RAY SHOOTING AND PARAMETRIC SEARCH*

PANKAJ K. AGARWAL[†] AND JIŘÍ MATOUŠEK[‡]

Abstract. Efficient algorithms for the *ray shooting* problem are presented: Given a collection Γ of objects in $\mathbb{R}^d$, build a data structure so that, for a query ray, the first object of Γ hit by the ray can be quickly determined. Using the parametric search technique, this problem is reduced to the *segment emptiness* problem. For various ray shooting problems, space/query-time trade-offs of the following type are achieved: For some integer b and a parameter m ($n \leq m \leq n^b$) the queries are answered in time $O((n/m^{1/b}) \log^{O(1)} n)$, with $O(m^{1+\varepsilon})$ space and preprocessing time ($\varepsilon > 0$ is arbitrarily small but fixed constant). $b = \lfloor d/2 \rfloor$ is obtained for ray shooting in a convex d -polytope defined as an intersection of n half spaces, $b = d$ for an arrangement of n hyperplanes in $\mathbb{R}^d$, and $b = 3$ for an arrangement of n half planes in $\mathbb{R}^3$. This approach also yields fast procedures for finding the first k objects hit by a query ray, for searching nearest and farthest neighbors, and for the hidden surface removal. All the data structures can be maintained dynamically in amortized time $O(m^{1+\varepsilon}/n)$ per insert/delete operation.

Key words. ray shooting, arrangements, convex polytope, range searching, parametric search

AMS subject classifications. 52B11, 68P05, 68Q20, 68Q25, 68U05

1. Introduction. Consider the following *ray shooting* problem: Given a collection Γ of n objects in $\mathbb{R}^d$, build a data structure so that, for a query ray ρ , we can quickly determine the first object of Γ intersected by ρ .

The ray shooting problem has received much attention in the past few years because of its applications in graphics and other geometric problems [11], [20], [1], [13], [6], [4], [9], [7]. But most of the work done so far has been for the planar case where Γ is a collection of line segments in $\mathbb{R}^2$. Chazelle and Guibas proposed an optimal algorithm for the special case where Γ is the boundary of a simple polygon [11]. Their algorithm answers a ray shooting query in $O(\log n)$ time using $O(n)$ space (see also [9]). If Γ is a collection of arbitrary segments in the plane, the best known algorithm answers a ray shooting query in time $O(\frac{n}{\sqrt{m}} \log^{O(1)} n)$ using $O(m^{1+\varepsilon})$ space and preprocessing¹ [1], [6], [4]. Although no lower bound is known for this case, it is conjectured that this bound is close to optimal. However, in three and higher dimensions the problem is far from being solved. For example, no efficient ray shooting algorithm is known for a convex d -polytope for $d > 3$. Even in $\mathbb{R}^3$, nontrivial solutions have been obtained only very recently (cf. [4], [7]).

In this paper, we present ray shooting algorithms for several cases in higher dimensions ($d \geq 3$), including a convex polytope, a collection of n hyperplanes in $\mathbb{R}^d$, and a collection of n half planes in $\mathbb{R}^3$. We will use a unified approach for all cases, which is, roughly speaking, a binary search along the query ray ρ . In order to make this approach work, we need to handle two problems: (i) Perform a binary search without computing all intersection points of ρ and the objects of Γ , and (ii) given a point $x \in \rho$, determine whether the first intersection point of Γ and ρ lies before or after x .

To handle the first problem we perform an implicit binary search using the *parametric search* technique of Megiddo [26], and, to handle the second problem, we use suitable range searching structures for detecting an intersection between ρ and the objects of Γ . Our approach can be easily extended to report the first k objects hit by the query ray.

*Received by the editors June 10, 1991; accepted for publication (in revised form) May 20, 1992. Work by the first author has been supported by National Science Foundation grant CCR-91-06514. Part of the work was done while the second author was visiting School of Mathematical Sciences, Georgia Tech University, Atlanta, Georgia.

[†]Computer Science Department, Duke University, Durham, North Carolina 27706 (pankaj@cs.duke.edu).

[‡]Department of Applied Mathematics, Charles University, Prague, Czech.

¹Throughout this paper, ε denotes an arbitrarily small positive constant. The multiplicative constants in the asymptotic bounds may depend on ε .

The range searching algorithms we have at our disposal usually admit space/query-time trade-offs: the more space (and preprocessing time) we use, the faster the queries can be answered. Such a trade-off then transfers to the ray shooting results. A usual form of this trade-off is the following: There are two fixed integers b, c (specific to the considered problem), such that with $O(m^{1+\varepsilon})$ space and preprocessing time, where $n \leq m \leq n^b$, a query can be answered in time $O(\frac{n}{m^{1/b}} \log^c n)$ [12]. Note that since we require $m \geq n$, the smallest amount of space we consider in this trade-off is $O(n^{1+\varepsilon})$. Hence the $\log^c n$ factor plays a role only when m is close to n^b ; actually it expresses the query complexity for the maximum permissible amount of space. For the sake of brevity, we just say that such a problem admits a *standard trade-off* with a certain value of b . It is understood that c is always a (reasonably small) constant. For the sake of simplicity, we will not discuss the best value of c . Also, in many cases the space and/or preprocessing time can be somewhat reduced because of improved results in range searching (e.g., for $m = n$, or $m = n^b$); we will not pay too much attention to this either.

