O(m+n), \quad \text{if } m\leq 1 \text{ or } n\leq 1$$

$$T(m,n) = \max_{m_1+m_2=m} \left[T(m_1,\frac{n}{2}) + T(m_2,\frac{n}{2})\right] + Q(m,n), \quad \text{otherwise}$$

where

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

for any $\delta>0$.

Proposition 4.2:

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

for any $\delta>0$.

Proof: It is clear that the contribution of the second and third terms of $Q(m,n)$ to $T(m,n)$ is bounded by the second and third terms of this expression. As to the first term, the inequality is an immediate consequence of

$$m_1^{2/3-\delta}(\frac{n}{2})^{2/3+2\delta} + m_2^{2/3-\delta}(\frac{n}{2})^{2/3+2\delta} \leq (m_1 + m_2)^{2/3-\delta}n^{2/3+2\delta}.\frac{2^{1/3+\delta}}{2^{2/3+2\delta}}$$

$$< m^{2/3-\delta}n^{2/3+2\delta}$$

□

Theorem 4.3: (i) The preprocessing version of the hidden surface removal problem for an ordered collection of n triangles in 3-space can be solved in $O(n \log n)$ randomized expected preprocessing time, $O(n)$ space, and $O(n^\gamma)$ query time, for any $\gamma>2/3$.

(ii) The batched version of this problem for n triangles and m points, can be solved in randomized expected time

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

for any $\delta>0$.

Remark: Recently and independently, Muller [Mu], [SML] has studied other variants of the ray shooting problem in three dimensions. In particular, he has looked at cases where the given objects are all iso-oriented horizontal rectangles, but the query ray can be in arbitrary directions. He has applied somewhat similar partitioning techniques, and obtained similar, though somewhat worse, performance bounds. This is due to the following two reasons, both

of which can be rectified using our approach. (i) He has used the conjugation tree technique of [EW] rather than the ϵ-net approach of [HW]; and (ii) In the batched version, he has used standard, non-customized trees.

5. Polygon Placement Queries

Consider next the following final application of our techniques. Let P be any polygonal object (not necessarily simply connected) having k sides, and let $\Delta_1, \ldots, \Delta_n$ be a collection of (possibly intersecting) triangles in the plane. We wish to preprocess the data so that, given any query placement of (a fixed reference point within) P, we can quickly determine whether at this placement P hits any of the triangles, or is otherwise "free" (we assume that P is kept at its initial orientation). Problems of this sort arise in a variety of contexts (see Chazelle [Ch]). If P is convex (and the "obstacles" do not intersect each other), then there exist efficient solutions to the problem ([CD], [Fo], [LS], [KLPS], [BZ]), but the general case has no comparably efficient solutions. The general approach to this problem is to form the Minkowski differences

$$K_i = \Delta_i - P = \{x-y:\ x \in \Delta_i,\ y \in P\}$$

(where $x-y$ denotes vector subtraction), and then calculate their union

$$K = \bigcup_{i=1}^{n} K_i .$$

Assuming the reference point on the given initial placement of P is the origin, then a placement of P with that reference point at some point z (we say in short, placement of P at z) is free, if and only if $z \notin K$.

Now if we triangulate P into $O(k)$ triangles, say $P_1, \ldots, P_k$, we can represent K as the union of $O(kn)$ Minkowski differences of the form $K_{ij} = \Delta_i - P_j$, each of which is a convex l-gon, for some $l \leq 6$. Thus our blocking detection procedure can be applied to these K_{ij}, to yield the following result.

Theorem 5.1: With preprocessing time $O(kn \log kn)$ and space $O(kn)$, one can obtain a data-structure, using which one can determine, for a given query placement of P, whether at this placement P is free or not, in time $O((kn)^{\gamma})$ for any $\gamma > 2/3$.

Remark: We can also obtain efficient solutions to the batched version of the problem, as above.

