SIAM J. COMPUT.
Vol. 21, No. 3, pp. 540–570, June 1992

RAY SHOOTING AND OTHER APPLICATIONS OF SPANNING TREES WITH LOW STABBING NUMBER*

PANKAJ K. AGARWAL†

Abstract. This paper considers the following problem: Given a set $\mathcal{G}$ of n (possibly intersecting) line segments in the plane, preprocess it so that, given a query ray ρ emanating from a point p, one can quickly compute the intersection point $\Phi(\mathcal{G}, \rho)$ of ρ with a segment of $\mathcal{G}$ that lies nearest to p. The paper presents an algorithm that preprocesses $\mathcal{G}$, in time $O(n^{3/2} \log^{\omega} n)$, into a data structure of size $O(n\alpha(n) \log^4 n)$, so that for a query ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in time $O(\sqrt{n\alpha(n)} \log^2 n)$, where ω is a constant < 4.33 and $\alpha(n)$ is a functional inverse of Ackermann's function. If the given segments are nonintersecting, the storage goes down to $O(n \log^3 n)$ and the query time becomes $O(\sqrt{n} \log^2 n)$. The main tool used is spanning trees (on the set of segment endpoints) with low stabbing number, i.e., with the property that no line intersects more than $O(\sqrt{n})$ edges of the tree. Such trees make it possible to obtain faster algorithms for several other problems, including implicit point location, polygon containment, and implicit hidden surface removal.

Key words. arrangements, fractional cascading, point location, ray shooting, spanning tree, stabbing number, zone

AMS(MOS) subject classifications. 52A37, 68Q20, 68Q25, 68R99

1. Introduction. In the last few years many efficient randomized algorithms, based on the random sampling techniques of [Cl] or on the related ϵ-net theory [HW], have been developed to solve efficiently a variety of geometric problems. One such recent development is due to Welzl [We] (see also [CW]), who showed that, for a given set S of n points in the plane, there exists a spanning tree T of S, such that no line intersects more than $O(\sqrt{n} \log n)$ edges of T. Such a tree T is called a spanning tree with *low stabbing number* (a formal definition is given in §2). Welzl used spanning trees with low stabbing number to obtain an almost optimal algorithm for *simplex range searching*, namely, given a set S of n points in the plane, preprocess it into a data structure of linear size so that, for a query triangle $\triangle$, one can quickly count (or more generally report) all points of S lying inside $\triangle$. His algorithm counts (respectively, reports) the points lying inside a query triangle $\triangle$ in time $O(\sqrt{n} \log^2 n)$ (respectively, $O(\sqrt{n} \log^2 n + K)$, where K is the number of points inside $\triangle$). Soon after this paper, Edelsbrunner et al. [EGH*] used these trees to preprocess a given set $\mathcal{L}$ of n lines in the plane into a data structure of size $O(n \log n)$ so that, for a query point p, the face of the arrangement $\mathcal{A}(\mathcal{L})$ containing p can be computed quickly. The main challenge in both of these papers was to use only roughly linear space (i.e., $O(n \log^{O(1)} n)$ space), because if we allow quadratic space, then a query can be easily answered in $O(\log n)$ time [Ed], [EOS], [EG].

In this paper we present several new applications of spanning trees with low stabbing number. The algorithms presented in this paper are faster than the previously best known algorithms for these problems. One of the main goals of this paper is to demonstrate that such a spanning tree is a versatile tool that can be applied to obtain efficient algorithms for a large class of problems, much beyond the simplex range searching problem for which they were originally introduced. We also show that by combining the spanning tree data structure with the recent partitioning algorithm of [Aga] and [Agb], we can

*Received by the editors June 12, 1989; accepted for publication (in revised form) July 16, 1991. Work on this paper was done while the author was at the Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. This work has been supported by Office of Naval Research grant N00014-87-K-0129, by National Science Foundation grant DCR-83-20085, and by grants from the Digital Equipment Corporation and the IBM Corporation.

† Computer Science Department, Duke University, Durham, North Carolina 27706.

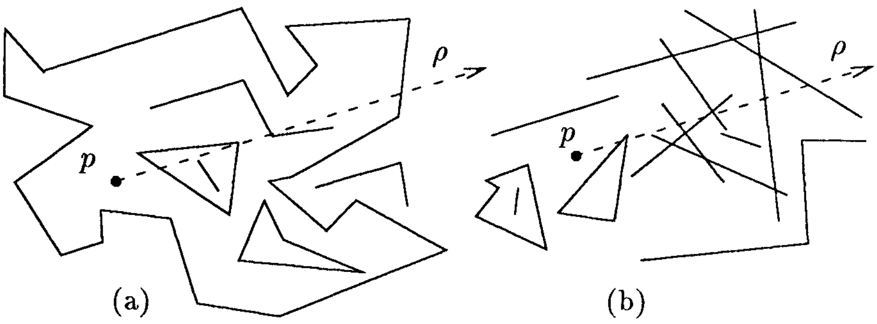

FIG. 1. *Ray shooting in an arrangement of* (a) *nonintersecting and* (b) *arbitrary segments.*

obtain a trade-off between space and query time. Similar trade-offs have been obtained earlier [EGH*], [Agc], [Chd].

The first and perhaps the most interesting application that we consider is *ray shooting* in arrangements of segments. There are two versions of this problem, one for segments that are nonintersecting, and one for an arbitrary collection of segments. Formally, these problems can be stated as follows:

(a) *Given a collection $\mathcal{G} = \{e_1, \cdots, e_n\}$ of n nonintersecting line segments in the plane, preprocess it so that, given a query ray ρ emanating from a point p in direction d, we can quickly compute the intersection point $\Phi(\mathcal{G}, \rho)$ of ρ with the segments of $\mathcal{G}$ that lies nearest to p (see Fig.* 1(a)).

(b) *Same problem, except that the segments in $\mathcal{G}$ can intersect arbitrarily (see Fig.* 1(b)).

If the segments in $\mathcal{G}$ form the boundary of a simply connected region, then the algorithm of Chazelle and Guibas [CGa] preprocesses $\mathcal{G}$ into a data structure of linear size so that, for any ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time (see also [GHLST]). For the general case, however, the ray shooting and other visibility problems are much harder even for nonintersecting segments. For example, a result of Suri and O'Rourke [SO] shows that the portion of a polygon, with holes, visible from a fixed edge can have $\Omega(n^4)$ edges on its boundary, while for simple polygons such a region is bounded by only $O(n)$ edges.

We are not aware of any ray shooting algorithm for nonsimple polygons (or for an arrangement of segments), which answers a query in $O(\log^{O(1)} n)$ time, using roughly linear space. If we allow quadratic space, then a query is easy to answer in time $O(\log n)$ (see §4.1). Our goal in this paper is to obtain efficient solutions that use roughly linear space, and to establish a trade-off between space and query time.

For a special case, where $\mathcal{G}$ is a set of lines, a result of Edelsbrunner et al. [EGH*] implies that we can construct, in randomized expected time $O(n^{3/2} \log^2 n)$, a data structure of size $O(n \log^2 n)$, so that a ray shooting query in $\mathcal{A}(\mathcal{L})$ can be answered in $O(\sqrt{n} \log^3 n)$ time. (The preprocessing has been made deterministic and the query time has been reduced to $O(\sqrt{n} \log n)$ in [Agc].) Unfortunately, this algorithm does not apply to segments. An algorithm with a sublinear query time for the case of segments can be developed using the "recursive space-cutting tree" of Dobkin and Edelsbrunner [DE] (see also [EW]). The best-known algorithm for computing $\Phi(\mathcal{G}, \rho)$ is by Guibas et al. [GOS], which constructs a data structure of size $O(n)$, so that a query can be answered in

$O(n^{2/3+\delta})$ time, for any $\delta > 0$. Their algorithm is based on the random sampling technique of [Cl] and [HW], and constructs a multilevel partition tree. The preprocessing of their algorithm is randomized with $O(n \log n)$ expected running time. However, the preprocessing can be made deterministic without any additional overhead using the recent partitioning algorithms of Matoušek [Maa] or Agarwal [Agb].

In this paper we show that ray shooting can be performed in roughly (that is, up to polylogarithmic factors) $\sqrt{n}$ time, while still using only roughly linear space and employing deterministic, rather than randomized, preprocessing techniques. We first give an algorithm for the case of nonintersecting segments. This algorithm constructs, in time $O(n^{3/2} \log^{\omega} n)$, a data structure of size $O(n \log^3 n)$ so that, for a given ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\sqrt{n} \log^2 n)$ time, where ω is a constant less than 4.33. Our algorithm is simpler than that of [GOS] because it maintains only a two-level data structure. We then extend the above algorithm to general arrangements of segments. Although the basic idea remains the same, we need several new techniques, and the algorithm is more complex. In this case a query can be answered in $O(\sqrt{n\alpha(n)} \log^2 n)$ time, using $O(n\alpha(n) \log^4 n)$ space, after $O(n^{3/2} \log^{\omega} n)$ preprocessing. Another major difference between the two cases is that in the first case we can report all K intersections between a query ray ρ and $\mathcal{G}$ in $O(\sqrt{n} \log^2 n + K \log n)$ time, while we still do not know how to report these intersections in a comparably efficient manner in the general case. One disadvantage of our algorithms over those of [GOS] and [DE] is that our preprocessing time is roughly $n^{3/2}$ instead of roughly linear. This is the price that we must pay to achieve deterministic preprocessing and to reduce the query time.

The second problem for which we give an efficient algorithm using the spanning tree data structure is *implicit point location*. The implicit point location problem is an extension of the widely studied planar point location problem (see [Ki], [EGS], and [ST]). In the latter problem, a planar map M consisting of n faces is given, and the goal is to preprocess M into a data structure that supports fast point location queries, i.e., queries that seek the face of M containing a query point p. The above algorithms construct, in time $O(n \log n)$ (or sometimes linear), a data structure of linear size, so that a query point can be located in M in $O(\log n)$ time. In the implicit point location problem the map is defined implicitly. In particular, we assume that it is defined as the arrangement (i.e., overlay) of a given set of n geometric polygonal (possibly intersecting) objects of some simple shape (or as a collection of arbitrary line segments), and the goal is to obtain certain information related to the arrangement of the objects; for example, to determine whether a query point lies in the union of the objects. A more formal description is given in §7. Guibas et al. [GOS] have presented an algorithm with $O(n^{2/3+\delta})$ query time, for any $\delta > 0$, using the random sampling technique. We improve the query time to $O(\sqrt{n} \log^2 n)$ and use deterministic preprocessing. The algorithm of [GOS] uses $O(n)$ space, while ours requires $O(n \log^2 n)$ space.

Guibas et al. [GOS] have described several applications of the implicit point location problem, such as polygon containment, implicit hidden surface removal, polygon placement, etc. We show that our implicit point location algorithm improves the query time of these algorithms too.

This paper is organized as follows. In §2 we discuss spanning trees with low stabbing number. Section 3 describes our ray shooting algorithm for arrangements of nonintersecting segments. In §4 we show that ray shooting queries can be performed faster, if we are allowed to use more space. Section 5 extends the algorithms of §§3 and 4 to report all intersections between $\mathcal{G}$ and a query ray ρ at logarithmic cost per intersection. In §6 we generalize our ray shooting algorithms to arrangements of arbitrary (possibly intersecting) segments. Section 7 presents an efficient algorithm for implicit point location and §8 discusses other applications of the spanning tree data structure. We conclude in §9 with some final remarks.

2. Spanning trees of low stabbing number. Let S be a set of n points in $\mathbb{R}^d$, and let $\mathcal{T}$ be a spanning tree of S whose edges are line segments. The *stabbing number* $\sigma(\mathcal{T})$ of $\mathcal{T}$ is the maximum number of edges of $\mathcal{T}$ that can be intersected by a hyperplane h. Chazelle and Welzl [CW] (see also [Chc], [We]) have proved that, for any set of n points in $\mathbb{R}^d$, there exists a spanning tree with stabbing number $O(n^{1-1/d})$, and that this bound is tight in the worst case. For a family $\mathbf{T}$ of trees, the stabbing number $\sigma(\mathbf{T})$ is s if for each hyperplane h there is a tree $\mathcal{T} \in \mathbf{T}$ such that h intersects at most s edges of $\mathcal{T}$.

Chazelle and Welzl [CW] also proved that a spanning tree of n points in $\mathbb{R}^d$ with stabbing number $O(n^{1-1/d})$ can be constructed in polynomial time. In the plane, a spanning tree with stabbing number $O(\sqrt{n})$ can be constructed in $O(n^3 \log n)$ time. A recent algorithm of Matoušek [Mab] improves the running time to $O(n^{5/2} \log^2 n)$ at the cost of increasing $\sigma(\mathcal{T})$ to $O(\sqrt{n} \log n)$. As for constructing a family of spanning trees, Edelsbrunner et al. [EGH*] have presented a randomized algorithm, with expected running time $O(n^{3/2} \log^2 n)$, to compute a family $\mathbf{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_k\}$ of $O(\log n)$ spanning trees, with $\sigma(\mathbf{T}) = O(\sqrt{n} \log^2 n)$. The running time of their algorithm has been improved to $O(n^{4/3} \log^2 n)$ in another randomized algorithm by Matoušek [Mab]. (The stabbing number of $\mathbf{T}$ computed by Matoušek's algorithm can actually be improved to $O(\sqrt{n} \log n)$; see [Agc].) An additional property of the algorithms of [EGH*] and [Mab] is that the trees they produce are actually spanning paths. The best known deterministic algorithm for constructing a family of spanning path is due to Agarwal [Agc], who has shown the following.

THEOREM 2.1. [Agc] *Given a set S of n points in the plane, we can deterministically construct a family $\mathbf{C}$ of $O(\log n)$ spanning paths on S with $\sigma(\mathbf{C}) = O(\sqrt{n})$, in $O(n^{3/2} \log^{\omega} n)$ time, using $O(n^{3/2})$ working storage, where ω is a constant less than 4.33. Moreover, for any query line ℓ, we can determine in $O(\log n)$ time a spanning path $C \in \mathbf{C}$ such that ℓ intersects at most $O(\sqrt{n})$ edges of C.*

The paths constructed by [Mab] and [Agc] can generally be self-intersecting. However, Edelsbrunner et al. [EGH*] have shown that a spanning tree $\mathcal{T}$ can be converted into a simple polygonal path C in $O(n \log n)$ time, so that if a line ℓ intersects s edges of $\mathcal{T}$, then ℓ intersects at most $2s$ edges of C. Therefore, if desired, we can assume that the spanning paths produced by the techniques of [Mab] and [Agc] are non–self-intersecting.

Let C be a spanning path on S. For our applications we need to construct a balanced binary tree $\mathcal{B}$ on C whose leaves store the points of S in their order along C. Each node v of $\mathcal{B}$ is associated with the subpath C_v of C connecting the points stored at the leaves of the subtree rooted at v; let us denote by S_v the subset of S consisting of these points (see Fig. 2).

A line ℓ *stabs* a node v of $\mathcal{B}$ if ℓ intersects C_v. Let $V_{\mathcal{B}}(\ell)$ denote the set of nodes v of $\mathcal{B}$ such that v is not stabbed by ℓ but its parent (if one exists) is stabbed. It is easily seen that $\{S_v : v \in V_{\mathcal{B}}(\ell)\}$ is a disjoint partitioning of S. Moreover, we have the following lemma.

LEMMA 2.2. *If a line ℓ intersects s edges of C, then $|V_{\mathcal{B}}(\ell)| \leq 2(s+1)\log n$ and the nodes of $V_{\mathcal{B}}(\ell)$ lie on at most $2(s+1)$ paths of $\mathcal{B}$.*

Another simple but key observation is given in the following lemma.