In this paper, we consider the following specific ray shooting problems:

(i) Ray shooting in an arrangement of n hyperplanes in $\mathbb{R}^d$. We get a standard trade-off with $b = d$. Previously, efficient solutions were known only for $d \leq 3$. Moreover, they did not support insertion/deletion of planes for $d = 3$.

(ii) Ray shooting in a convex d -polytope defined as the intersection of n half spaces. We get a standard trade-off with $b = \lfloor d/2 \rfloor$. Previously, nontrivial solutions were known only for $d \leq 3$ [17].

(iii) Ray shooting in an arrangement of n half planes in $\mathbb{R}^3$. We obtain a standard trade-off with $b = 3$. The previously best known result answered a query in time $O(n^{16/15+\varepsilon}/m^{4/15})$ using $O(m^{1+\varepsilon})$ space and preprocessing [4].

By a well-known reduction, a suitable ray shooting algorithm yields a procedure that can process a set of n points in $\mathbb{R}^d$ into a data structure, so that the nearest or the farthest neighbor of a query point can be computed quickly; we get a standard trade-off with $b = \lceil d/2 \rceil$ [18]. It can also report the first k neighbors, for any given $k \leq n$, at an additional cost of $k \log^2 n$ in the query time.

Finally, we consider the following hidden surface removal problem: Given a set T of n nonintersecting triangles in $\mathbb{R}^3$ and a view point $z = +\infty$, compute the *visibility map* of T , that is, the subdivision of the viewing plane so that the same triangle is visible from all the points of a face. The goal is to come up with an “output-sensitive” algorithm. Although there have been several output-sensitive algorithms for special cases, only very recently de Berg et al. [7] presented an output-sensitive algorithm for the general case. Their algorithm relies on a fast procedure for ray shooting among a collection of “curtains”; a *curtain* is an unbounded vertical triangle with two of its edges being vertical rays extending to $-\infty$. We shall present a faster procedure for ray shooting among curtains, which will improve the running time of the hidden surface removal algorithm of [7].

Tables 1 and 2 offer a succinct comparison of our ray shooting and k -intersection results with the previously known ones. In these tables, the space and preprocessing time are usually $O(m^{1+\varepsilon})$ (and better in some cases; see the appropriate sections).

TABLE 1
Ray shooting results.

Objects	Previous results	New results
d -polytope	$O(\log n)$, $d \leq 3$ [17]	$O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$
hyperplanes in $\mathbb{R}^d$	$O(\frac{n^{1+\varepsilon}}{m^{1/d}})$, $d \leq 3$ [4]	$O(\frac{n}{m^{1/d}} \log^4 n)$
half planes in $\mathbb{R}^3$	$O(\frac{n^{16/15+\varepsilon}}{m^{4/15}})$ [4]	$O(\frac{n}{m^{1/3}} \log^4 n)$
curtains in $\mathbb{R}^3$	$O(\frac{n^{2+\varepsilon}}{m})$ [7]	$O(\frac{n}{\sqrt{m}} \log^6 n)$

TABLE 2
k-intersection results.

Objects	Previous results	New results
hyperplanes in $\mathbb{R}^d$	None	$O(\frac{n}{m^{1/d}} \log^4 n + k)$
segments in $\mathbb{R}^2$	$O(\frac{n}{\sqrt{m}} \log^{O(1)} n + k \log n)$	$O(\frac{n}{\sqrt{m}} \log^{O(1)} n + k)$
triangles in $\mathbb{R}^3$	None	$O(\frac{n^{6/15}}{m^{4/15}} \log^{O(1)} n + k)$

This paper is organized as follows. Section 2 introduces the ray shooting problem in a general setting and gives a reduction to the segment emptiness problem using Megiddo's parametric search technique. In §3, we discuss specific applications of this general result to ray shooting and related problems in various situations. Ray shooting algorithms are applied to proximity problems and to the hidden surface removal problem in §4 and §5, respectively. We finally conclude by mentioning some open problems.

2. Ray shooting using parametric search. It will be convenient to formulate the ray shooting problem in a reasonably general setting. Let $\mathcal{G}$ be a class of (topologically closed) geometric objects in $\mathbb{R}^d$ (in the examples we consider, these will be hyperplanes or parts of them), and let Γ be some set of n objects of $\mathcal{G}$. Further let $\mathcal{R}$ be a set of *admissible rays*. Let $o(\rho)$ denote the origin point of a ray ρ . The points of every ray will be ordered increasingly along the ray starting from its origin, i.e., for $p, q \in \rho$ we say $p < q$ if $o(\rho)$ is closer to p than to q . For a ray $\rho \in \mathcal{R}$ and $g \in \Gamma$ let $\varphi(g, \rho)$ denote the first point of g hit by ρ if it exists; otherwise $\varphi(g, \rho) = +\infty$. We set $\varphi(\Gamma, \rho) = \min_{g \in \Gamma} \varphi(g, \rho)$. We want to build a data structure that, given a ray $\rho \in \mathcal{R}$, computes $\varphi(\Gamma, \rho)$ quickly, together with a $g \in \Gamma$ such that $\varphi(g, \rho) = \varphi(\Gamma, \rho)$. Abusing the notation slightly, we shall use $\varphi(\Gamma, \rho)$ to denote the first intersection point as well as the object that contains the intersection point.