How significant is this result? If the obstacle triangles Δ_i intersect one another, it seems that our implicit approach is superior to any explicit alternative technique. However, suppose the triangles Δ_i have pairwise disjoint interiors. Then, if P is convex, the previously mentioned results [CD], [Fo], [LS], [KLPS], [BZ] show that the complexity of K is only $O(kn)$, and that it can be calculated in a variety of efficient techniques, the best being of time complexity $O(kn \log kn)$ [LS]. Using standard point location techniques, we can then process each placement query in time $O(\log kn)$.

Now, this observation yields an alternative approach to the case of a general P. Namely, we decompose P into triangles $P_1, \ldots, P_k$, and then apply the above techniques to each triangle P_i separately. Given a query placement z of P, we locate z in each of the k maps obtained for the individual parts P_i of P. Then z is free if and only if it is free in each of these maps. This gives us preprocessing time $O(kn \log n)$, space $O(kn)$, and query time $O(k \log n)$.

Comparing this query time to our solution, it is easily checked that our method becomes superior when $k>n^2$.

Remark: Both approaches, that of the k separate queries and our implicit point location approach, also apply to cases where P is defined as the union of k arbitrary triangles or line segments. Thus even in cases of this kind where the "explicit" complexity of P is $O(k^2)$ (e.g. P is a $k\times k$ grid of horizontal and vertical line segments), we can determine whether a placement of P is free or not within the same bounds noted above.

6. Conclusion

The main purpose of this paper has been to point out the versatility and rather wide applicability of the partition tree technique. We have demonstrated this by developing a general technique for "implicit" point location in arrangements of overlapping polygonal objects, and by three other applications to ray shooting and containment problems. We believe that there are many additional potential applications of this method.

References

[Ag] P.K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, to appear.

[BZ] B.K. Bhattacharya and J. Zorbas, Solving the two-dimensional findpath problem using a line-triangle representation of the robot, *J. Algorithms* 9 (1988), pp. 449-469.

[Ch] B. Chazelle, The polygon containment problem, in *Advances in Computing Research, Vol. I: Computational Geometry*, (F.P. Preparata, Ed.), JAI Press, Greenwich, Connecticut (1983), pp. 1-33.

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

[CW] B. Chazelle and E. Welzl, Range searching and VC dimension: A characterization of efficiency, manuscript, 1988.

[CD] L.P. Chew and R.L. Drysdale, Voronoi Diagrams Based on Convex Distance Functions, *Proc. ACM Symp. on Computational Geometry*, 1985, pp. 235-244.

[CS] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symbolic Computation* 7 (1989), pp. 11-30.

[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. Algorithms* 8 (1987) pp. 348-361.

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

[EGSh] H. Edelsbrunner, L. Guibas and M. Sharir, The complexity of many faces in arrangements of lines or of segments, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 44-55.

[EGS] H. Edelsbrunner, L. Guibas and J. Stolfi, Optimal point location in monotone subdivisions, *SIAM J. Computing* 15 (1986), pp. 317-340.

[EOS] H. Edelsbrunner, J. O'Rourke and R. Seidel, Constructing arrangements of lines and hyperplanes with applications, *SIAM J. Comput.* 15 (1986), pp. 341-363.

[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.

[EW2] H. Edelsbrunner and E. Welzl, Private communication, 1987.

[Fo] S. Fortune, Fast Algorithms for Polygon Containment, *Proc. 12th International Colloquium on Automata, Language and Programming*, Lecture Notes in Comp. Science 194, Springer-Verlag, New York, 1985, pp. 189-198.

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

[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.

[Ki] D. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Computing* 12 (1983) pp. 28-35..

[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.

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

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

[Mu] H. Muller, manuscript, 1987.

[PSS] R. Pollack, M. Sharir and S. Sifrony, Separating two simple polygons by a sequence of translations, *Discrete and Computational Geometry* 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, manuscript 1987.

[ST] N. Sarnak and R.E. Tarjan, Planar point location using persistent search trees, *Comm. ACM* 29 (1986) pp. 669-679.

[SML] A. Schmitt, H. Muller and W. Leister, Ray tracing algorithms - Theory and practice, in *Theoretical Foundations of Computer Graphics and CAD*, Nato ASI Series, Vol. F-40, Springer Verlag, 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.