LEMMA 2.3. *A line ℓ intersects a polygonal path C if and only if ℓ intersects the convex hull of the vertices of C.*

Lemma 2.3 implies that ℓ stabs a node v if and only if ℓ intersects the convex hull

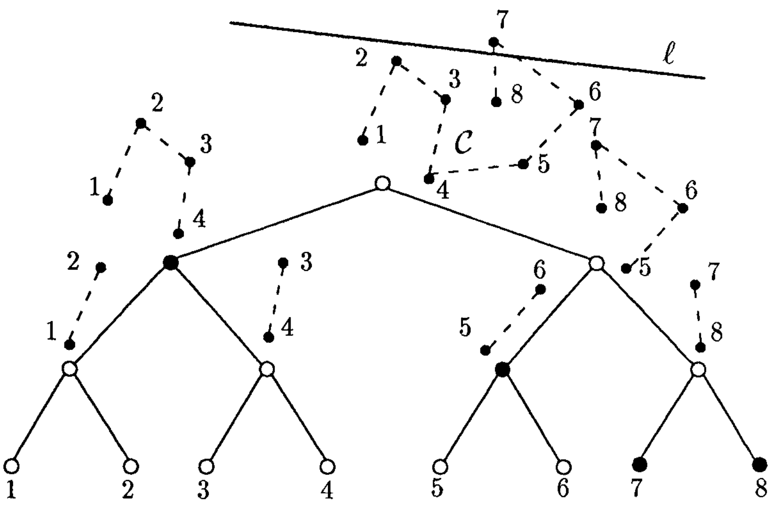

FIG. 2. *$\mathcal{C}$ and $\mathcal{B}(\mathcal{C})$: black nodes of $\mathcal{B}$ denote $V_{\mathcal{B}}(\ell)$.*

of S_v. Since an intersection between a line and a convex n-gon can be detected in $O(\log n)$ time, it follows that $V_{\mathcal{B}}(\ell)$ and the paths containing its nodes can be computed in $O(|V_{\mathcal{B}}(\ell)| \log n)$ time, if we store the convex hull of the subpath $\mathcal{C}_v$ at each node v of $\mathcal{B}$. The running time of this computation can actually be improved to time $O(|V_{\mathcal{B}}(\ell)| + \log n)$, using fractional cascading (cf. [CGc]).

All the problems considered in this paper involve a set of segments in $\mathbb{R}^2$ and most of the algorithms presented here are based on spanning paths with low stabbing number. The spanning path is constructed either on the endpoints of the segments or on the points dual to the lines containing the segments. To answer a query, we choose a line ℓ depending on the query and the problem (e.g., in the ray shooting problem, we take ℓ to be the line containing the query ray), and compute the intersection points of ℓ and the spanning path. The portion π of the spanning path between two consecutive intersection points lies either above or below ℓ. The query for segments corresponding to the points lying on π is answered directly in $O(\log^{O(1)} n)$ time, see below for details. We repeat this procedure for all such portions of the spanning path and then compute the overall answer from them. If ℓ intersects s edges of the path, the query time is $O(s \log^{O(1)} n)$. Since $s = O(\sqrt{n})$, the query time of these algorithms will be $O(\sqrt{n} \log^{O(1)} n)$.

3. Ray shooting in arrangements of nonintersecting segments. In this section we present an algorithm that preprocesses a given set $\mathcal{G}$ of n nonintersecting segments so that, given a query ray ρ emanating from a point p in direction d, $\Phi(\mathcal{G}, \rho)$ can be computed quickly. (For technical reasons we consider ρ as an open ray, i.e., the point p does not belong to ρ.) We will also use $\Phi(\mathcal{G}, \rho)$ to denote the distance of that point from p; if no such intersection exists, we put $\Phi(\mathcal{G}, \rho) = +\infty$. Without loss of generality, we restrict our attention to rightward-directed rays; leftward-directed rays can be handled in a symmetric way. We also assume that there is no vertical segment in $\mathcal{G}$. Denote the set of left endpoints of the segments of $\mathcal{G}$ as $S = \{p_1, \cdots, p_m\}$, where $m \leq n$. Let $\mathbf{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_k\}$ denote a family of $k = O(\log n)$ spanning paths on S, with $\sigma(\mathbf{C}) = O(\sqrt{n})$.

We show how to preprocess a single path $\mathcal{C} \in \mathbf{C}$. First, construct the binary tree $\mathcal{B} = \mathcal{B}(\mathcal{C})$. Let $\mathcal{G}_v \subseteq \mathcal{G}$ be the set of segments whose left endpoints are in S_v (see Fig. 3).

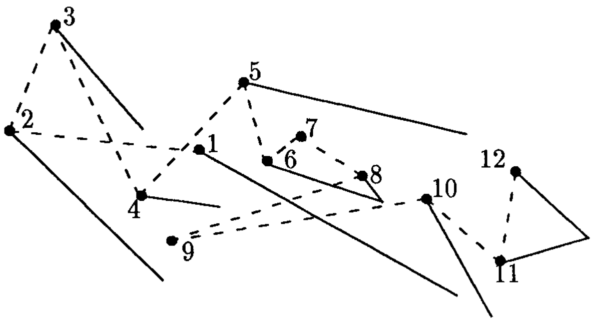

FIG. 3. $\mathcal{G}_v$: *dashed path denotes* C_v; *solid lines denote* $\mathcal{G}_v$; *bullets denote* S_v.

Let ℓ denote the line containing the query ray ρ; then

$$\Phi(\mathcal{G}, \rho) = \min_{v \in V_{\mathcal{B}}(\ell)} \{\Phi(\mathcal{G}_v, \rho)\}. \tag{3.1}$$

Note that for a node $v \in V_{\mathcal{B}}(\ell)$ C_v is a connected path; therefore, either all points in S_v lie above ℓ or all of them lie below ℓ. In what follows we assume that all points of S_v lie above ℓ. We will show below that $\Phi(\mathcal{G}_v, \rho)$ can be computed in $O(\log n)$ time. First, a few notations: Let ℓ^- (respectively, ℓ^+) denote the half plane lying below (respectively, above) the line ℓ. We distinguish between the two sides of a segment e, the top (respectively, bottom) side of e is denoted by e^+ (respectively, e^-). We say that a ray ρ hits e from *above* (respectively, *below*) if slightly to the left of their intersection, ρ lies above (respectively, below) e. If we think of e as expanded into a very thin rectangle and of e^+, e^- as denoting the top and bottom sides of that rectangle, respectively, then ρ hits e from above if, when traversed from left to right, ρ first intersects e^+ and then e^-, and symmetrically for rays that hit e from below (see Fig. 4). If ρ hits e from above (respectively, below), then we also say that it hits e^+ (respectively, e^-). The following lemma is quite obvious, so we state it without proof.

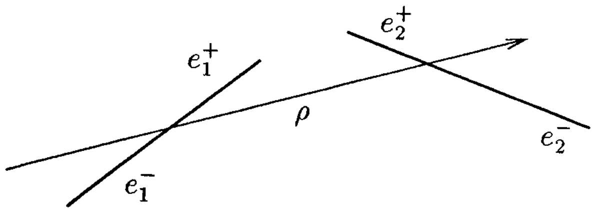

FIG. 4. *Two-sided segments:* ρ *hits* e_1 *from above and* e_2 *from below.*

LEMMA 3.1. *Let v be a node of $V_{\mathcal{B}}(\ell)$. Under the assumption that all points of S_v lie above ℓ, if ρ hits a segment $e \in \mathcal{G}_v$, then it hits e from below.*

Before proceeding, we introduce a linear ordering among the segments of $\mathcal{G}_v$, as defined in [GOS] (see also [GY]). As we will see later, this ordering sorts the segments in a manner that is consistent with any order in which they can be crossed by a rightward-directed ray (from below).

DEFINITION 3.2 [GOS]. For a given set $\mathcal{G} = \{e_1, \cdots, e_n\}$ of segments,

(i) $e_i <_s e_j$ if there exists a (nonvertical) line ℓ hitting both e_i^- and e_j^- such that its intersection with e_i lies to the left of its intersection with e_j, and such that ℓ does not hit any e_k^+, for $k \neq i, j$, at a point between e_i and e_j.

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

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

(iv) $e_i < e_j$ 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 $e_i <_x e_j$.

THEOREM 3.3 [GOS]. *$<_s$ is a partial order, and $<$ is a linear order that extends $<_s$. Moreover $<$ can be computed in time $O(n \log n)$.*

Remark 3.4. It is possible for a pair of segments e_1, e_2 that $e_2 < e_1$ within some set $\mathcal{G}$, but $e_1 < e_2$ relative to a subset $\mathcal{G}' \subset \mathcal{G}$ (see Fig. 5). Therefore, it is important to mention the set relative to which we are ordering the segments.

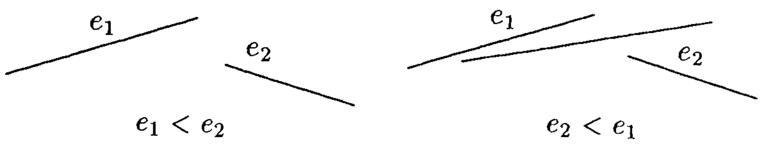

FIG. 5. *Ordering of a pair of segments is relative to a set.*

Next we prove a technical lemma about $<$ that we will need later. Let l_e (respectively, r_e) denote the left (respectively, right) endpoint of a nonvertical segment e.

LEMMA 3.5. *For all segments $a, b \in \mathcal{G}_v$, if r_a lies below ℓ and $x(r_a) < x(l_b)$, then $a < b$ (relative to $\mathcal{G}_v$).*

Proof. Suppose, to the contrary that, there is a pair of segments $a, b \in \mathcal{G}_v$ such that r_a lies below ℓ and $x(r_a) < x(l_b)$, but $b < a$. Since the x-projection of b is to the right of that of a, the only way b can precede a in $<$-ordering is by the transitive relation $<_{v^*}$. Thus there exists a sequence of segments in $\mathcal{G}_v$ such that $b = e_1 <_v e_2 <_v \cdots <_v e_k = a$. Let $\pi_{b,a}$ denote a shortest sequence among all such sequences, and let $d_{b,a}$ denote the length of $\pi_{b,a}$. We obtain a contradiction by showing that, for every $k > 0$, there is no sequence $\pi_{b,a}$ such that $d_{b,a} = k$.

Obviously $d_{b,a} > 2$, because $x(r_a) < x(l_b)$. If $d_{b,a} = 3$, then there is a segment $c \in \mathcal{G}_v$ such that $b <_v c <_v a$. This implies that $x(r_c) \geq x(l_b) > x(r_a) \geq x(l_c)$. Let q be the intersection point of c and the vertical line $x = x(r_a)$. Note that a and c satisfy the following properties: (i) $c <_v a$, (ii) $x(r_c) > x(r_a)$, (iii) a does not intersect c, and (iv) the point l_c lies above ℓ (because $c \in \mathcal{G}_v$). These properties imply that $\overline{qr_c}$ lies below ℓ (see Fig. 6), which contradicts the assumption that $b <_v c$ (because $x(r_c) \geq x(l_b)$ and c lies below b at $x = x(l_b)$). Hence $d_{b,a} > 3$.

Now assume that, for all segments $a, b \in \mathcal{G}_v$ satisfying the conditions of the lemma, either $a < b$ or $d_{b,a} \geq k$. Suppose there exists a pair a, b such that $b <_{v^*} a$ and $d_{b,a} = k$. Let $b = e_1 <_v \cdots <_v e_{k-1} <_v e_k = a$ be a corresponding shortest sequence $\pi_{b,a}$, and let $c = e_{k-1}$. Since $\pi_{b,a}$ is a shortest sequence, it is easily seen that $x(r_c) > x(r_a)$. Indeed, let e_j be the first segment in this sequence whose x-projection overlaps that of a. Then e_j must lie below a, for otherwise we would have obtained a cycle in $<_{v^*}$, which is impossible. Hence $e_j <_v a$ and we can shortcut the sequence after e_j. Clearly, $x(r_j) > x(r_a)$. Let q be the intersection of c with $x = x(r_a)$, as above. Again we can argue that $\overline{qr_c}$ lies below ℓ. If $x(r_c) < x(l_b)$, then c and b satisfy the property of the lemma, and thus contradict the inductive hypothesis because $d_{b,c} < k$. On the other hand, if $x(r_c) \geq x(l_b)$, then we have $c <_v b$ (because c lies below b at $x = x(l_b)$), contradicting the assumption that $b <_{v^*} c$. Hence, we can conclude that $a < b$, and this completes the proof. □

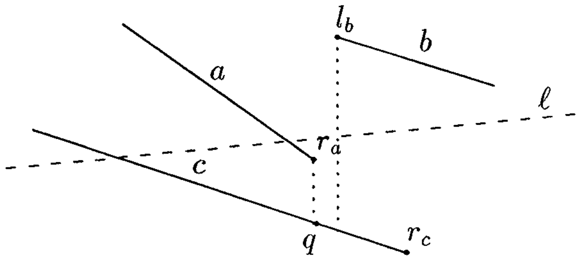

FIG. 6. *Illustration for Lemma* 3.5, $d_{b,a} = 3$.

Using Lemma 3.1 and Theorem 3.3 we obtain the following lemma.

LEMMA 3.6. *Let* $\mathcal{E} = (e_1, \cdots, e_m)$ *denote the segments of* $\mathcal{G}_v$ *ordered with respect to* $<$ *(relative to* $\mathcal{G}_v$*), and suppose* $\Phi(\mathcal{G}_v, \rho) = \rho \cap e_f$, *for some* $1 \leq f \leq m$. *Then for all* $i < f$, e_i *does not intersect* ρ.

Proof. If e_i intersects ρ, it does so at a point to the right of $\rho \cap e_f$. This implies that $e_f <_s e_i$, which means $e_f < e_i$ since $<$ extends $<_s$. □

Hence the original problem is reduced to the following restricted problem:

> *Given a sequence* $\mathcal{E}$ *of* m *segments sorted according to* $<$, *preprocess* $\mathcal{E}$ *so that for any (rightward-directed) query ray* ρ *emanating from a point* p *and lying on a line that passes below the left endpoints of all segments in* $\mathcal{E}$, *we can quickly determine* $e_f(\rho)$, *the first segment of* $\mathcal{E}$ *hit by the ray* ρ.

A possible approach to solving this problem is to do a binary search on $\mathcal{E}$, where each step of the search tests whether ρ intersects a segment in some contiguous subsequence of t segments of $\mathcal{E}$. If ρ were a full line ℓ, then such an intersection could be easily detected in $O(\log t)$ time after $O(t \log t)$ preprocessing (in which we construct the convex hull of the right endpoints of these t segments). However, no equally fast procedure is known to detect an intersection between a ray and such a set of segments. To overcome this problem, we next show how to reduce the intersection detection problem to one involving the line containing ρ rather than ρ itself.

For any point q in the plane, let $e_h = e_{h(q)}$ denote the first segment of $\mathcal{E}$ whose left endpoint lies to the right of (or above) q (see Fig. 7), and let $e_u = e_{u(q)}$ denote the segment in $\mathcal{E}$ lying immediately above q, that is, the vertical ray emanating from q in the upward direction hits e_u before any other segment. If e_h (respectively, e_u) is not defined, we put $h = m + 1$ (respectively, $u = m + 1$). Finally, put $\phi_q = \min\{h, u\}$.

To compute e_h, construct a balanced binary tree L whose leaves store the segments of $\mathcal{E}$ in their order in $\mathcal{E}$. For each interior node z of L we store the rightmost left endpoint of the segments stored at the subtree rooted at z. L can be constructed in $O(m \log m)$ time, and e_h can be determined, by searching for q through L, in $O(\log m)$ time. As for e_u, we can easily calculate it in time $O(\log m)$ after $O(m \log m)$ preprocessing, as in [ST].