Let $\text{Seg}(\mathcal{R})$ denote the set of all *initial segments* of the rays of $\mathcal{R}$, i.e.,

$$\text{Seg}(\mathcal{R}) = \{o(\rho)x; \rho \in \mathcal{R}, x \in \rho\}.$$

Suppose that we have an efficient algorithm that, given a segment $ox \in \text{Seg}(\mathcal{R})$, decides whether it intersects any object $g \in \Gamma$. We refer to this procedure as the *segment emptiness* algorithm. We also assume that the algorithm can detect the case when an initial segment ox intersects Γ only at x and that it can identify the intersected object in this case.

Observe that, given a point x on a ray $\rho \in \mathcal{R}$, we can use the segment emptiness algorithm to decide the relative order of x and $\varphi(\Gamma, \rho)$: if the segment $o(\rho)x$ intersects Γ only at x , then $x = \varphi(\Gamma, \rho)$; if ox is empty then $x < \varphi(\Gamma, \rho)$; otherwise, $x > \varphi(\Gamma, \rho)$. As it is often the case in similar situations, the *parametric search technique* due to Megiddo [26] can be used to turn this “verification” algorithm into a “searching” algorithm. Let us outline this technique applied to our specific problem.

Let A be a segment emptiness algorithm. Let $\mathbf{v}$ be the unit direction vector of ρ and let $\{x(t) = o(\rho) + t\mathbf{v}; t \in \mathbb{R}^+\}$ be a parametric representation of ρ . Let t^* denote the (yet unknown) value of the parameter t such that $x(t^*) = \varphi(\Gamma, \rho)$. The first idea of the parametric search technique is to run the algorithm A to decide the emptiness of the segment $o(\rho)x(t^*)$, and to run it “generically,” without specifying the value of t^* . The execution of the algorithm A will, sooner or later, need some information about t^* . As observed earlier, we can gain some information about t^* : we can compare it with some given t , by running the segment emptiness algorithm on the segment $o(\rho)x(t)$ (for a reason which becomes apparent later, we shall think of this algorithm as another “template” of A and call it B).

Specifically, assume that the flow of execution of A depends on comparisons, each of which involves testing the sign of a low-degree polynomial in t whose coefficients may depend

on ρ , on the objects of Γ but not on t^* . We maintain an interval I , which is either a singleton or an open interval that contains t^* . Each time a comparison is to be made, the few roots of the associated polynomial are computed, and we run the algorithm B “off line” at each of them. If one of the roots is t^* itself, we can stop; otherwise we determine the location of t^* among these roots, and thus also the sign of the polynomial at t^* . If we know the two consecutive roots β_i, β_{i+1} such that $t^* \in (\beta_i, \beta_{i+1})$, we can compute the sign of the polynomial at t^* , i.e., the outcome of the comparison at t^* . We now set I to $I \cap (\beta_i, \beta_{i+1})$. If $I = \emptyset$, we can conclude that t^* does not exist, and we stop. Otherwise we resume the execution of the generic algorithm A . As we proceed through this execution, each comparison that we resolve further constrains the range where t^* can lie, and we thus obtain a sequence of progressively smaller intervals, each known to contain t^* , until we either reach the end of A or hit t^* at one of the comparisons at A . Since the outcome of A changes at t^* , it can be shown that t^* is a root of a polynomial associated with one of the comparisons resolved by A (see [3] for a proof). This in turn implies that the computation will stop at the desired value t^* .

The most expensive steps in this computation are calls to the subroutine B for resolving comparisons. To reduce this cost, Megiddo suggests replacing the sequential algorithm A by its parallel version, A_p . If A_p uses p processors and runs in T_A parallel steps, then each such step involves at most p independent comparisons, that is, each can be carried out without having to know the outcome of the others. We can then compute the roots of all p polynomials associated with these comparisons, and perform a binary search to locate t^* among them (using the algorithm B at each binary search step). If T_B is the time complexity of the subroutine B , then the binary search requires $O(p + T_B \log p)$ time per parallel step, for a total of $O(pT_A + T_B T_A \log p)$ time for a sequential algorithm that simulates A_p . An improvement of this technique by Cole [15] can further reduce the running time in certain cases by another logarithmic factor (this, however, depends on the specific algorithm A_p).

Let us summarize our discussion in a (rather long) theorem.

THEOREM 2.1. *Let Γ be a set of objects and $\mathcal{R}$ a collection of rays. Suppose that we have a data structure Σ supporting segment emptiness queries with respect to Γ for the segments of $\text{Seg}(\mathcal{R})$. Let A_p be a parallel algorithm for answering a segment emptiness query, which uses p processors and runs in T_A parallel steps², and such that for a query segment ox , the computation of A_p uses the information about x only in deciding the signs of certain fixed-degree polynomials in the coordinates of x . Let B be another version of segment emptiness algorithm, which can report an object $g \in \Gamma$ intersecting the endpoint of the query segment, provided that the segment is otherwise empty, and let T_B be the running time of B . Then the ray shooting problem for rays in $\mathcal{R}$ can be solved using the same data structure Σ , in time $O(pT_A + T_B T_A \log p)$.*

This approach can obviously be extended to find the first k objects of Γ intersected by the query ray. In this case, the generic algorithm A_p should decide whether the query segment $o(\rho)x(t^*)$ intersects exactly k objects of Γ , and algorithm B decides whether the number of the objects of Γ intersected by query segment are fewer than, equal to, or more than k . After having computed the value of t^* , the answer (the first k objects hit by ρ) can be computed by a segment range reporting algorithm C ; usually a variant of A or B gives such an algorithm. The running time of the resulting algorithm will then be $O(pT_A + T_B T_A \log p + T_C)$, where T_A , T_B , and p have the same meaning as above and T_C is the running time of C (T_C may depend on k).