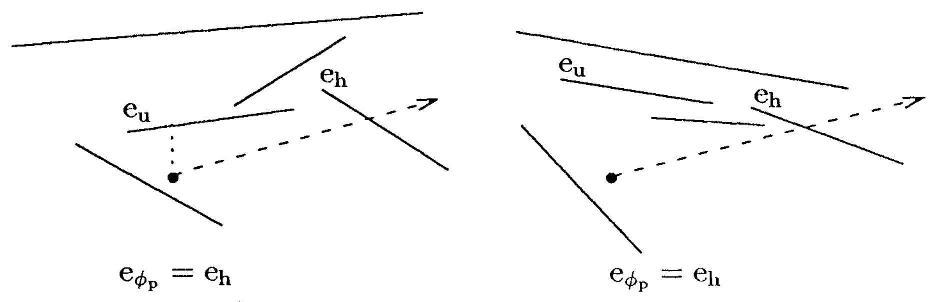

FIG. 7. *Segments e_h, e_u, and e_{ϕ_p}.*

LEMMA 3.7. *The query ray ρ emanating from a point p cannot intersect any segment $e_i \in \mathcal{E}$ for $i < \phi_p$. Moreover, ρ intersects e_i for $i \geq \phi_p$ if and only if its right endpoint lies below the line containing ρ.*

Proof. If the first part of the lemma were not true, then there would exist a segment e_i for $i < \phi_p$, intersecting the ray ρ. In this case the left endpoint of e_i must lie to the left of p, so the vertical ray η from p in the upward direction must intersect e_i. But then the first segment e_u hit by η must satisfy $i \geq u \geq \phi_p$ (because $e_u <_{v^*} e_i$ and by definition of ϕ_p), a contradiction that proves the first half of Lemma 3.7.

The "only if" part of the second half of Lemma 3.7 follows from the fact that if both the left and the right endpoints of a segment e lie above ℓ, then e cannot intersect ℓ. For the "if" part let e_i be a segment of $\mathcal{E}$, for $i \geq \phi_p$, whose right endpoint lies below ℓ, but e_i does not intersect ρ. If the left endpoint l_{e_i} of e_i lies to the right of p, then obviously e_i intersects ρ, so l_{e_i} must lie to the left of p. Since e_i does not intersect ρ, the intersection point ξ of e_i and ℓ lies to the left of p. Moreover if $x(r_{e_i}) < x(l_{e_{u(p)}})$, then by Lemma 3.5 $e_i < e_{u(p)}$. If $x(r_{e_i}) \geq x(l_{e_{u(p)}})$, then e_i and $e_{u(p)}$ must have x-projections that overlap at some point between ξ and p; since e_i lies below $e_{u(p)}$ at this point, we again have $e_i <_{v^*} e_{u(p)}$. Similarly we can show that $e_i < e_{h(p)}$. Hence $i < \min\{u, h\}$, contradicting the assumption that $i \geq \phi_p$. □

Lemma 3.7 implies that the binary search technique proposed above will work, provided we can detect quickly whether the right endpoint of any segment in some contiguous subsequence of $\mathcal{E}$ lies below ℓ. In other words, the problem now has been reduced to that of detecting an intersection between a set of points and a query half plane. Clearly, this is equivalent to detecting an intersection between the convex hull of these points and the half plane (see Fig. 8).

We are now ready to describe how to preprocess $\mathcal{E}$ so that $e_f(\rho)$, the first segment of $\mathcal{E}$ hit by ρ, can be computed quickly, for any ray ρ with the above properties. Let r_i denote the right endpoint of $e_i \in \mathcal{E}$, and let $R = \{r_1, \cdots, r_m\}$. We construct a binary tree $\mathcal{T}$ on R in the same way as we constructed $\mathcal{B}$, i.e., the points r_i are stored at the leaves of $\mathcal{T}$ in order, and each node w of $\mathcal{T}$ is associated with the subsequence R_w of R containing all points stored at the leaves of the subtree rooted at w.

At every node w of $\mathcal{T}$, we store the convex hull of R_w. Using $\mathcal{T}$ we can determine $e_f(\rho)$ in time $O(\log^2 m)$ as follows: We first find ϕ_p, as described above, in $O(\log m)$ time. Then we treat the suffix $\{r_{\phi_p}, \cdots, r_m\}$ of R as the union of $\log m$ subsets R_w, $w \in \mathcal{T}$, which we can compute in $O(\log m)$ time. We test each R_w in increasing, left-to-right order, to find the first w for which the line ℓ containing ρ intersects the hull of R_w. Then we do a binary search within R_w until we find $e_f(\rho)$. All this takes $O(\log^2 m)$ time.

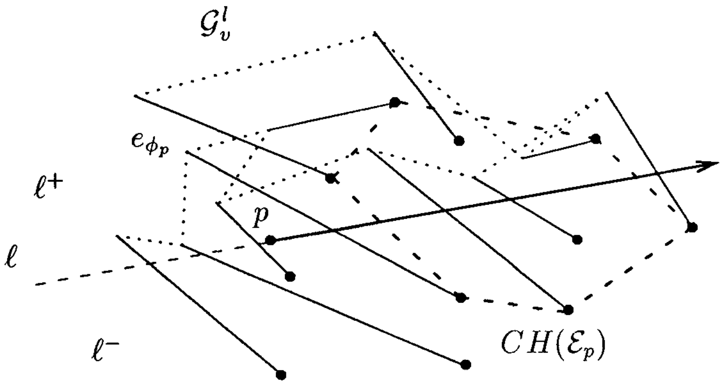

FIG. 8. *Convex hull* $\mathrm{CH}(\mathcal{E}_p)$ *intersecting* ℓ^-.

However, we can easily reduce the time to $O(\log m)$, using fractional cascading. This is possible since, as in [CGc], detecting intersection between ℓ and a convex polygon amounts to searching for the slope of ℓ in the sequence of slopes of the edges of the polygon (see [CGb] and [CGc] for more details). We thus have the following lemma.

LEMMA 3.8. *Given a set $\mathcal{E}$ of m nonintersecting line segments in the plane, we can preprocess it, in time $O(m \log m)$, into a data structure of size $O(m \log m)$ so that, given a (rightward-directed) query ray ρ whose containing line lies below the left endpoints of all the segments in $\mathcal{E}$, we can compute the first segment of $\mathcal{E}$ hit by ρ in time $O(\log m)$.*

Returning to the original problem, Lemma 3.8 and the preceding discussion imply that we can compute $\Phi(\mathcal{G}_v, \rho)$ for each $v \in V_B(\ell)$, in time $O(\log n)$. Equation (3.1) and Lemma 2.2 then imply the following theorem.

THEOREM 3.9. *Given a set $\mathcal{G}$ of n nonintersecting line segments, we can preprocess it in time $O(n^{3/2} \log^{\omega} n)$ for some $\omega < 4.33$ into a data structure of size $O(n \log^3 n)$, using $O(n^{3/2})$ working storage, so that, given a query ray ρ, its first intersection $\Phi(\mathcal{G}, \rho)$ with $\mathcal{G}$ can be computed in time $O(\sqrt{n} \log^2 n)$.*

Remark 3.10.

(i) The space used can be reduced to $O(n \log^2 n)$ without affecting the query time if we use a single tree structure instead of a family of $O(\log n)$ trees. But then the preprocessing time increases to $O(n^3 \log n)$ (see [EGH*]).

(ii) If we allow randomization, the (expected) preprocessing time of the algorithm can be reduced to $O(n^{4/3} \log^2 n)$ using Matoušek's algorithm for computing a family of spanning trees [Mab], but then the query time bound increases by a factor of $\log n$.

4. Trade-off between space and query time.

In this section we show that the query time for the ray shooting problem in arrangements of nonintersecting segments can be improved if we allow ourselves more storage. Similar trade-offs have been obtained for several related problems, such as computing a face in an arrangement of lines [EGH*] and simplex range searching [Agc], [Chd]. The main result of this section is an algorithm for computing $\Phi(\mathcal{G}, \rho)$ with $O(\frac{n}{\sqrt{m}} \log^{7/2} \frac{n}{\sqrt{m}} + \log n)$ query time, using $O(m)$ space, where $n \log^3 n \leq m \leq n^2$.

4.1. The case of quadratic storage.

First, we show that if we allow $O(n^2)$ space, the query time can be reduced to $O(\log n)$.

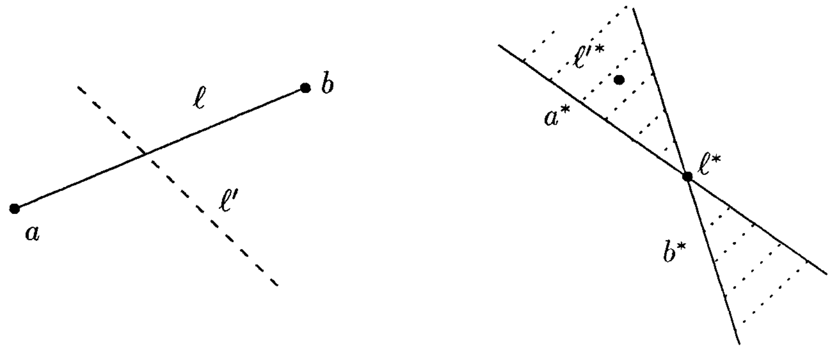

FIG. 9. *A segment e and its dual e*.*

Let $\mathcal{G} = \{e_1, \cdots, e_n\}$ be a collection of n nonintersecting segments. The dual of a segment $e = \overline{ab}$ is a double wedge e^* formed between the dual lines a^*, b^* of a, b, respectively, and not containing any vertical line (see Fig. 9). Dualize all segments $e \in \mathcal{G}$, obtaining a set $\mathcal{G}^*$ of n double wedges. Let $\mathcal{L}^*$ denote the set of lines bounding the double wedges of $\mathcal{G}^*$ (i.e., the duals of the endpoints of segments in $\mathcal{G}$). Let $\mathcal{A}(\mathcal{L}^*)$ denote the arrangement of $\mathcal{L}^*$, and let w_f be the set of segments dual to the double wedges of $\mathcal{G}^*$ containing the face $f \in \mathcal{A}(\mathcal{L}^*)$. Standard duality arguments yield the following lemma (see, e.g., [CGL]).

LEMMA 4.1. *Let p be a point lying in the interior of a face f of $\mathcal{A}(\mathcal{L}^*)$. Then p^* intersects each segment $e \in w_f$ transversally at an interior point, and is disjoint from any other segment of $\mathcal{G}$.*

LEMMA 4.2. *If the segments of $\mathcal{G}$ are nonintersecting, then for all points p in a face f of $\mathcal{A}(\mathcal{L}^*)$, the line p^* intersects the segments of w_f in the same order.*

Proof. Suppose there are two points x and y in a face f such that the lines x^* and y^* intersect the segments of w_f in two different orders. Since the segments in $\mathcal{G}$ are nonintersecting, rotating x^* towards y^* (in the direction that avoids a vertical orientation) we must reach a line p^* that either contains a segment of w_f, or passes through an endpoint of a segment of w_f. (Note that this claim does not hold if the segments can intersect.) The dual p of p^* is a point that lies on the segment $\overline{xy}$, hence in f. This, however, contradicts Lemma 4.1, thus showing that the duals of all points in f intersect the segments of w_f in the same order. □

Sort the segments in w_f in the order provided by Lemma 4.2. For a ray ρ, let the *image* of ρ be the dual of the line containing ρ. If the image of ρ lies in the face $f \in \mathcal{A}(\mathcal{L}^*)$, then $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time by a binary search on w_f. Therefore, it suffices to show how to store all the lists w_f using only $O(n^2)$ space, so that binary search in each of them can still be done in $O(\log n)$ time.

Let $\mathcal{D}$ denote the dual graph of $\mathcal{A}(\mathcal{L}^*)$, i.e., the graph whose nodes represent faces of $\mathcal{A}(\mathcal{L}^*)$ and whose edges connect pairs of nodes representing adjacent faces. Let $\mathcal{T}$ denote a spanning tree of $\mathcal{D}$. We can convert $\mathcal{T}$ into a path Π by tracing an Eulerian tour around the tree. Observe that if two vertices v_1, v_2 in Π represent faces f_1, f_2 sharing an edge γ, which is a portion of a line ℓ, then $w_{f_1} \oplus w_{f_2}$ is the set of segments having the dual of ℓ as an endpoint. Let δ_γ denote this set of segments. The set w_{f_2} can be obtained from w_{f_1} by deleting the segments of $\delta_\gamma \cap w_{f_1}$ and inserting the segments of $\delta_\gamma - w_{f_1}$.

Therefore, we can maintain all lists w_f using a persistent data structure (see [Co], [ST] and [DSST]). Since at each edge γ of Π only the segments of δ_γ are inserted or deleted, the total space required to store all w_f is $O(n + \sum_{\gamma\in\Pi} |\delta_\gamma|)$ and the total preprocessing time is $O((n + \sum_{\gamma\in\Pi} |\delta_\gamma|) \log n)$. Moreover, using the persistent data structure, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time (see [ST]). Thus, it suffices to prove that $\sum_\gamma |\delta_\gamma| = O(n^2)$. Suppose the segments of $\mathcal{G}$ have $t \le 2n$ distinct endpoints and ν_i segments are incident to the ith endpoint. It is easy to check that if γ is a portion of the line dual to the ith endpoint, then $|\delta_\gamma| \le \nu_i$. Obviously, $\sum_{i=1}^t \nu_i = 2n$ and each line of $\mathcal{L}^*$ is split into $\le t+1$ edges, which implies that

$$\sum_{\gamma\in\mathcal{D}} |\delta_\gamma| \quad \le \quad \sum_{i=1}^{t} (t+1)\nu_i \quad \le \quad 2n(2n+1).$$

Hence, we have the following theorem.

THEOREM 4.3. *Given a collection $\mathcal{G}$ of n nonintersecting segments, we can preprocess it, in $O(n^2 \log n)$ time, into a data structure of size $O(n^2)$ so that, for any query ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time.*

4.2. The general case. Theorems 3.9 and 4.3 represent roughly two extremes of the spectrum, because we need at least $O(n)$ space, and we cannot hope to answer a query in $o(\log n)$ time. The general case where the allowed storage m assumes an intermediate value between $n \log^3 n$ and n^2 is handled as follows. For technical reasons we assume for the time being that no endpoint of a segment in $\mathcal{G}$ has degree > 3 (that is, incident to more than three segments of $\mathcal{G}$). In §4.3 we show how to handle degenerate cases (i.e., when there are endpoints of degree > 3).

Using the algorithm of [Agb], partition the dual plane, in time $O(nr \log n \log^{\omega-1} r)$, into $M = O(r^2)$ triangles $\triangle_1, \cdots, \triangle_M$, each meeting at most $\frac{n}{r}$ lines dual to the endpoints of the given segments, where r is a parameter to be chosen later. Let $\mathcal{L}_i^*$ denote the set of dual lines that intersect the triangle $\triangle_i$ for $i = 1, \cdots, M$; $|\mathcal{L}_i^*| \le \frac{n}{r}$. For each $\triangle_i$, define the subset $\mathcal{G}_i$ of $\mathcal{G}$ to consist of all segments e having at least one endpoint whose dual is in $\mathcal{L}_i^*$. Obviously $|\mathcal{G}_i| \le \frac{3n}{r}$. We define $W_i \subset \mathcal{G}$ as

$$W_i \;=\; \{\, e \mid (e \in \mathcal{G}) \wedge (\triangle_i \subset e^*) \,\}.$$

LEMMA 4.4. *For each point p lying inside $\triangle_i$, the line p^* does not intersect any segment of $\mathcal{G} - (\mathcal{G}_i \cup W_i)$.*

Lemma 4.4 implies that

$$\Phi(\mathcal{G}, \rho) \;=\; \min\{\Phi(W_i, \rho),\ \Phi(\mathcal{G}_i, \rho)\}, \tag{4.1}$$

where $\triangle_i$ is the triangle containing the image of ρ. Using the same argument as in Lemma 4.2, we can prove

LEMMA 4.5. *All lines whose dual points lie inside $\triangle_i$ intersect the segments of W_i in the same order.*

We can thus order the segments of W_i in the order provided by Lemma 4.5, and compute $\Phi(W_i, \rho)$, for any ray ρ whose image lies in $\triangle_i$, in $O(\log n)$ time, using binary search. Let $\triangle_1$ and $\triangle_2$ be two adjacent triangles and let

$$\mathcal{G}_{12} \;=\; \{\, e \mid (e \in \mathcal{G}_1) \wedge (\triangle_2 \subset e^*) \,\}. \tag{4.2}$$

It follows from the definition of W_i that $W_2 = (W_1 \cup \mathcal{G}_{12}) - \mathcal{G}_2$. Since $|\mathcal{G}_1|, |\mathcal{G}_2| = O(\frac{n}{r})$, we have $|W_2 - W_1| = O(\frac{n}{r})$. As earlier, we define a graph $\mathcal{D}$, whose vertices are the triangles $\triangle_i$ and whose edges connect pairs of vertices representing adjacent triangles.

Now an edge between v_1 and v_2 has the set $\mathcal{G}_1 \cup \mathcal{G}_2$ associated with it. Again, we construct a path Π on a spanning tree of $\mathcal{D}$, and obtain a persistent data structure $\Upsilon_1(\mathcal{G})$ to store W_i for all triangles. It can be easily shown that $\Upsilon_1(\mathcal{G})$ requires $O(nr)$ space, and can be constructed in $O(nr \log n)$ time. For any ray ρ, $\Phi(W_i, \rho)$ can still be computed in $O(\log n)$ time, where $\triangle_i$ is the triangle containing the image of ρ.

We preprocess each $\mathcal{G}_i$ into a data structure $\Upsilon_2(\mathcal{G}_i)$ of size $O(|\mathcal{G}_i| \log^3 |\mathcal{G}_i|)$ for ray shooting queries, as described in §3, so that for any ray having its image in $\triangle_i$, we can find $\Phi(\mathcal{G}_i, \rho)$ in $O(\sqrt{\frac{n}{r}} \log^2 \frac{n}{r})$ time.

To compute $\Phi(\mathcal{G}, \rho)$, for a given ray ρ, we first find the triangle $\triangle_i$ that contains its image; this can be done in $O(\log n)$ time, using an efficient point location algorithm [EGS], [ST]. It follows from (4.1) that $\Phi(\mathcal{G}, \rho)$ can be computed by calculating $\Phi(W_i, \rho)$ and $\Phi(\mathcal{G}_i, \rho)$, as described above; therefore the query time is

$$Q(n) = O\left(\sqrt{\frac{n}{r}} \log^2 \frac{n}{r} + \log n\right).$$

As for the space complexity $S(n)$, we will need $O(r^2)$ space to store the triangle $\triangle_1, \cdots, \triangle_M$, $O(nr)$ space to store Υ_1 and $O(\frac{n}{r} \log^3 \frac{n}{r})$ to store each $\mathcal{G}_i$ (cf. Theorem 3.9). Thus,

$$S(n) = O(nr) + O\left(r^2 \cdot \frac{n}{r} \log^3 \frac{n}{r}\right) = O\left(nr \log^3 \frac{n}{r}\right).$$

If we choose $r = m/n \log^3 \frac{n}{\sqrt{m}}$, then $S(n) = O(m)$ and the query time becomes

$$\begin{aligned} Q(n) &= O\left(\sqrt{\frac{n}{m/(n \log^3 \frac{n}{\sqrt{m}})}} \cdot \log^2 \frac{n}{\sqrt{m}} + \log n\right) \\ &= O\left(\frac{n}{\sqrt{m}} \log^{7/2} \frac{n}{\sqrt{m}} + \log n\right). \end{aligned}$$

Next, we bound the preprocessing time $P(n)$. We can compute $\triangle_1, \cdots, \triangle_M$ in $O(nr \log n \cdot \log^{\omega-1} r)$ time (see [Agb]). Since Υ_1 can be constructed in $O(nr \log n)$ time and each $\mathcal{G}_i$ can be preprocessed in $O((\frac{n}{r})^{3/2} \log^{\omega} \frac{n}{r})$ time (cf. Theorem 3.9), we have

$$\begin{aligned} P(n) &= O(nr \log n \log^{\omega-1} r) + O\left(r^2 \cdot \frac{n^{3/2}}{r^{3/2}} \log^{\omega} \frac{n}{r}\right) \\ &= O\left(nr \log n \log^{\omega-1} r + n^{3/2}\sqrt{r} \log^{\omega} \frac{n}{r}\right) \\ &= O\left(n \frac{m}{n \log^3 \frac{n}{\sqrt{m}}} \log n \log^{\omega-1} \frac{m}{n} + n^{3/2} \sqrt{\frac{m}{n \log^3 \frac{n}{\sqrt{m}}}} \log^{\omega} \frac{n}{\sqrt{m}}\right) \\ &= O\left(m \log^{\omega} n + n\sqrt{m} \log^{\omega-3/2} \frac{n}{\sqrt{m}}\right). \end{aligned}$$

Since we need $O(nr)$ space to compute $\triangle_1, \cdots, \triangle_M$, and $O(\frac{n^{3/2}}{r^{3/2}})$ to preprocess each $\mathcal{G}_i$, the total space required for preprocessing is

$$O\left(nr + \frac{n^{3/2}}{r^{3/2}}\right) = O\left(\frac{m}{\log^3 \frac{n}{\sqrt{m}}} + \frac{n^{3/2}}{m^{3/2}/(n \log^3 \frac{n}{\sqrt{m}})^{3/2}}\right)$$

$$= O\left(m + \frac{n^3}{m^{3/2}} \log^{9/2} \frac{n}{\sqrt{m}}\right).$$

Hence, we can conclude the following theorem.

THEOREM 4.6. *Given a collection $\mathcal{G}$ of n nonintersecting segments in the plane with the property that no endpoint has degree > 3, and a parameter $n \log^3 n < m < n^2$, we can preprocess $\mathcal{G}$, in $O(m \log^{\omega} n + n\sqrt{m} \log^{\omega-3/2} n)$ time, into a data structure of size $O(m)$ so that, for query ray ρ, we can compute $\Phi(\mathcal{G}, \rho)$ in $O((n/\sqrt{m}) \log^{7/2}(n/\sqrt{m}) + \log n)$ time. The working storage required for preprocessing is $O(m + (n^3/m^{3/2}) \log^{9/2}(n/\sqrt{m}))$.*

Remark 4.7.

(i) If we allow randomization, then using Matoušek's algorithm $\mathcal{G}_i$ can be preprocessed in $O\left(\left(\frac{n}{r}\right)^{4/3} \log^2 n\right)$ time, but the query time increases by a factor of $O(\log n)$. Therefore, following the same analysis we obtain

$$\begin{aligned} P(n) &= O\left(m^{2/3}n^{2/3} + m \log^{\omega} n\right) \\ Q(n) &= O\left(\frac{n}{\sqrt{m}} \log^{9/2} \frac{n}{\sqrt{m}} + \log n\right). \end{aligned}$$

(ii) If we maintain a single tree data structure for each $\mathcal{G}_i$, the query time can be reduced to $O(\frac{n}{\sqrt{m}} \log^3 n)$, but the preprocessing time increases considerably.

4.3. Coping with degenerate cases. The analysis of the algorithm described in the previous subsection breaks down if the segments of $\mathcal{G}$ have endpoints of arbitrarily large degree, because then we cannot guarantee that $|\mathcal{G}_i| = O(\frac{n}{r})$, and the analysis to bound the total space required to store Υ_1 relies heavily on this bound for $|\mathcal{G}_i|$. In this subsection we overcome this difficulty by showing that, given a set $\mathcal{G}$ of n nonintersecting segments, we can transform it into another set $\mathcal{G}'$ of at most $3n$ (nonintersecting) segments such that no endpoint of a segment in $\mathcal{G}'$ has degree > 3, and $\Phi(\mathcal{G}, \rho)$ can be determined from $\Phi(\mathcal{G}', \rho)$ in $O(1)$ time.

Let $\mathcal{G}_p = \{e_1, \cdots, e_t\}$ be a subset of segments of $\mathcal{G}$ all having a common endpoint p. Let δ be the minimum distance from p to its closest neighbor in $\mathcal{G} - \mathcal{G}_p$, and let c be the circle of radius $\frac{\delta}{2}$ with p as center. For a segment $e_i \in \mathcal{G}_p$, let q_i denote the intersection point of c and e_i. Assume that the segments of $\mathcal{G}_p$ are ordered in counterclockwise direction along p. There are two cases to consider:

(i) There exist two consecutive segments in $\mathcal{G}_p$, say e_t and e_1, such that the angle between e_t and e_1 is $> 180°$. For $1 < i < t$, we remove the portion of e_i that lies in the interior of c (i.e., $\overline{pq_i}$), and add the segments $\overline{q_1q_2}, \cdots, \overline{q_{t-1}q_t}$ to $\mathcal{G}$ (see Fig. 10(a)).

(ii) The angle between every two consecutive segments of $\mathcal{G}_p$ is $< 180°$. For each $e \in \mathcal{G}_p$, we remove the portion of e that lies in the interior of c, and add the segments $\overline{q_1q_2}, \cdots, \overline{q_{t-1}q_t}, \overline{q_tq_1}$ to $\mathcal{G}$ (see Fig. 10(b)).

We repeat this process for each endpoint of the segments of $\mathcal{G}$ whose degree is greater than 3. Let $\mathcal{G}'$ be the new set of segments; obviously $|\mathcal{G}'| \le 3n$, and each endpoint has degree ≤ 3. If the ray origin s of ρ lies inside one of the newly created little polygons, say in $\triangle pq_{i-1}q_i$ of the polygon created around the endpoint p, then $\Phi(\mathcal{G}, \rho)$ lies on one of the segments incident to q_{i-1}, q_i, and can be determined in $O(\log n)$ time by locating s in $\mathcal{A}(\mathcal{G}')$. On the other hand, if s does not lie in any of the newly created polygons and $\Phi(\mathcal{G}', \rho)$ lies on a segment of $\mathcal{G}$, then $\Phi(\mathcal{G}, \rho) = \Phi(\mathcal{G}', \rho)$. Finally, if s lies outside all newly created polygons but $\Phi(\mathcal{G}', \rho)$ lies on a segment $\overline{q_{i-1}q_i}$ and e_{i-1} (respectively, e_i)

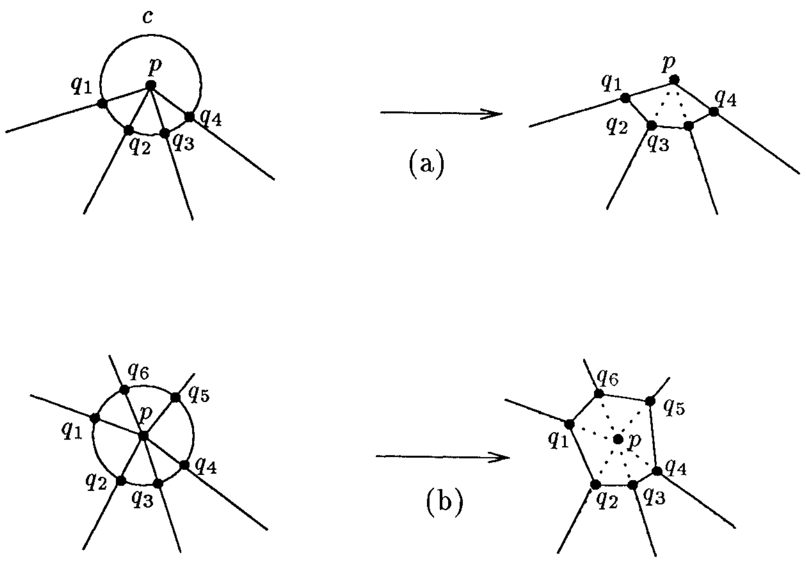

FIG. 10. *Modifying segments having a common endpoint of degree* > 3.

is the segment of $\mathcal{G}$ incident to q_{i-1} (respectively, q_i), then $\Phi(\mathcal{G}, \rho)$ lies on either e_{i-1} or e_i. Therefore, $\Phi(\mathcal{G}, \rho)$ can be computed from $\Phi(\mathcal{G}', \rho)$ in $O(1)$ time. Moreover, for each endpoint p, the minimum distance δ_p can be computed in $O(n \log n)$ time by constructing the closest point Voronoi diagram of $\mathcal{G}$ [Ya]. Hence by Theorem 4.6, we have the following theorem.

THEOREM 4.8. *Given a set $\mathcal{G}$ of n segments and a parameter $n \log^3 n < m < n^2$, we can preprocess it, in time $O(m \log^{\omega} n + n\sqrt{m} \log^{\omega-3/2}(n/\sqrt{m}))$, into a data structure of size $O(m)$ so that, for a query ray ρ, we can compute $\Phi(\mathcal{G}, \rho)$ in $O((n/\sqrt{m}) \log^{7/2}(n/\sqrt{m}) + \log n)$ time.*

5. Reporting all intersections. In the last two sections we gave algorithms to compute $\Phi(\mathcal{G}, \rho)$ for a collection of nonintersecting segments. We now extend these algorithms to solve the following problem:

Given a set $\mathcal{G}$ of n nonintersecting segments, preprocess it so that, for a query ray ρ, we can quickly report all intersections $\mathcal{I}_\rho$ between $\mathcal{G}$ and ρ in their order along ρ.

Dobkin and Edelsbrunner [DE] have given an algorithm that preprocesses $\mathcal{G}$ into a data structure of linear size so that, for a query ray ρ, $\mathcal{I}_\rho$ can be computed in $O(n^{0.695} + |\mathcal{I}_\rho|)$ time. (In fact, their algorithm works for an arbitrary collection of segments.) We first present an algorithm that uses roughly linear space, by generalizing the algorithm described in §3.

Preprocess $\mathcal{G}$, as in §3, in $O(n^{3/2} \log^{\omega} n)$ time using $O(n \log^3 n)$ space. For a given ray ρ, we compute $\mathcal{I}_\rho$ as follows. Let ℓ denote the line containing the ray ρ, and let $\mathcal{C}$ be a spanning path in C that intersects ℓ in $O(\sqrt{n})$ edges. As described in §2, compute $V_{\mathcal{B}}(\ell)$ in $O(\sqrt{n} \log n)$ time. Now we report all intersection points in $\mathcal{I}_p$ by walking along the ray ρ and stopping at each point of $\mathcal{I}_\rho$. For a point $q \in \rho$, let ρ_q be the ray emanating from q and contained in ρ.