3. Specific results on ray shooting. In this section we apply the general technique described in the previous section to obtain fast solutions for some specific instances. In particular,

²Actually we count only the steps with comparisons involving x .

we present ray shooting algorithms in an arrangement of hyperplanes in d -dimensions (§3.1), a collection of half spaces (§3.2), and an arrangement of half planes in $\mathbb{R}^3$ (§3.3).

3.1. Ray shooting among hyperplanes. In this section we describe an efficient ray shooting algorithm for a collection H of n hyperplanes in $\mathbb{R}^d$. For $d \leq 3$, Agarwal and Sharir [4] gave an algorithm that gives a standard trade-off for ray shooting among hyperplanes with $b = d$. In this section we obtain similar bounds for higher dimensions. In view of Theorem 2.1, it suffices to describe an efficient procedure for the segment emptiness problem.

The dual of a hyperplane (respectively, segment) in $\mathbb{R}^d$ is a point (respectively, double wedge),³ and a segment e intersects a hyperplane h if and only if the double wedge e^* contains the point h^* . Therefore, the segment emptiness problem for H is the same as detecting whether a query double wedge contains any point of H^* . This problem is a special case of the *simplex range searching problem*, where we want to report or count the points contained in a query simplex.

Chazelle, Sharir, and Welzl [12] showed that, using $O(m^{1+\epsilon})$, ($n \leq m \leq n^d$) space and preprocessing, we can answer a double wedge range query in time $O(\frac{n}{m^{1/d}} \log^2 n)$ (see Appendix). A simpler algorithm (with improved performance for the case of linear space) was given in [22]. More specific properties of this algorithm will be briefly described in the Appendix. With a knowledge of this data structure, it is straightforward to check that the algorithm can be run in $O(\log n)$ parallel steps using $O(\frac{n}{m^{1/d}} \log n)$ processors. Also, it is shown in [22] that the data structure can be maintained dynamically under insertions and deletions of points. Hence, in view of Theorem 2.1, we can conclude the following theorem.

THEOREM 3.1. *Given a set H of n hyperplanes in $\mathbb{R}^d$, a parameter m , $n \leq m \leq n^d$, we can build, in time $O(m^{1+\epsilon})$, a data structure of size $O(m^{1+\epsilon})$ that supports ray shooting queries in time $O(\frac{n}{m^{1/d}} \log^4 n)$. The data structure can be maintained dynamically, as hyperplanes are being inserted in or deleted from H , in $O(m^{1+\epsilon}/n)$ amortized time per update operation.*

Remark 3.2. With a linear space and $O(n \log n)$ preprocessing time, we actually get query time $O(n^{1-1/d} \log^{O(1)} n)$. On the other hand, with $O(n^{d+\epsilon})$ space and preprocessing, the query time becomes $O(\log^2 n)$. This and other possibilities of small improvements follow from more specific results of [12], [22].

In the above result, the simplex range searching results were applied only to decide the emptiness of a query segment. However, these algorithms can also count the number of hyperplanes intersecting a query segment within the same time bound, or can report all such k hyperplanes at an additional cost of $O(k)$. The remarks at the end of §2 thus imply the following.

THEOREM 3.3. *Given a set H of n hyperplanes in $\mathbb{R}^d$, a parameter m , $n \leq m \leq n^d$, we can build, in time $O(m^{1+\epsilon})$, a data structure so that, the first k hyperplanes of H intersected by a query ray can be reported in time $O(\frac{n}{m^{1/d}} \log^4 n + k)$. This data structure can be maintained dynamically, as hyperplanes are being inserted in or deleted from H , in $O(m^{1+\epsilon}/n)$ amortized time per update operation.*

Agarwal and Sharir [4] showed that if Γ is a set of n segments in $\mathbb{R}^2$ or a set of n triangles in $\mathbb{R}^3$, we can count the number of objects in Γ intersected by a query segment in time $O(n^{1+\epsilon}/\sqrt{m})$ or $O(n^{16/15+\epsilon}/m^{4/15})$, respectively, using $O(m^{1+\epsilon})$ space and preprocessing, where $n \leq m \leq n^2$ and $n \leq m \leq n^4$, respectively. Hence, we obtain similar bounds for reporting the first k objects of Γ hit by a query ray. A further improvement is possible using recent results of Aronov and Sharir [5] on the complexity of a zone of an algebraic surface in a hyperplane arrangement; we will not elaborate on this in this paper.

³Throughout this paper, we will denote by γ^* the dual of an object γ , and by Γ^* the set $\{\gamma^*; \gamma \in \Gamma\}$.