The algorithm maintains the following invariant: When we are at a point $q \in \rho$, we maintain a list of all points $\Phi(\mathcal{G}_v, \rho_q)$ for $v \in V_{\mathcal{B}}(\ell)$, as a priority queue $\mathcal{Q}$ (with respect to their order along ρ). Observe that $\mathcal{Q}$ remains the same between two consecutive points of $\mathcal{I}_\rho$ and that the root of $\mathcal{Q}$ stores the point of $\mathcal{I}_\rho$ that we are going to encounter next. Therefore, it suffices to show how to update $\mathcal{Q}$ after visiting a point of $\mathcal{I}_\rho$. Suppose, when we are at a point q, the root of $\mathcal{Q}$ stores $\sigma = \Phi(\mathcal{G}_u, \rho_q)$ for some $u \in V_\mathcal{B}(\ell)$. It is easily seen that when we cross σ, the next intersection point of ρ and $\mathcal{G}_v$, for $v \in V_\mathcal{B}(\ell) - \{u\}$ does not change. Thus $\mathcal{Q}$ can be updated by deleting σ from $\mathcal{Q}$ and inserting $\Phi(\mathcal{G}_u, \rho_\sigma)$ in $\mathcal{Q}$ provided $\Phi(\mathcal{G}_u, \rho_\sigma) \neq \infty$. Continue this process until $\mathcal{Q}$ becomes empty. It is easily seen that this procedure reports all intersection points of ρ and the segments of $\mathcal{G}$ in their order along ρ.

In order to bound the running time of the algorithm, observe that initially we spend $O(\sqrt{n}\log^2 n)$ time to construct the queue $\mathcal{Q}$ for $q = p$, and then spend $O(\log n)$ time in updating $\mathcal{Q}$ after each intersection. Hence, we have the following theorem.

THEOREM 5.1. *Given a collection $\mathcal{G}$ of n nonintersecting segments, we can preprocess it, in time $O(n^{3/2}\log^\omega n)$, into a data structure of size $O(n\log^3 n)$ so that, given a query ray ρ, $\mathcal{I}_\rho$ can be computed in $O(\sqrt{n}\log^2 n + |\mathcal{I}_\rho|\log n)$ time.*

An immediate corollary of Theorem 5.1 is the following.

COROLLARY 5.2. *Given a collection $\mathcal{G}$ of n nonintersecting segments, we can preprocess it, in time $O(n^{3/2}\log^\omega n)$, into a data structure of size $O(n\log^3 n)$ so that, given a query segment e, we can compute all K intersections between e and $\mathcal{G}$ in time $O(\sqrt{n}\log^2 n + K\log n)$.*

Next we show that, as in §4, the query time can be improved if we allow more space. Now preprocess $\mathcal{G}$ as described in §4 (if the segments of $\mathcal{G}$ have endpoints with degree > 3, we modify the set $\mathcal{G}$, as described in §4.3). Recall that in §4 we maintain two data structures: (i) the persistent data structure Υ_1 to store W_i for each triangle $\triangle_i$, and (ii) $\Upsilon_2(\mathcal{G}_i)$ for ray shooting queries. For a query ray ρ, we compute $\mathcal{I}_\rho$ as follows.

Suppose the ray origin p lies in the triangle $\triangle_i$; let the sorted W_i be $(e_1, e_2, \cdots, e_m)$ and suppose $\Phi(W_i, \rho) \in e_k$. Then by Lemma 4.2, $e_k, \cdots, e_m$ intersect ρ in that order along ρ, and we thus obtain all intersections between W_i and ρ. The intersections between $\mathcal{G}_i$ and ρ are obtained by the procedure described above, except that the the size of $\mathcal{Q}$ is now only $O(\sqrt{\frac{n}{r}}\log\frac{n}{r})$ because $|\mathcal{G}_i| \leq \frac{n}{r}$. $\mathcal{I}_\rho$ is then obtained by merging the two output lists of intersections with W_i and $\mathcal{G}_i$. Hence, following the same analysis as in §4, we can conclude the following theorem.

THEOREM 5.3. *Given a collection $\mathcal{G}$ of nonintersecting segments and a parameter $n\log^3 n < m < n^2$, we can preprocess it, in time $O(m\log^\omega n + n\sqrt{m}\log^{\omega-3/2}(n/\sqrt{m}))$, into a data structure of size $O(m)$ so that, given a query ray ρ, $\mathcal{I}_\rho$ can be computed in $O((n/\sqrt{m})\log^{7/2}(n/\sqrt{m}) + \log n + |\mathcal{I}_\rho|\log(n/\sqrt{m}))$ time.*

COROLLARY 5.4. *Given a collection $\mathcal{G}$ of n nonintersecting segments and a parameter $n\log^3 n < m < n^2$, we can preprocess it, in $O(m\log^\omega n + n\sqrt{m}\log^{\omega-3/2}(n/\sqrt{m}))$ time, into a data structure of size $O(m)$ so that, given a segment e, we can compute all K intersections between e and $\mathcal{G}$ in $O((n/\sqrt{m})\log^{7/2}(n/\sqrt{m}) + \log n + K\log(n/\sqrt{m}))$ time.*

6. Ray shooting in general arrangements of segments.

In this section we extend our algorithm to arrangements of possibly intersecting segments. The section is organized as follows. In §6.1 we describe how to preprocess $\mathcal{G}$ for ray shooting queries, and in §6.2 we show how to answer a query. We analyze the time and space complexity of our algorithm in §6.3 and finally, in §6.4, we derive a trade-off between space and query time, similar to that of §4.

6.1. Preprocessing the segments.

In this section $\mathcal{G}$ denotes an arbitrary collection of n segments in the plane. To simplify the exposition, we assume that the segments of

$\mathcal{G}$ are bounded. The preprocessing of $\mathcal{G}$ is done as follows. We construct a partition tree $\mathcal{T}$, and associate with each node $v \in \mathcal{T}$ a collection $\mathcal{G}_v \subseteq \mathcal{G}$ of n_v segments, a triangle Δ_v, and another auxiliary set $\mathcal{G}'_v$ of n'_v segments. If $n_v \leq c$, for some fixed constant c, then v is a leaf of $\mathcal{T}$. Otherwise it is an internal node of $\mathcal{T}$, which is further processed as follows. For some fixed constant $r \geq 2$, partition Δ_v into $M = O(r^2)$ triangles $\Delta_1, \cdots, \Delta_M$, using the algorithm of [Agb] (or of [Maa]), so that the interior of each triangle Δ_i intersects at most $\frac{n_v}{r}$ lines containing the segments of $\mathcal{G}_v$. Create M children $w_1, \cdots, w_M$ of v, and associate with each child w_i the corresponding triangle $\Delta_{w_i} = \Delta_i$. We put a segment $e \in \mathcal{G}_v$ in $\mathcal{G}_{w_i}$ if at least one of the endpoints of e lies in Δ_i. We also associate with w_i an auxiliary set $\mathcal{G}'_{w_i}$ of all segments of $\mathcal{G}_v$ that intersect the interior of Δ_i. For the sake of convenience we regard each element of $\mathcal{G}'_{w_i}$ as the subsegment $e \cap \Delta_{w_i}$ of the corresponding segment e. Let $\mathcal{M}_v$ be the planar map formed by the triangles $\Delta_1, \cdots, \Delta_M$. The root u of $\mathcal{T}$ is associated with $\mathcal{G}$ itself, and Δ_u is a triangle that contains all the segments of $\mathcal{G}$. Moreover, $\mathcal{G}'_u = \mathcal{G}$, by definition.

We preprocess each node $v \in \mathcal{T}$ as follows. Preprocess the planar map $\mathcal{M}_v$ for point location queries (see [EGS] and [ST]) and store the resulting data structure at v. Let $\mathcal{L}_v$ denote the set of lines containing the segments of $\mathcal{G}'_v - \mathcal{G}_v$; $|\mathcal{L}_v| \leq n'_v$. Preprocess $\mathcal{L}_v$ into a data structure $\Upsilon_1(\mathcal{L}_v)$ for computing $\Phi(\mathcal{L}_v, \rho)$ as described in Edelsbrunner et al. [EGH*] (see also [Agc]). If $\Phi(\mathcal{L}_v, \rho)$ lies outside Δ_v, then we reset it to $+\infty$.

DEFINITION 6.1. The *zone* of a triangle Δ in an arrangement $\mathcal{A}(\mathcal{G})$ of a set $\mathcal{G}$ of segments is the collection of the face portions $f \cap \Delta$, for all faces $f \in \mathcal{A}(\mathcal{G})$, that intersect $\partial\Delta$ (see Fig. 11).

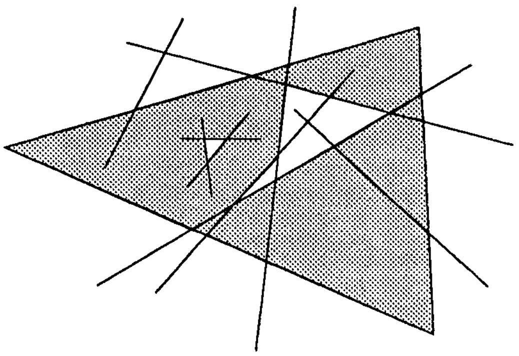

FIG. 11. *Zone of a triangle Δ_i; some faces are nonsimple polygons.*

Using the same argument as in [EGP*], it can be proved that the total number of edges in the zone of a triangle in an arrangement of n segments is $O(n\alpha(n))$, where $\alpha(n)$ is a functional inverse of Ackermann's function. Let $\mathcal{H}_v$ denote the zone of Δ_v in $\mathcal{A}(\mathcal{G}'_v)$; $\mathcal{H}_v$ has $O(n'_v\alpha(n'_v))$ edges. $\mathcal{H}_v$ can be computed using the algorithm of Guibas et al. [GSS] because under the assumption that the segments of $\mathcal{G}'_v$ have been clipped to Δ_v, $\mathcal{H}_v$ is the unbounded face of $\mathcal{A}(\mathcal{G}'_v)$. Since the edges of $\mathcal{H}_v$ are nonintersecting, we preprocess $\mathcal{H}_v$ into a data structure $\Upsilon_2(\mathcal{G}'_v)$ for computing $\Phi(\mathcal{H}_v, \rho)$, using the algorithm described in §3.

We repeat this preprocessing for every node v of $\mathcal{T}$. The resulting collection of data structures is the output of the preprocessing stage.

6.2. Answering a query. Let ρ be a query ray emanating from a point p in direction d. The query is answered by traversing a path Π_ρ of $\mathcal{T}$ and computing $\sigma_v = \Phi(\mathcal{G}_v, \rho)$ at each node $v \in \Pi_\rho$ in a bottom-up fashion. At the end of this process we obtain at the root u, $\sigma_u = \Phi(\mathcal{G}_u, \rho) = \Phi(\mathcal{G}, \rho)$.

The path Π_ρ is defined so that for each node v along Π_ρ the ray origin p lies in $\triangle_v$. At each node $v \in \Pi_\rho$ we compute σ_v as follows. Let W_ρ be the set of children w of v for which ρ intersects $\triangle_w$. Obviously,

$$\sigma_v = \min_{w \in W_\rho} \{\Phi(\mathcal{G}'_w, \rho)\}. \tag{6.1}$$

If $w \in \Pi_\rho$, that is, p lies in $\triangle_w$, then we have already computed $\Phi(\mathcal{G}_w, \rho)$. Concerning $\sigma'_w = \Phi(\mathcal{G}'_w - \mathcal{G}_w, \rho)$, since the segments of $\mathcal{G}'_w - \mathcal{G}_w$ completely cross $\triangle_w$, σ'_w is the same as $\Phi(\mathcal{L}_w, \rho)$ (we are assuming that $\Phi(\mathcal{L}_w, \rho)$ and σ'_v are set to $+\infty$ if they do not lie in the interior of $\triangle_w$). Thus σ'_w can be computed using $\Upsilon_1(\mathcal{L}_w)$.

For all other children $z \in W_\rho$, the fact that p lies outside $\triangle_z$ implies that $\Phi(\mathcal{G}'_z, \rho)$ is the same as $\Phi(\mathcal{H}_z, \rho)$, and therefore it can be computed using $\Upsilon_2(\mathcal{G}'_z)$ (see Fig. 12).

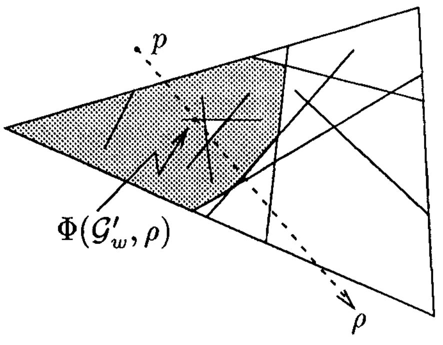

FIG. 12. $\mathcal{A}(\mathcal{G}'_w)$; *shaded region is a face of* $\mathcal{H}_w$.

Repeating the above step for all $w \in W_p$ and using (6.1) we can compute σ_v.

6.3. Analysis of the algorithm. The correctness of the algorithm follows from the above discussion, so we only have to analyze the time and space complexity of the algorithm. First, consider the query time $Q(n)$. Let Π_ρ be the path followed by the algorithm as it computes $\Phi(\mathcal{G}, \rho)$. We bound the time spent at each node $v \in \Pi_\rho$. We spend $O(\log r)$ time to find the triangle $\triangle_w$ containing the ray origin p. It follows from [EGH*] (see also [Aga] and [Agc]) that $\Phi(\mathcal{L}_w, \rho)$ can be computed in $O(\sqrt{n'_w} \log n'_w)$ time. At other triangles $\triangle_z$ intersected by ρ, we spend $O(\sqrt{n'_z \alpha(n'_z)} \log^2 n'_z)$ time to compute $\Phi(\mathcal{G}'_z, \rho)$, for $\mathcal{H}_z$ has at most $O(n'_z \alpha(n'_z))$ edge (cf. Theorem 3.9). Since $n'_z \le n_v$ for all children of v, the time spent at v is $O(r^2 \sqrt{n_v \alpha(n_v)} \log^2 n_v)$. Summing over all nodes of Π_ρ and using the fact that $r = O(1)$, we obtain

$$Q(n) = \sum_{v \in \Pi_\rho} O\left(\sqrt{n_v \alpha(n_v)} \log^2 n\right). \tag{6.2}$$

For a node v at level i, $n_v \le (n/r^i)$; therefore,

$$\begin{aligned} Q(n) &= \sum_{i=0}^{\log n} O\left(\sqrt{\frac{n\alpha(n)}{r^i}} \log^2 n\right) \\ &= O(\sqrt{n\alpha(n)} \log^2 n), \end{aligned}$$

because $r \geq 2$. Next, let us analyze the space complexity $S(n)$ and the preprocessing time $P(n)$ of our algorithm. At each node $v \in \mathcal{T}$ we store the following data structures:

(i) $\mathcal{M}_v$: The node v is partitioned into $O(r^2)$ triangles in $O(n_v r \log n_v \log^{\omega-1} r)$ time [Agb], therefore by [EGS], $\mathcal{M}_v$ can be preprocessed, in time $O(r^2 \log r)$, into a data structure of size $O(r^2)$ for point location queries. Since r is chosen to be constant, the time bound is just $O(n_v \log n_v)$ and the space required is $O(n_v)$.

(ii) $\Upsilon_1(\mathcal{L}_v)$: It follows from the result of Edelsbrunner et al. [EGH*] (see also [Agc]) that $\Upsilon_1(\mathcal{L}_v)$ requires $O(n'_v \log^2 n'_v)$ space and $O(n'^{3/2}_v \log^\omega n'_v)$ preprocessing time.

(iii) $\Upsilon_2(\mathcal{G}'_v)$: Since $\mathcal{H}_v$ has $O(n'_v \alpha(n'_v))$ edges, $\Upsilon_2(\mathcal{G}'_v)$ requires $O(n'_v \alpha(n'_v) \log^3 n'_v)$ space and $O(n'^{3/2}_v \alpha^{3/2}(n'_v) \log^\omega n'_v)$ preprocessing time (which subsumes the time $O(n'_v \alpha(n'_v) \log^2 n'_v)$ needed to compute $\mathcal{H}_v$ [GSS]).