3.2. Ray shooting in a convex polytope. In this section we consider the ray shooting problem for a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$. We assume that $\mathcal{P}$ is described as the intersection of n half spaces in $\mathbb{R}^d$. Let H denote the set of hyperplanes bounding these half spaces. For $d = 2$ a straightforward binary search can answer a ray shooting query in $O(\log n)$ time, and for $d = 3$ we can use the Dobkin–Kirkpatrick hierarchical representation of a 3 polytope to obtain an optimal algorithm [17]. But for $d > 3$, the Dobkin–Kirkpatrick hierarchical representation does not work, and no efficient algorithm is known for ray shooting in higher dimensions. Even in $\mathbb{R}^3$ no efficient ray shooting algorithm is known if the polytope $\mathcal{P}$ changes dynamically.

We first describe the algorithm for the special case when o , the origin point of the ray, lies inside $\mathcal{P}$. This is simpler and sufficient in many applications. After a suitable projective transformation, we can assume that $\mathcal{P}$ is the region lying above all hyperplanes of H . (Actually, the algorithm can be modified suitably, so that it works even if $\mathcal{P}$ does not lie above all hyperplanes of H .) By Theorem 2.1, we are interested in a data structure that, for a query segment ox , detects whether ox intersects the boundary of $\mathcal{P}$, which, by our assumption, is equivalent to whether there is a hyperplane of H lying above x . In the dual setting, this means that the half space lying above the hyperplane x^* contains at least one point of H^* .

We thus want to preprocess H^* for *half-space emptiness* queries, i.e., we need a data structure deciding whether there is a point of H^* in a query half space. In [23] it is shown that the half-space emptiness queries can be answered in time $O(\frac{n}{m^{1/d/2}} \log n)$ using $O(m^{1+\varepsilon})$ space and preprocessing (the algorithm includes the data structure due to Clarkson [14] for the “large space” case). It also allows the insertion or deletion of a half space in $O(m^{1+\varepsilon}/n)$ amortized time, see, e.g., [2] for details. A parallel implementation with $O(\log n)$ time and number of processors bounded by the sequential running time is quite straightforward. Furthermore, the algorithm can also detect the case when the interior of the query half space is empty but its boundary contains a point (see original papers [14], [23] for details). We thus have the following lemma.

LEMMA 3.4. *Given a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$, described as the intersection of n half spaces, and a parameter m , $n \leq m \leq n^{\lfloor d/2 \rfloor}$, we can construct in time $O(m^{1+\varepsilon})$ a data structure of size $O(m^{1+\varepsilon})$ so that, for a query ray whose origin point lies inside $\mathcal{P}$, we can determine the first point of the polytope boundary hit by the ray in $O(\frac{n}{m^{1/d/2}} \log^3 n)$ time. This data structure can be maintained dynamically, as half spaces are being inserted in or deleted from $\mathcal{P}$, in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

This result has one surprising consequence. Namely, the half-space emptiness algorithm of [23] sometimes concludes that the query half space is nonempty, but does not exhibit a “witness point” contained in the query half space (which may be required in some applications). Our ray shooting result allows us to obtain such a witness: Let h denote the hyperplane bounding the query half space and, without loss of generality, assume that the query half space lies above h . In the dual setting, we thus want to return a hyperplane of H lying above the point h^* if it exists. We shoot a vertical ray in the $-x_d$ direction from the point $(h_1^*, \dots, h_{d-1}^*, +\infty)$ (where $(h_1^*, \dots, h_d^*)$ is the coordinate vector of h^*). If h^* lies below the first hyperplane hit by the ray, then that hyperplane is the desired witness; otherwise h^* lies above all hyperplanes of H . Hence we have the following corollary.

COROLLARY 3.5. *Given a set S of n points in $\mathbb{R}^d$, we can construct in time $O(m^{1+\varepsilon})$ ($n \leq m \leq n^{\lfloor d/2 \rfloor}$) a data structure of size $O(m^{1+\varepsilon})$, such that, for a query half space γ , we can determine in time $O(\frac{n}{m^{1/d/2}} \log^3 n)$ a point of $\gamma \cap S$ or conclude that $\gamma \cap S = \emptyset$.*

Next, we extend the ray shooting algorithm for the case when the origin point o of ρ does not lie in $\mathcal{P}$. Note that this does not quite fall into our general framework (at least not if we take the set of hyperplanes for Γ), so we must exhibit a specific (although very similar) solution using parametric search. It suffices to find a point $x^* = x(t^*)$ of the query ray inside $\mathcal{P}$ if it

exists (then the previous ray shooting result can be applied); in our setting this means a point x^* of the ray lying above all hyperplanes of H . For this problem, the generic algorithm A will check whether $x(t) \in \mathcal{P}$. The oracle B should decide on which side of a point $x = x(t)$ the potential intersection of the query ray with $\mathcal{P}$ lies. We let B be the algorithm dual to the one from Corollary 3.5, i.e., it checks whether $x \in \mathcal{P}$ (if *yes* the computation may finish), and if not, it exhibits a hyperplane $h \in H$ lying above x . The crucial observation is that at least one of the two portions of the query ray determined by $x(t)$ also lies below h , and therefore $\rho \cap \mathcal{P}$ is bound to lie in the other portion (provided it exists at all). As a result the algorithm B can still resolve comparisons. Clearly, if ρ intersects $\mathcal{P}$, a point in $\rho \cap \mathcal{P}$ will be found. On the other hand, if ρ does not intersect $\mathcal{P}$, the answers given by B will become inconsistent (i.e., the interval for t^* becomes empty). By Corollary 3.5 and Theorem 2.1, we can conclude the following theorem.

THEOREM 3.6. *Given a convex polytope $\mathcal{P}$ in $\mathbb{R}^d$ described as the intersection of n half spaces, and a parameter m ($n \leq m \leq n^{\lfloor d/2 \rfloor}$), we can preprocess $\mathcal{P}$ in time $O(m^{1+\epsilon})$ into a data structure of size $O(m^{1+\epsilon})$, so that for a query ray the first point of $\mathcal{P}$ hit by the ray can be determined in $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^5 n)$ time. The data structure can be maintained dynamically, as half spaces are being inserted in or deleted from $\mathcal{P}$, in amortized time $O(m^{1+\epsilon}/n)$ per update operation.*

Remark 3.7. This algorithm can also be improved in some special cases. For instance, if we do not require a dynamic structure, we may apply a result of [23]: Given $O(n)$ space and $O(n^{1+\epsilon})$ preprocessing time, we can answer half space emptiness queries in $O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)})$ time; the algorithm can be implemented with $O(\log \log n)$ parallel steps. Similarly, if we allow $O(n^{L_{d/2}+\epsilon})$ space, the query time in the above theorem can be improved to $O(\log^2 n)$.

3.3. Ray shooting among half planes. In this section we consider the case when Γ is a set of half planes in $\mathbb{R}^3$. Although a ray shooting query among planes in $\mathbb{R}^3$ can be answered in roughly $n/m^{1/3}$ time, no such procedure is known for half planes. The query time of the best known algorithm for half spaces is close to $n^{16/15}/m^{4/15}$, which is the same as the query time for ray shooting among triangles in $\mathbb{R}^3$ [4]. Here we improve the query time to $O(\frac{n}{m^{1/3}} \log^{O(1)} n)$; as usual, we will describe an efficient procedure for the segment emptiness problem. We use a “multilevel” range searching structure tailored to this specific application. General principles of building such structures are outlined in the Appendix.

Let H denote the set of planes supporting the half planes of Γ . We construct a partition tree $\mathcal{T}$ on the set of points in H^* . For a node v of $\mathcal{T}$, let Γ_v be the set of half planes corresponding to the points of the canonical subset associated with v , and let L_v be the set of lines bounding the half planes of Γ_v . We orient each line ℓ of L_v so that the half plane bounded by ℓ lies to its right (i.e., in the clockwise direction). L_v is thus a set of oriented lines.

As a secondary data structure in each node v , we construct a data structure for deciding whether a query line ℓ (in our application, the line carrying the query segment) has positive (or negative) orientation with respect to all lines of L_v . Chazelle et al. [10] give a data structure for this problem with space and preprocessing time $O(n^{2+\epsilon})$ and with $O(\log n)$ query time. They actually reduce the problem to answering a half-space emptiness query in five dimensions⁴. Combining their reduction with the already mentioned results of [23] for

⁴The reduction uses so-called Plucker coordinates of lines. Every line ℓ' of the set L_v is mapped to a Plucker point $\pi(\ell')$ in projective five-space, and a query line ℓ is mapped to a Plucker hyperplane $\varpi(\ell)$ in projective five-space. The query line ℓ has positive (respectively, negative) orientation with respect to all lines in L_v if and only if the hyperplane $\varpi(\ell)$ lies above (respectively, below) the Plucker points of all lines in L_v , see [29], [10] for details. The problem thus reduces to determining whether the half space lying above (respectively, below) $\varpi(\ell)$ is empty, which can be done using the data structure of Clarkson [14].

the half-space emptiness problem, we get a data structure for the segment emptiness problem (with respect to half planes in three-dimensions) that admits a standard trade-off with $b = 2$.

When answering a segment emptiness query for a segment $e = pq$ with this two-level data structure, we first query the first level structure with the double wedge e^* dual to e . It gives the set of half planes, whose supporting planes intersect e , as a pairwise disjoint union of $O(\frac{n}{m^{1/3}} \log n)$ canonical subsets. Let Γ_v be a first-level canonical subset of the query output. Then either p lies below all the planes containing the half planes of Γ_v and q lies above all of them, or vice versa. Without loss of generality, assume that p lies below all of them.

The query segment e intersects a half plane $\gamma \in \Gamma_v$ if and only if ℓ , the line supporting e and oriented from p to q , intersects γ . If the xy -projection of ℓ is in clockwise (respectively, counterclockwise) direction to the xy -projection of the line $\partial\gamma$, then ℓ intersects γ if and only if ℓ lies below (respectively, above) $\partial\gamma$, i.e., ℓ has negative orientation with respect to $\partial\gamma$ (see Fig. 1). In other words, ℓ does not intersect any half plane of Γ_v if and only if ℓ has positive orientation with respect to all lines of L_v . Similarly, if p lies above all the planes containing the half planes of Γ_v , then e does not intersect any half plane of Γ_v if and only if ℓ has negative orientation with respect to all lines of L_v . We can perform this test using the secondary structure stored at v . By repeating this step for all first-level canonical subsets of the query output, we can answer the emptiness query. This data structure gives a standard trade-off with $b = 3$ for the segment emptiness problem. We thus obtain the following result.