Thus, the space used at v is $O(n'_v \alpha(n_v) \log^3 n_v)$. Summing over all nodes of $\mathcal{T}$, we get

$$S(n) = \sum_{v \in \mathcal{T}} (n'_v \alpha(n'_v) \log^3 n'_v).$$

Observe that each triangle of $\mathcal{M}_v$ intersects $O(\frac{n_v}{r})$ segments of $\mathcal{G}_v$, so $\sum_{w \in W_\rho} n'_w = O(n_v r)$. Consequently,

$$\begin{aligned} S(n) &= \sum_{v \in \mathcal{T}} O(n_v r \alpha(n_v) \log^3 n_v) \\ &= \sum_{v \in \mathcal{T}} O(n_v \alpha(n_v) \log^3 n_v) \end{aligned}$$

Since each endpoint of a segment $e \in \mathcal{G}$ falls in the interior of only one triangle, $\triangle_v$, for each level of $\mathcal{T}$, e appears in $\mathcal{G}_v$ of at most two nodes of the same level. Let $l(v)$ denote the level of the node v in $\mathcal{T}$. Then for every $i \leq \log n$ we have

$$\sum_{l(v)=i} n_v \leq 2n. \tag{6.3}$$

Hence,

$$S(n) = \sum_{i=1}^{\log n} O(n\alpha(n) \log^3 n) = O(n\alpha(n) \log^4 n).$$

Finally, we bound the preprocessing time $P(n)$ of our algorithm. The above discussion implies that the total time spent in preprocessing is at most

$$\begin{aligned} P(n) &= \sum_{v \in \mathcal{T}} O(n'^{\,3/2}_v \alpha^{3/2}(n'_v) \log^\omega n) \\ &= \sum_{v \in \mathcal{T}} O(n^{3/2}_v \alpha^{3/2}(n_v) \log^\omega n). \end{aligned} \tag{6.4}$$

Since ω is a constant less than 4.33, we can write $\alpha^{3/2}(n_v) \log^\omega n_v$ in (6.4) as $\log^\omega n$, where ω is a different constant but whose value is still less than 4.33. Thus

$$P(n) = \sum_{i=1}^{\log n} \sum_{l(v)=i} O(n^{3/2}_v \log^\omega n)$$

$$\begin{aligned}
&= \sum_{i=1}^{\log n} O\left(r^i \left(\frac{n}{r^i}\right)^{3/2} \log^\omega n\right)\\
&= O\left(n^{3/2} \log^\omega n \sum_{i=1}^{\log n} \frac{1}{(\sqrt{r})^i}\right)\\
&\le O(n^{3/2} \log^\omega n),
\end{aligned}$$

because $r \ge 2$. Hence, we can conclude the following theorem.

THEOREM 6.2. *Given a collection $\mathcal{G}$ of n (possibly intersecting) segments, we can preprocess $\mathcal{G}$, in time $O(n^{3/2} \log^\omega n)$, into a data structure of size $O(n\alpha(n) \log^4 n)$ so that, for any query ray ρ, we can compute $\Phi(\mathcal{G}, \rho)$ in $O(\sqrt{n\alpha(n)} \log^2 n)$ time.*

Remark 6.3. If $\mathcal{G}$ contains unbounded segments, then the triangle $\triangle_u$ associated with the root u of $\mathcal{T}$ should be a triangle that contains all intersection points and all bounded segments of $\mathcal{G}$. Such a $\triangle_u$ can be easily computed in $O(n \log n)$ time. Now for each segment $e \in \mathcal{G}$, we compute $e' = e \cap \triangle_u$ and apply our algorithm to the new set of segments. The portions of the segments lying in the exterior of $\triangle_u$ do not intersect each other, and are ordered in the nondecreasing order of their slopes along $\partial\triangle_u$ in counterclockwise direction. Therefore, if a query ray does not hit a segment of $\mathcal{G}$ inside $\triangle_u$, we can determine, in additional $O(\log n)$ time, the first segment hit by the ray outside $\triangle_u$, which shows that our algorithm works for unbounded segments as well.

6.4. Trade-off between space and query time. In this subsection we establish a trade-off between space and query time for ray shooting in general arrangements of segments. As in §4 we first give a very simple algorithm that preprocesses $\mathcal{G}$, in time $O(n^2\alpha^2(n) \log n)$, into a data structure of size $O(n^2\alpha^2(n))$ so that, given a query ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time.

Compute the arrangement $\mathcal{A}(\mathcal{G})$ in time $O(n^2 \log n)$ using the line sweep method [PS] (or in time $O(n^2)$ using a more involved algorithm [EOS]), and preprocess $\mathcal{A}(\mathcal{G})$ for point location queries [EGS], [ST]. Since the edges of $\mathcal{A}(\mathcal{G})$ are nonintersecting, we can preprocess each face $f \in \mathcal{A}(\mathcal{G})$ into a data structure Υ_f for logarithmic-time ray shooting queries, using $O(|n_f|^2)$ space, where n_f is the number of edges bounding f, as described in §4.

To compute $\Phi(\mathcal{G}, \rho)$, for a query ray ρ, first locate the face f of $\mathcal{A}(\mathcal{G})$ containing the ray origin p. Obviously $\Phi(\mathcal{G}, \rho)$ lies on the boundary of f, and therefore $\Phi(\mathcal{G}, \rho) = \Phi(\partial f, \rho)$ can be computed in $O(\log n)$ time, using Υ_f. Thus, the overall query time is $O(\log n)$.

As for the storage, $\mathcal{A}(\mathcal{G})$ can be preprocessed for point location queries using $O(n^2)$ space (cf. [EGS] and [ST]). The total space required to store all Υ_f is $O(\sum_f n_f^2)$. Theorem 4.3 implies that the preprocessing time is $O(\sum_f n_f^2 \log n)$. It has been shown in [EGP*] that

$$\sum_{f \in \mathcal{A}(\mathcal{G})} n_f^2 = O(n^2\alpha^2(n)).$$

Hence, we have the following theorem.

THEOREM 6.4. *Given a collection $\mathcal{G}$ of n segments in the plane, we can preprocess $\mathcal{G}$, in time $O(n^2\alpha^2(n) \log n)$, into a data structure of size $O(n^2\alpha^2(n))$ so that, for a query ray ρ, $\Phi(\mathcal{G}, \rho)$ can be computed in $O(\log n)$ time.*

Next we give an algorithm for the general case, where $n^{1+\epsilon_0} \le m \le n^{2-\epsilon_1}$, for some constants $\epsilon_0, \epsilon_1 > 0$. Let $m = f(n)$. To preprocess $\mathcal{G}$ into a data structure of size $O(m)$, we proceed in the same way as in §6.1 except that at each node $v \in \mathcal{T}$ we are allowed more space, so we construct larger-size data structures that facilitate faster ray shooting in $\mathcal{L}_v$, $\mathcal{G}'_v$, etc.

Edelsbrunner et al. [EGH*] (see also [Agc]) have shown that, given a set $\mathcal{L}$ of n lines and a parameter $1 \le \beta < n$, $\mathcal{L}$ can be preprocessed, in $O(n^{3/2}\sqrt{\beta}\log^{\omega} n)$ time,[1] into a data structure $\Upsilon_1(\mathcal{L})$ of size $O(n\beta \log^2 n)$ so that, for any query ray ρ, we can compute $\Phi(\mathcal{L}, \rho)$ in $O(\sqrt{n/\beta}\log n)$ time. At each node v of level i, we store $\Upsilon_1(\mathcal{L}_v)$ with an appropriate value of $\beta = \beta_i$ (to be specified later).

Similarly, we have shown in §4 that, given a set $\mathcal{E}$ of n nonintersecting segments and a parameter β, we can preprocess $\mathcal{E}$, in time $O(n^{3/2}\sqrt{\beta}\log^{\omega} n)$, into a data structure $\Upsilon_2(\mathcal{E})$ of size $O(n\beta\log^3 n)$ so that, for a query ray ρ, we can compute $\Phi(\mathcal{E}, \rho)$ in time $O(\sqrt{n/\beta}\log^2 n)$. For a node v at level i, we store $\Upsilon_2(\mathcal{G}'_v)$ with $\beta = \beta_i$. (Recall that if $\mathcal{H}_v$ has a vertex of degree > 3, then the segments of $\mathcal{H}_v$ need to be modified, as described in §4.3.)

Next, we analyze the complexity of this algorithm. First, consider the space used by our algorithm. Since $|\mathcal{L}_v| \le n'_v$ and $|\mathcal{H}_v| = O(n'_v\alpha(n'_v))$, the space used by a node v of $\mathcal{T}$ at level i is $O(n'_v\beta_i \log^2 n'_v + n'_v\alpha(n'_v)\beta_i\log^3 n'_v) = O(n'_v\alpha(n)\beta_i\log^3 n'_v)$. The total space used is therefore

$$\begin{aligned} S(n) &= \sum_{i=0}^{\log n}\sum_{l(v)=i} O(n'_v\alpha(n)\beta_i\log^3 n'_v) \\ &= \sum_{i=0}^{\log n} O\left(\left(\sum_{l(v)=i} n'_v\right)\beta_i\alpha(n)\log^3 n'_v\right) \\ &= \sum_{i=0}^{\log n} O(\beta_i n\alpha(n)\log^3 n_i), \end{aligned}$$

where n_i is the maximum value of $|\mathcal{G}'_v|$ for a node v at level i. The last equality follows from (6.3) and the fact that $n'_v \le rn_v$. If we choose $\beta_i = (f(n_i)/(n_i\alpha(n)\log^3 n_i))$, which is easily seen to satisfy $\beta_i > 1$, then

$$\begin{aligned} S(n) &= \sum_{i=0}^{\log n} O\left(\frac{f(n_i)}{n_i\alpha(n)\log^3 n_i} n\alpha(n)\log^3 n_i\right) \\ &= O\left(\sum_{i=0}^{\log n} r^i f(n/r^i)\right) \\ &= O(f(n)) \quad \text{because } f(n) > n^{1+\epsilon_0} \\ &= O(m). \end{aligned}$$

As for the query time,

$$Q(n) = \sum_{i=0}^{\log n} O\left(\sqrt{\frac{n_i\alpha(n_i)}{\beta_i}}\log^2 n_i\right)$$

[1]Actually, the preprocessing time is $O((n\beta + n^{3/2}\sqrt{\beta})\log^{\omega} n)$, but it can be verified that for our choice of β the first term never dominates, so for simplicity we only write the second term.

$$\begin{aligned}
&= \sum_{i=0}^{\log n} O\left(\sqrt{\frac{n_i\alpha(n_i)}{f(n_i)/(n_i\alpha(n)\log^3 n_i)}}\log^2 n_i\right)\\
&= \sum_{i=0}^{\log n} O\left(\frac{n/r^i}{\sqrt{f(n/r^i)}}\alpha(n)\log^{7/2} n\right)\\
&= O\left(\frac{n}{\sqrt{f(n)}}\alpha(n)\log^{7/2} n\right) \quad \text{because } f(n) < n^{2-\epsilon_1}\\
&= O\left(\frac{n\alpha(n)}{\sqrt{m}}\log^{7/2} n\right).
\end{aligned}$$

Finally, the preprocessing time at a node v of level i is $O(n'^{3/2}_v\sqrt{\beta_i}\alpha^{3/2}(n'_v)\log^\omega n'_v)$. The total preprocessing time is thus

$$\begin{aligned}
P(n) &= \sum_{i=0}^{\log n}\sum_{l(v)=i} O(n'^{3/2}_v\sqrt{\beta_i}\alpha^{3/2}(n'_v)\log^\omega n'_v)\\
&= \sum_{i=0}^{\log n} O\left(\left(\sum_{l(v)=i} n'^{3/2}_v\right)\sqrt{\beta_i}\alpha^{3/2}(n_i)\log^\omega n_i\right)\\
&= \sum_{i=0}^{\log n} O\left(r^i\left(\frac{n}{r^i}\right)^{3/2}\sqrt{\frac{r^i f(n/r^i)}{n\alpha(n)\log^3 n}}\alpha^{3/2}(n)\log^\omega n\right)\\
&= O\left(n\alpha(n)\log^{\omega-3/2} n\sum_{i=0}^{\log n}\sqrt{f(n/r^i)}\right)\\
&= O(n\sqrt{f(n)}\alpha(n)\log^{\omega-3/2} n)\\
&= O(n\sqrt{m}\alpha(n)\log^{\omega-3/2} n).
\end{aligned}$$

Using the same argument as for (6.5), we can ignore the term $\alpha(n)$ in the above equality. Hence, we can conclude the following theorem.

THEOREM 6.5. *Given a set $\mathcal{G}$ of n segments and a parameter $n^{1+\epsilon_0} \le m \le n^{2-\epsilon_1}$, for some constants $\epsilon_0, \epsilon_1 > 0$, we can preprocess $\mathcal{G}$, in $O(n\sqrt{m}\log^{\omega-3/2} n)$ time, into a data structure of size $O(m)$ so that, for any query ray ρ, we can compute $\Phi(\mathcal{G},\rho)$ in time $O\left((n\alpha(n)/\sqrt{m})\log^{7/2} n\right)$.*

Remark 6.6. The algorithm of [EGH*] actually constructs $\Upsilon_1(\mathcal{L})$, in time

$$O\left(n\beta\log n\log^{\omega-1}\beta + n^{3/2}\sqrt{\beta}\log^\omega\frac{n}{\beta}\right),$$

using $O(n\beta\log\frac{n}{\beta})$ space, and answers a query in time $O(\sqrt{n/\beta}\log\frac{n}{\beta})$. Similarly the algorithm described in §4 constructs $\Upsilon_2(\mathcal{G})$, in time

$$O\left(n\alpha(n)\beta\log n\log^{\omega-1}\beta + (n\alpha(n))^{3/2}\sqrt{\beta}\log^\omega\frac{n}{\beta}\right),$$

using $O(n\beta\log\frac{n}{\beta})$ space, and answers a query in $O(\sqrt{n/\beta}\log^2\frac{n}{\beta})$ time. Using these bounds in the above analysis, we can improve the query time $Q(n)$ to

$$O\left(\frac{n\alpha(n)}{\sqrt{m}}\log^{7/2}\left(\frac{n\alpha(n)}{\sqrt{m}}\right) + \log n\right).$$

The preprocessing time is now

$$O\left(m \log^{\omega} n + n\alpha(n)\sqrt{m} \log^{\omega-3/2}\left(\frac{n\alpha(n)}{\sqrt{m}}\right)\right).$$

7. Implicit point location. The planar point location problem is a well-studied problem in computational geometry [Ki], [EGS], [ST]. In this problem we are to preprocess a given planar subdivision so that, for a query point, we can quickly determine the face of the subdivision containing it. Guibas et al. [GOS] have considered a generalization of this problem, in which the map is defined as the arrangement of n possibly intersecting polygonal objects of some simple shape, and the goal is to compute, for a query point p, certain information related to its position within the arrangement of the objects; for example, to determine whether p lies in the union of the objects. For simplicity we break the given objects into a collection of segments, and consider the following formal statement of the problem:

We are given a collection $\mathcal{G} = \{e_1, \cdots, e_n\}$ of n segments, and with each segment e we associate a function ψ_e defined on the entire plane, which assumes values in some associative and commutative semigroup S (denote its operation by $+$). Define $\Psi(x) = \sum_{e \in \mathcal{G}} \psi_e(x)$. We want to preprocess $\mathcal{G}$ so that, for any query point p, we can quickly compute $\Psi(p)$.

We assume that ψ_e and Ψ satisfy the following conditions:

(i) For any given point x, $\psi_e(x)$ can be computed in $O(1)$ time.

(ii) Any two values in S can be added in $O(1)$ time.

(iii) Given a set $\mathcal{G}$ of n segments in the plane, we can preprocess it in time $O(n \log^k n)$, for some constant $k \geq 0$, into a linear-size data structure $\mathcal{D}(\mathcal{G})$ so that, given a point x lying either above all the lines containing the segments of $\mathcal{G}$, or below all these lines, $\Psi(x)$ can be calculated in $O(\log n)$ time.