FIG. 1. Query line ℓ and half plane γ .

THEOREM 3.8. *Given a set Γ of n half planes in $\mathbb{R}^3$ and a parameter m , $n \leq m \leq n^3$, we can preprocess Γ , in time $O(m^{1+\epsilon})$, into a data structure of size $O(m^{1+\epsilon})$, so that a ray shooting query can be answered in time $O(\frac{n}{m^{1/3}} \log^{O(1)} n)$. The data structure can be maintained dynamically, as half planes are being inserted in or deleted from Γ , in amortized time $O(m^{1+\epsilon}/n)$ per update operation.*

4. Nearest and farthest neighbors searching. We now divert our attention to the neighbor searching problems: *Given a set S of n points in $\mathbb{R}^d$, store S in a data structure so that, for a query point ξ , we can quickly compute a point of S closest to (or farthest from) ξ .*

Clarkson showed that if we allow $O(n^{\lceil d/2 \rceil + \epsilon})$ space, we can compute a nearest or farthest neighbor of a query point in S in time $O(\log n)$, but no efficient algorithm is known, if we allow, say, $O(n \log n)$ space. In this section we apply the ray shooting algorithms to search for the closest and farthest neighbors.

We map each point $p = (p_1, p_2, \dots, p_d)$ of S to the hyperplane $\mathcal{E}(p)$ in $\mathbb{R}^{d+1}$, which is the graph of a d -variate linear function

$$h_p(x_1, x_2, \dots, x_d) = 2p_1x_1 + 2p_2x_2 + \dots + 2p_dx_d - (p_1^2 + p_2^2 + \dots + p_d^2).$$

It is well known that $p \in S$ is a closest neighbor of a point $\xi = (\xi_1, \xi_2, \dots, \xi_d)$ if and only if

$$h_p(\xi_1, \xi_2, \dots, \xi_d) = \max_{q \in S} h_q(\xi_1, \xi_2, \dots, \xi_d)$$

(see [18], [19] for details). The problem of computing a closest neighbor thus reduces to finding a hyperplane in the upper envelope of $\mathcal{E}(S)$ hit by the vertical ray ρ , emanating from the point $(\xi_1, \xi_2, \dots, \xi_d, +\infty)$ in $-x_{d+1}$ direction. Since the upper envelope of $\mathcal{E}(S)$ is a convex polytope defined as the intersection of n half spaces and the origin point of ρ lies in $\mathcal{E}(S)$, we can use Lemma 3.4. Similarly, a farthest neighbor of ξ in S is a point p if

$$h_p(\xi_1, \xi_2, \dots, \xi_d) = \min_{q \in S} h_q(\xi_1, \xi_2, \dots, \xi_d),$$

and the problem of computing a farthest neighbor thus reduces to shooting a vertical ray in the lower envelope of $\mathcal{E}(S)$ from $(\xi_1, \xi_2, \dots, \xi_d, -\infty)$ (in $+x_{d+1}$ direction). Hence, by Lemma 3.4, we obtain the following result.

THEOREM 4.1. *Given a set S of n points in $\mathbb{R}^d$ and a parameter $n \leq m \leq n^{\lceil d/2 \rceil}$, we can preprocess S in time $O(m^{1+\varepsilon})$ into a data structure of size $O(m^{1+\varepsilon})$, so that, for a query point ξ , we can compute its closest or farthest neighbor in S in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n)$. Moreover, the data structure can be maintained dynamically, as points are being inserted in or deleted from S , in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

We can extend this algorithm to report k nearest or farthest neighbors of a query point. We will restrict ourselves to nearest neighbors; the farthest neighbors can be handled analogously. The k nearest neighbors of ξ are the same as the first k hyperplanes of $\mathcal{E}(S)$ intersected by the vertical ray ρ emanating from $(\xi_1, \xi_2, \dots, \xi_d, +\infty)$. Therefore, by Theorem 3.3, we can find k nearest neighbors of ξ in time $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$, but we can do better when k is not very large.

By our reductions, it suffices to have an algorithm deciding whether a query point lies below at most k hyperplanes, and also to have a suitable reporting algorithm. A result of [23] in a dual setting shows that, in $\mathbb{R}^d$, all k hyperplanes lying above a query point can be reported in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log n + k)$ using $O(m^{1+\varepsilon})$ space and preprocessing.

Such a reporting algorithm can be turned into an algorithm that checks whether there are at most k hyperplanes above a query point: Given an integer k , let t_k be (an upper bound for) the running time of the reporting algorithm provided that it reports at most k points. One runs the reporting algorithm, terminating it if it does not stop within t_k time units. If it has not stopped by this time, we know that the number of hyperplanes lying above the query hyperplane is larger than k ; otherwise we can check the number of reported hyperplanes directly. Hence, we can check in time $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$ whether a query point lies below at least k hyperplanes. A parallel implementation of the reporting algorithm with $O(\log n)$ parallel steps and $O(\frac{n}{m^{1/(d+1)}} \log^4 n + k)$ processors is again straightforward, and in the checking version, we terminate it when it tries to use more than t_k processors. Plugging these algorithms into the parametric search technique, we obtain a data structure that reports the first k hyperplanes hit by a ray originating in the polytope in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k \log^2 n)$. As usual, the space and preprocessing time of the algorithm are $O(m^{1+\varepsilon})$.