It is shown in [GOS] that many natural problems including the problem of determining whether p lies in the union of the given objects, or of counting how many objects contain p, fall into this scheme. See also the following section for details.

The goal is to come up with an algorithm that uses $O(n \log^{O(1)} n)$ space and computes $\Psi(p)$, for any query point p, in sublinear time. Guibas et al. [GOS] gave a randomized algorithm, with $O(n \log^{k+1} n)$ expected running time, to construct a data structure of $O(n)$ size so that, for a query point p, $\Psi(p)$ can be computed in $O(n^{2/3+\delta})$ time for any $\delta > 0$. In this section we present an algorithm that improves the query time to $O(\sqrt{n} \log^2 n)$, and makes the preprocessing deterministic (albeit no longer close to linear).

Let $\mathcal{L}$ denote the set of lines containing the segments of $\mathcal{G}$. Dualize the lines of $\mathcal{L}$ to obtain a set $\mathcal{L}^*$ of n points. Let $\mathbf{C} = \{C_1, \cdots, C_k\}$ denote a family of $k = O(\log n)$ spanning paths on $\mathcal{L}^*$ with $\sigma(\mathbf{C}) = O(\sqrt{n})$. We show how to preprocess a single path $C \in \mathbf{C}$.

First, construct a binary tree $\mathcal{B} = \mathcal{B}(C)$ as in §2. With each node v of $\mathcal{B}$ we associate a set $\mathcal{G}_v$ of segments $e \in \mathcal{G}$ such that the dual of the line containing e belongs to S_v (as defined in §2). At each node v we store $\mathcal{D}(\mathcal{G}_v)$ so that, for any query point p lying either above all the lines containing the segments of $\mathcal{G}_v$ or below all of them, $\Psi_v(p) = \sum_{e \in \mathcal{G}_v} \psi_e(p)$ can be computed in $O(\log n)$ time.

For a given query point p, we compute $\Psi(p)$ as follows. Let p^* denote the dual of p. Obviously,

$$\sum_{i=1}^{n} \psi_{e_i}(p) = \sum_{v \in V_{\mathcal{B}}(p^*)} \Psi_v(p).$$

Therefore, it suffices to show how to compute $\Psi_v(p)$, for a node $v \in V_{\mathcal{B}}(p^*)$. Observe that for any $v \in V_{\mathcal{B}}(p^*)$, p^* lies either above all the points of S_v, or below all of them, say below. Since duality preserves the above-below relationship, p lies below all the lines containing the segments of $\mathcal{G}_v$. Therefore, $\Psi_v(p)$ can be easily computed in $O(\log n)$ time using $\mathcal{D}(\mathcal{G}_v)$.

Next, let us analyze the complexity of our algorithm. First consider the time spent in answering a query. By Theorem 2.1, we can determine, in $O(\log n)$ time, a path $\mathcal{C} \in \mathbf{C}$ that intersects p^* in at most $O(\sqrt{n})$ edges, and it follows from the discussion in §2 that $V_{\mathcal{B}}(p^*)$, for a given line p^*, can be computed in $O(\sqrt{n}\log n)$ time. By property (iii), for each $v \in V_{\mathcal{B}}(p^*)$, $\Psi_v(p)$ can be calculated in $O(\log n)$ time. The total time spent is thus $O(\sqrt{n}\log^2 n)$. As for the space complexity, $\mathcal{D}(\mathcal{G}_v)$ requires $O(|\mathcal{G}_v|)$ space. Since the segments associated with the nodes of $\mathcal{B}$ at the same level are pairwise disjoint, the total space required to store $\mathcal{B}$ is $O(n\log n)$. Finally, the preprocessing time is bounded by the time spent in computing $\mathbf{C}$ plus the time spent in preprocessing $\mathcal{G}_v$ for all $v \in \mathcal{B}$. Hence, the total preprocessing time is $O(n^{3/2}\log^{\omega} n + n\log^{k+2} n) = O(n^{3/2}\log^{\omega} n)$.

Therefore, we can conclude the following theorem.

THEOREM 7.1. *Given a collection $\mathcal{G}$ of n segments, and function ψ_e associated with each segment satisfying properties* (i)–(iii), *we can preprocess $\mathcal{G}$, in $O(n^{3/2}\log^{\omega} n)$ time, into a data structure of size $O(n\log^2 n)$ so that, for any query point p, $\Psi(p)$ can be computed in $O(\sqrt{n}\log^2 n)$ time.*

Remark 7.2.

(i) As in §3, we can reduce the space complexity to $O(n\log n)$ by maintaining a single tree structure instead of a family of $O(\log n)$ trees. Also, if we allow randomization, then the (expected) preprocessing time is $O(n^{4/3}\log^2 n)$, but the query time increases by a factor of $\log n$.

(ii) In some applications, where calculation of $\Psi(x)$ in (iii) above is accomplished by a binary search, it is possible to reduce the query time to $O(\sqrt{n}\log n)$, using fractional cascading.

(iii) As in the case of the ray shooting problem, the query time can be improved by allowing more storage. Instead of describing the trade-off for the general case, we will describe it in the next section for a specific example.

(iv) In a companion paper [Agc] we solve the *batched* version of this problem, where all the query points p are given in advance. We present there a solution that runs in time

$$O\left(m^{2/3}n^{2/3}\log^{2/3} n \log^{\omega/3}\frac{n}{\sqrt{m}} + n\log^k n \log\frac{n}{\sqrt{m}} + m\log n\right),$$

where m is the number of given query points.

8. Other applications. In this section we consider other applications of our technique. All these problems were studied in [GOS], who obtained algorithms with $O(n^{2/3+\delta})$ query time, for any $\delta > 0$. We show that using our approach the query time can be reduced to roughly $\sqrt{n}$.

8.1. Polygon containment problem: Preprocessing version. First consider the following problem:

> *Given a set* $\mathbf{T}$ *of* n *(possibly intersecting) triangles, we want to preprocess* $\mathbf{T}$ *so that, given a query point* p*, we can quickly count the number of triangles in* $\mathbf{T}$ *containing* p *(or just determine whether* p *lies in the union of these triangles); see Fig.* 13.

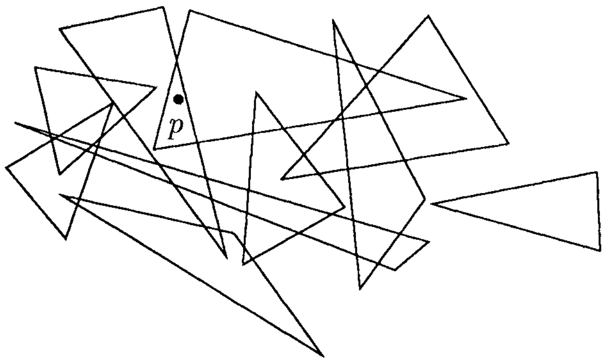

FIG. 13. *Polygon containment problem.*

We first present an algorithm that uses roughly linear space, and then show that the query time can be improved by using more space. Our algorithm is based on the following observation of [GOS]. Let $\mathcal{G}$ denote the set of edges bounding the triangles in $\mathbf{T}$ and, for each $e \in \mathcal{G}$, let $B(e)$ denote the semi-infinite trapezoidal strip lying below e. Define a function ψ_e in the plane so that $\psi_e(p) = 0$ for a point p outside B_e, and for $p \in B_e$, $\psi_e(p) = 1$, if the triangle corresponding to e lies below the line containing the segment e, otherwise $\psi_e(p) = -1$. It can be checked that $\Psi(p)$, for a point p, gives the number of triangles of $\mathbf{T}$ containing p. Moreover, ψ_e obviously satisfies properties (i) and (ii). As to property (iii), if a point p lies above all lines containing the given edges then $\Psi(p) = 0$, by definition. On the other hand, if p lies below all these lines, we do the following. Let $\hat{e}$ denote the x-projection of an edge e of some triangle. It is easily checked that

$$\Psi(p) = \sum_{p_x \in \hat{e}_j} \epsilon_j,$$

where p_x is the x-coordinate of p and ϵ_j is the nonzero value of ψ_{e_j} at p. Note that the sum of the right-hand side remains the same between two consecutive endpoints of the projected segments, and the constant values of Ψ over these intervals can be computed, in overall time $O(n \log n)$, by scanning the projected segments from left to right. Hence, we can preprocess $\mathbf{T}$, in time $O(n \log n)$, into a data structure $\mathcal{D}$ so that, for a point p lying below all lines of $\mathcal{L}$, $\Psi(p)$ can be computed in $O(\log n)$ time.

Thus, the observation of [GOS] and Theorem 7.1 imply that by preprocessing $\mathcal{G}_v$ into the above data structure $\mathcal{D}_v$, for each node v of $\mathcal{B}$, the number of triangles in $\mathbf{T}$ containing a query point p can be counted in $O(\sqrt{n} \log^2 n)$ time. But observe that each of the data structures $\mathcal{D}_v$ is a sorted list, and at each node v we do a binary search in $\mathcal{D}_v$ to compute Ψ_v. We can therefore apply the fractional cascading technique of [CGb] to the collection of lists $\mathcal{D}_v$ attached to the nodes v of $\mathcal{B}$. This will allow us to search through the lists $\mathcal{D}_v$ of all nodes $v \in V_{\mathcal{B}}(\ell)$ in overall time $O(\log n + |V_{\mathcal{B}}(\ell)|) = O(\sqrt{n}\log n)$. Hence, we have the following theorem.

THEOREM 8.1. *Given a set* $\mathbf{T}$ *of* n *triangles in the plane, we can preprocess* $\mathbf{T}$, *in time* $O(n^{3/2}\log^{\omega} n)$, *into a data structure of size* $O(n\log^2 n)$ *so that, given a query point* p, *we can determine, in time* $O(\sqrt{n}\log n)$, *the number of triangles in* $\mathbf{T}$ *containing the point* p.

We next establish a trade-off between space and query time for the polygon containment problem. If we allow $O(n^2)$ space, then we can construct the entire arrangement $\mathcal{H}$ of $\bigcup_{e\in\mathcal{G}} B_e$. It is easily seen that the value of Ψ does not change within a face of $\mathcal{H}$, and while constructing $\mathcal{H}$ we can compute Ψ for each of its face. Now given a point p, we can compute $\Psi(p)$ in $O(\log n)$ time by locating p in $\mathcal{H}$. Thus if we allow quadratic storage, the query time can be reduced to $O(\log n)$. Next we give an algorithm for the general case when $n\log^2 n < m < n^2$.

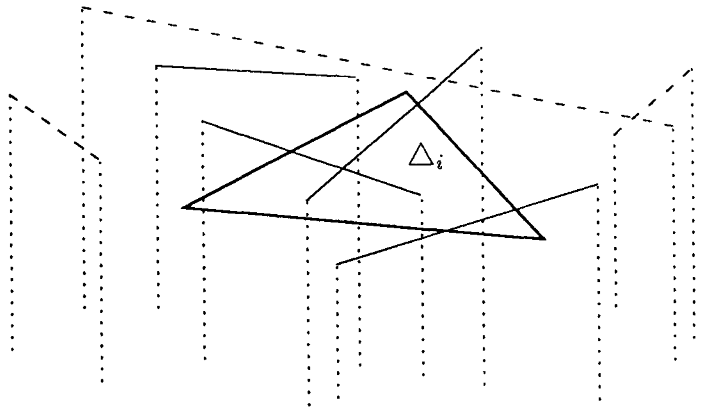

FIG. 14. *Triangle* $\triangle_i$ *and segments of* $\mathcal{G}$: *solid lines are* $\mathcal{G}_i$; *dashed lines are* $\mathcal{G}_i'$.

Let Γ denote the set of lines bounding the trapezoidal strips B_e, that is, the lines containing the segments of $\mathcal{G}$ and the vertical lines passing through the endpoints of segments in $\mathcal{G}$. Partition the plane into $M = O(r^2)$ triangles $\triangle_1, \cdots, \triangle_M$, each meeting at most $\frac{n}{r}$ lines of Γ [Agb]. With each $\triangle_i$ we associate a set $\mathcal{G}_i$ consisting of all segments $e \in \mathcal{G}$ such that either e or one of the two downward-directed vertical rays emanating from its endpoints intersects $\triangle_i$ (see Fig. 14). Let $\mathcal{G}_i' = \mathcal{G} - \mathcal{G}_i$. We can compute $\mathcal{G}_i$, for each i, in total time $O(nr\log n)$. Since $\triangle_i$ does not intersect the boundary of B_e for $e \in \mathcal{G}_i'$, ψ_e remains constant over $\triangle_i$. Moreover, $\sum_{e\in\mathcal{G}_i'}\psi_e$ for every $\triangle_i$ can be computed in $O(nr)$ time, as described in [Agc]. We preprocess $\mathcal{G}_i$ into a data structure of size $O(\frac{n}{r}\log^2\frac{n}{r})$, using the method just mentioned. For answering a query, we first locate the triangle $\triangle_k$ containing the query point p. Once we know $\triangle_k$, $\sum_{e\in\mathcal{G}_k'}\psi_e(p)$ can be determined in $O(1)$ time, and $\sum_{e\in\mathcal{G}_k}\psi_e(p)$ can be computed as described above. Since $|\mathcal{G}_i| = O(\frac{n}{r})$, the query time is

$$Q(n) = O\left(\sqrt{\frac{n}{r}}\log\frac{n}{r} + \log n\right).$$

We need $O(r^2)$ space to store the planar map formed by $\triangle_1, \cdots, \triangle_M$ and $O(\frac{n}{r}\log^2\frac{n}{r})$ to store the data structure constructed for each $\mathcal{G}_i$. Therefore, the total space used is

$$S(n) = O(r^2) + O\left(r^2\cdot\frac{n}{r}\log^2\frac{n}{r}\right)$$

$$= O\left(nr\log^2\frac{n}{r}\right).$$

If we choose $r = (m/n\log^2\frac{n}{\sqrt{m}})$, which is easily seen to satisfy $1 \le r < n$, then $S(n) = O(m)$, and the query time is

$$\begin{aligned} Q(n) &= O\left(\sqrt{\frac{n}{m/(n\log^2\frac{n}{\sqrt{m}})}}\log\frac{n}{\sqrt{m}} + \log n\right) \\ &= O\left(\frac{n}{\sqrt{m}}\log^2\frac{n}{\sqrt{m}} + \log n\right). \end{aligned}$$

Finally, the preprocessing time is

$$\begin{aligned} P(n) &= O(nr\log n\log^{\omega-1} r) + O\left(r^2\cdot\left(\frac{n}{r}\right)^{3/2}\log^{\omega}\frac{n}{r}\right) \\ &= O\left(m\log^{\omega} n + n^{3/2}\sqrt{\frac{m}{n\log^2\frac{n}{\sqrt{m}}}}\log^{\omega}\frac{n}{\sqrt{m}}\right) \\ &= O\left(m\log^{\omega} n + n\sqrt{m}\log^{\omega-1}\frac{n}{\sqrt{m}}\right). \end{aligned}$$

Hence, we can conclude the following theorem.

THEOREM 8.2. *Given a collection* $\mathbf{T}$ *of* n *(possibly intersecting) triangles in the plane, we can preprocess* $\mathbf{T}$*, in time* $O(m\log^{\omega} n + n\sqrt{m}\log^{\omega-1}\frac{n}{\sqrt{m}})$*, into a data structure of size* $O(m)$ *so that, for a query point* p*, we can count the number of triangles of* $\mathbf{T}$ *containing* p *in time* $O(\frac{n}{\sqrt{m}}\log^2\frac{n}{\sqrt{m}} + \log n)$.

Remark 8.3. The batched version of this problem, when all points are given in advance, can be solved, in time $O(m^{2/3}n^{2/3}\log^{2/3} n\log^{\omega/3}\frac{n}{\sqrt{m}} + (m+n)\log n)$, using a different technique [Agc].

8.2. Implicit hidden surface removal. The next problem that we consider is the following version of hidden surface removal problem:

Given a collection of opaque objects in three-dimensional space, and a viewing point a*, we wish to calculate the scene obtained by viewing these objects from* a.

The hidden surface removal problem has been extensively studied by many researchers (see, e.g., [De] and [MK]), because of its applications in graphics and other areas. For the sake of simplicity let us restrict our attention to polyhedral objects, whose boundary $\mathbf{T}$ is a collection $\{\triangle_1, \cdots, \triangle_n\}$ of n nonintersecting triangles. In the case of *implicit* hidden surface removal, we do not want to compute the scene explicitly, but only to preprocess them so as to determine quickly the object seen at any particular query pixel [CS], [GOS]. In this subsection, we consider the following special case of the implicit hidden surface removal problem. Let $\mathbf{T} = \{\triangle_1, \cdots, \triangle_n\}$ be a collection of n nonintersecting triangles such that $\triangle_i$ lies in the plane $z = c_i$, where $0 < c_1 \le c_2 \le \cdots \le c_n$ are some fixed heights. Preprocess $\mathbf{T}$ so that, given a query point p on the xy-plane, one can determine the lowest triangle $\triangle_i$ hit by the upward-directed vertical ray emanating from p.

[GOS] have given an algorithm for this problem that uses randomized processing and has $O(n^{2/3+\delta})$ query time, for any $\delta > 0$. Their algorithm first projects all triangles on the xy-plane, and then performs a binary search through the sequence $(\triangle_1^*, \cdots, \triangle_n^*)$ of projected triangles to find the first index j such that $\triangle_j^*$ contains the query point p. Each step of the binary search tests whether p lies in the union of some contiguous block of projected triangles, using the polygon containment algorithm. Therefore, the preprocessing step consists of constructing a binary tree $\mathcal{Z}$ on $\mathbf{T}$ whose leaves store the triangles of $\mathbf{T}$ in increasing height, and each internal node w is associated with a set of triangles $\mathbf{T}_w$, stored at the leaves of the subtree rooted at w. For each node w of $\mathcal{Z}$, preprocess $\mathbf{T}_w$ for the polygon containment problem, using the algorithm described in §8.1. It now follows from the above discussion that a query can be answered by following a path π in $\mathcal{Z}$ and solving the polygon containment problem at each node of π. Hence using Theorem 8.1, we can conclude with the following theorem.

THEOREM 8.4. *The implicit hidden surface removal problem for an ordered collection of n triangles in three-dimensional space can be solved in $O(\sqrt{n}\log^2 n)$ query time, $O(n\log^3 n)$ space, and $O(n^{3/2}\log^\omega n)$ preprocessing.*

Remark 8.5.

(i) Recently several algorithms for other variants of the implicit hidden surface removal problem have been developed; see [SML], [Be].

(ii) As in the case of the polygon containment problem, the query time can be improved if we allow more space. In particular, if we allow $O(m)$ space, where $n < m < n^2$, then $Q(n) = O(\frac{n}{\sqrt{m}}\log^2\frac{n}{\sqrt{m}} + \log^2 n)$ and $P(n) = O(m\log^\omega n + n\sqrt{m}\log^{\omega-1}\frac{n}{\sqrt{m}})$.

(iii) We can easily modify our algorithm without affecting its time complexity so that the query point p lies anywhere in $\mathbb{R}^3$, rather than lying on the xy-plane. We leave it for the reader to fill in the details.

8.3. Polygon placement problem. Finally consider the following problem:

Let P be a k-gon (not necessarily simple) and let $\mathbf{\Delta} = \{\triangle_1, \cdots, \triangle_n\}$ be a set of n (possibly intersecting) triangles. Preprocess $\mathbf{\Delta}$ so that, given a (translated) placement of P, we can quickly determine whether P intersects any of the obstacles at that placement.

Such a situation arises in several applications [Cha]. A special case, in which P is convex and the triangles are non-intersecting, has been widely studied (see, e.g., [BZ], [CD], [Fo], [LS]). But the best known solution for the general case is by [GOS], who have given an algorithm with randomized preprocessing and $O((kn)^{2/3+\delta})$ query time, for any $\delta > 0$, by reducing this problem to the polygon containment problem. Using their technique, and applying Theorem 8.1, we can easily obtain the following theorem.

THEOREM 8.6. *We can preprocess $\mathbf{\Delta}$ and P, in $O((kn)^{3/2}\log^\omega kn)$ time, into a data structure of size $O(kn\log^2 kn)$ so that, given a translated placement of P, we can determine in time $O(\sqrt{kn}\log kn)$, whether P collides with the obstacles at that placement.*

Remark 8.7. The trade-off between space and query time described in §8.1 works here as well. Therefore, if we allow $O(m)$ space, where $n < m < n^2$, then $Q(n) = O(\frac{kn}{\sqrt{m}}\log^2\frac{kn}{\sqrt{m}} + \log kn)$ and $P(n) = O(m\log^\omega kn + kn\sqrt{m}\log^{\omega-1} kn)$.

9. Conclusions. In this paper we presented efficient algorithms for various problems involving collections of segments in the plane, using spanning trees with low stabbing number. Since the submission of this paper there have been a number of significant developments on these problems. We summarize some of the new results here:

(i) Matoušek has proposed an $O(n^{3/2}\log^2 n)$ algorithm to construct a single spanning tree of a set of n points in $\mathbb{R}^2$ with $O(\sqrt{n})$ stabbing number [Mac]. It immediately improves the space complexity and the preprocessing time of all the algorithms presented here by a factor of $\log n$ and $\log^{\omega-2} n$, respectively.

(ii) Cheng and Janardan [CJ] have shown that a set of n (possibly intersecting) segments can be preprocessed into a data structure of size $O(n \log^3 n)$ so that a ray shooting query can be answered in $O(\sqrt{n} \log n)$ time. Their algorithm is based on spanning trees with low stabbing number and therefore its space complexity and preprocessing time can also be improved by incorporating Matoušek's procedure.

(iii) Using an entirely different approach, Yehuda and Fogel [BF] have designed another ray shooting algorithm for nonintersecting segments that requires $O(n \log n)$ space and supports $O(\sqrt{n} \log n)$ time queries. The preprocessing time of their algorithm is $O(n^{3/2})$. Their algorithm can be extended to intersecting segments using the approach described in §6.

(iv) Another recent development in this area is by Chazelle et al. [CEGGSS], who showed that a polygonal region with k holes can be preprocessed into a data structure of size $O(n \log n)$ so that a ray shooting query can be answered in time $O(\sqrt{k} \log n)$. The preprocessing time of their algorithm is roughly $n^{3/2}$.

(v) A drawback of all these algorithms is that unlike Guibas et al.'s algorithm [GOS] their preprocessing time is not close to linear. Agarwal and Sharir [AS] have shown that the preprocessing can be improved to $O(n^{1+\epsilon})$ without affecting the query time significantly. In particular, their algorithm preprocesses a collection of segments, in time $O(n^{1+\epsilon})$, into a data structure of size $O(n^{1+\epsilon})$, so that a ray shooting query can be answered in $O(n^{1/2+\epsilon})$ time, where ϵ is an arbitrarily small positive constant. Their algorithm relies on a recent partitioning scheme of Chazelle et al. [CSW]. It can be modified to report all k intersections between a collection of n given segments and a query segment in time $O(n^{1/2+\epsilon} + k)$.

(vi) Another shortcoming of the above algorithms is that they do not extend to arbitrary arcs (except the algorithm of [AS]). Some progress in this direction has been made by Agarwal et al. [AKO], who have developed a ray shooting algorithm for nonintersecting Jordan arcs that answers a query in time $O(\sqrt{n} \log^2 n)$ and requires $O(n \log n)$ space.

In spite of these various developments, there are several interesting open problems:

1. The most challenging open problem is to give nontrivial lower bounds for the ray shooting and the implicit point location problems. Recently Chazelle [Chb] showed that if we allow only $O(n)$ space, then a simplex range query (i.e., counting the number of points of a given set contained in a query triangle) requires $\Omega(\sqrt{n})$ time. We conjecture that similar lower bounds hold for these problems as well.
2. Mark Overmars has posed the following problem, which is a generalization of the polygon containment problem: *Given a set* $\mathbf{T}$ *of triangles, preprocess it so that, for a query segment e, one can quickly determine if e is contained in the union of triangles of* $\mathbf{T}$. It will be interesting to come up with an efficient algorithm using spanning trees of low stabbing number.
3. Finally, there remains the task of looking for other interesting problems that can be solved efficiently using the spanning trees of low stabbing number.

Acknowledgments. I thank my advisor Micha Sharir for encouraging me to work on this problem, for several valuable discussions, and for reading earlier versions of this paper which significantly improved its quality. Thanks are also due to three anonymous

referees for several helpful comments and for pointing out an error in the earlier version of the paper.

REFERENCES

[Aga] P. K. AGARWAL, *A deterministic algorithm for partitioning arrangements of lines and its applications*, Proc. 5th Annual Symposium on Computational Geometry, 1989, pp. 11–22.

[Agb] ———, *Partitioning arrangements of lines: I. An efficient deterministic algorithm*, Discrete Comput. Geom., 5 (1990), pp. 449–483.

[Agc] ———, *Partitioning arrangements of lines: II. Applications*, Discrete Comput. Geom., 5 (1990), pp. 533-573.

[AKO] P. AGARWAL, M. VAN KREVALD, AND M. OVERMARS, *Intersection queries for curved objects*, Proc. 7th Annual Symposium on Computational Geometry, 1991, pp. 41–50.

[AS] P. AGARWAL AND M. SHARIR, *Applications of a new partitioning scheme*, Proc. 2nd Workshop on Algorithms and Data Structures, 1991, pp. 379–391; Discrete Comput. Geom., to appear.

[BF] R. BAR YEHUDA AND S. FOGEL, *Good splitters with applications to ray shooting*, Proc. 2nd Canadian Conf. on Computational Geometry, 1990, pp. 81–85.

[Be] M. BERN, *Hidden surface removal for rectangles*, J. Comput. Systems Sci., 40 (1990), pp. 49–69.

[BZ] B. 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.

[Cha] B. CHAZELLE, *The polygon containment problem*, in Advances in Computing Research, Vol. I: Computational Geometry, F. P. Preparata, ed., JAI Press, Greenwich, CT, 1983, pp. 1–33.

[Chb] ———, *Lower bounds on the complexity of polytope range searching*, J. Amer. Math. Soc., 2 (1989), pp. 637–666.

[Chc] ———, *Tight bounds on the stabbing number of trees in Euclidean plane*, Tech. Report CS-TR-155-58, Dept. Computer Science, Princeton University, Princeton, NJ, May 1988.

[Chd] ———, Private communication, 1989.

[CEGGSS] B. CHAZELLE, H. EDELSBRUNNER, M. GRIGNI, L. GUIBAS, M. SHARIR, AND J. SNOEYINK, *Ray shooting in polygons using geodesic triangulations*, Proc. 18th Int. Coll. on Automata, Languages and Programming, 1991.

[CGb] B. CHAZELLE AND L. GUIBAS, *Fractional cascading: I. A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.

[CGc] ———, *Fractional cascading: II. Applications*, Algorithmica, 1 (1986), pp. 163–191.

[CGa] ———, *Visibility and intersection problems in plane geometry*, Discrete Comput. Geom., 4 (1989), pp. 551–581.

[CGL] B. CHAZELLE, L. GUIBAS, AND D. T. LEE, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.

[CSW] B. CHAZELLE, M. SHARIR, AND E. WELZL, *Quasi optimal upper bounds for simplex range searching and new zone theorem*, Proc. 6th Annual Symposium on Computational Geometry, 1990, pp. 23–33.

[CW] B. CHAZELLE AND E. WELZL, *Quasi optimal range searching in spaces with finite* VC-*dimensions*, Discrete Comput. Geom., 4 (1989), pp. 467–489.

[CD] P. CHEW AND L. DRYSDALE, *Voronoi diagrams based on convex distance functions*, Proc. 1st Annual Symposium on Computational Geometry, 1985, pp. 235–244.

[CJ] S. CHENG AND R. JANARDAN, *Space-efficient ray shooting and intersection searching: Algorithms, dynamization, and applications*, Second SIAM-ACM Symposium on Discrete Algorithms, 1991, pp. 7–16.

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

[Co] R. COLE, *Searching and storing similar lists*, J. Algorithms, 7 (1986), pp. 202–220.

[CS] R. COLE AND M. SHARIR, *Visibility problems for polyhedral terrains*, J. Symbolic Comput., 7 (1989), pp. 11–30.

[De] F. DÉVAI, *Quadratic bounds for hidden line elimination*, Proc. 2nd Annual Symposium 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.

[DSST] J. DRISCOLL, N. SARNAK, D. SLEATOR, AND R. TARJAN, *Making data structures persistent*, J. Comput. Systems Sci., 38 (1989), pp. 86–124.

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

[EG] H. Edelsbrunner and L. Guibas, *Topologically sweeping an arrangement*, J. Comput. Systems Sci., 38 (1989), pp. 165–194.

[EGH*] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl, *Implicitly representing arrangements of lines or segments*, Discrete Comput. Geom., 4 (1989), pp. 433–466.

[EGP*] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir, *Arrangements of curves in the plane—topology, combinatorics, and algorithms*, Theoret. Comput. Sci., 92 (1992), pp. 319–336.

[EGS] H. Edelsbrunner, L. Guibas, and G. Stolfi, *Optimal point location in monotone subdivisions*, SIAM J. Comput., 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.

[Fo] S. Fortune, *Fast algorithms for polygon containment*, Proc. 12th International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci., 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 shortest path and visibility problems*, Algorithmica, 2 (1987) pp. 209–233.

[GOS] L. Guibas, M. Overmars, and M. Sharir, *Ray shooting, implicit point location, and related queries in arrangements of segments*, Tech. Report 433, Dept. Computer Science, New York University, New York, March 1989.

[GSS] L. Guibas, M. Sharir, and S. Sifrony, *On the general motion planning problem with two degrees of freedom*, Discrete Comput. Geom., 4 (1989), pp. 491–521.

[GY] L. Guibas and F. Yao, *On translating a set of rectangles*, in Advances in Computer Research, Vol. I: Computational Geometry, F. P. Preparata, ed., JAI Press, Greenwich, CT, 1983, pp. 61–77.

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

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

[LS] D. Lenen 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.

[Maa] J. Matoušek, *Construction of ϵ-nets*, Discrete Comput. Geom., 5 (1990), pp. 427–448.

[Mab] ———, *Spanning trees with low crossing numbers*, Inform. Théoret. Appl., 25 (1991), pp. 103–123.

[Mac] ———, *More on cutting arrangements and spanning trees with low crossing number*, Tech. Report B-90-2, Department of Mathematics, Freie Universität, Berlin, February 1990.

[PS] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, Heidelberg, 1985.

[SML] A. Schmitt, H. Muller, and W. Leister, *Ray tracing algorithm—Theory and practice*, in Theoretical Foundations of Computer Graphics and CAD, R. Earnshaw, ed., NATO ASI Series, Vol. F-40, Springer-Verlag, New York, 1988, pp. 997–1030.

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

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

[We] E. Welzl, *Partition trees for triangle counting and other range searching problems*, Proc. 4th Annual Symposium on Computational Geometry, 1988, pp. 23–33.

[Ya] C. K. Yap, *An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments*, Discrete Comput. Geom., 2 (1987), pp. 365–393.