THEOREM 4.2. *Given a set S of n points in $\mathbb{R}^d$, we can preprocess it, in time $O(m^{1+\varepsilon})$, into a data structure of size $O(m^{1+\varepsilon})$ so that, for a query point ξ , we can compute its k closest (or farthest) neighbors in S in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k \log^2 n)$. The data structure can be maintained dynamically, as points are being inserted in or deleted from S , in amortized time $O(m^{1+\varepsilon}/n)$ per update operation.*

5. Hidden surface removal. Consider the following problem: “Given a set T of n triangles in $\mathbb{R}^3$ and a viewpoint p at $z = +\infty$, we want to compute the *visibility map* $\mathcal{M}(T)$ of T , i.e., the subdivision of the viewing plane $z = +\infty$ such that the same triangle of T is visible in $-z$ direction from all points of a face.” Note that if we are given an arbitrary viewpoint p , we can apply an appropriate transformation so that p maps to $z = +\infty$.

In the last few years much work has been done on various special cases of the hidden surface removal problem, see [27], [4], [8], [7]. Most of these algorithms apply efficient ray shooting procedures to compute the visibility map. Recently, de Berg et al. [7] gave an $O(n^{1+\epsilon}\sqrt{k} + k)$ algorithm for the general case, where k is the size of the visibility map. Their algorithm uses a ray shooting procedure for a collection of curtains as a subroutine; a *curtain* is an unbounded vertical triangle with two of its edges being vertical rays extending to $-\infty$. The overall running time of their algorithm is governed by the time complexity (query time plus preprocessing) of the ray shooting procedure. In this section we will present a faster solution for the ray shooting among a family of curtains, which in turn will improve the time complexity of their hidden surface removal algorithm.

Let Γ be a collection of curtains in $\mathbb{R}^3$. As usual, we describe a data structure for the segment emptiness problem for Γ . Let ℓ denote the line containing e , ℓ_γ the line containing the bounded segment of a curtain γ , and $\bar{\delta}$ the xy -projection of an object δ . The segment e intersects a curtain γ if and only if (i) $\bar{e}$ intersects $\bar{\gamma}$, and (ii) ℓ lies below ℓ_γ . Let $\bar{\Gamma} = \{\bar{\gamma} \mid \gamma \in \Gamma\}$ denote a set of n segments in $\mathbb{R}^2$. It is known (see, e.g., [4]) that $\bar{e}$ intersects $\bar{\gamma}$ if and only if ℓ separates the endpoints of $\bar{\gamma}$ and $\bar{e}$ intersects ℓ_γ . We construct a four-level data structure (similar to the one used in [4]; see Appendix for details) on $\bar{\Gamma}$.

We first construct a partition tree on the left endpoints of the segments of $\bar{\Gamma}$ and then, for each node z of the tree, we construct a secondary partition tree on the right endpoints of the segments of $\bar{\Gamma}$ whose left endpoints are in the canonical subset associated with z . Next, for each second-level node we extend the segments of $\bar{\Gamma}$, whose right endpoints are stored at that node, to full lines in the plane and dualize them to points in $\mathbb{R}^2$. We construct a third-level partition tree on these points. For a third-level node v , let $\bar{\Gamma}_v$ (respectively, Γ_v) be the set of segments (respectively, curtains) corresponding to the points associated with v . Finally, at each third level node v , we store a data structure that, for a query line ℓ (the one extending a query segment e), decides whether it lies above all lines of $\{\ell_\gamma; \gamma \in \Gamma_v\}$ (see §3.3).

To detect an intersection between e and the curtains of Γ , we search the first two levels of the above structure with $\bar{\ell}$, the third level structure with the double-wedge dual to $\bar{e}$, and finally the fourth level structure with the Plücker hyperplane of ℓ (see the Appendix for details). If the tests come out positive for the fourth-level structures of every third-level node, filtered out by the previous levels of the partition tree, our query segment is empty. Putting all this together, we get a standard trade-off for the segment emptiness problem among curtains with $b = 2$.

Plugging the resulting ray shooting procedure into the hidden surface removal algorithm of [7] and choosing $m = \lceil n^{2/3}k^{2/3} \rceil$, we can compute the visibility map in time $O(n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon})$. Note that we do not know the value of k in advance, therefore we must guess an initial value of m and update it as the algorithm proceeds, as in [7], [27]; see any of these papers for details. Hence, we can conclude the following theorem.

THEOREM 5.1. *The visibility map of a given set of n triangles in $\mathbb{R}^3$ with respect to a viewing point can be computed in time $O(n^{2/3+\epsilon}k^{2/3} + n^{1+\epsilon})$, where k is the output size.*

6. Conclusion. In this paper we have applied the parametric search technique to obtain efficient procedures for the ray shooting and related problems. Recently, further progress was made in the construction of (static) range searching algorithms (see [24] for simplex range searching algorithms and multilevel range searching structures and [28] for half-space range reporting with polylogarithmic query time). As a result, the n^ϵ factors appearing in the

complexity bounds for our static ray shooting algorithms can be replaced by polylogarithmic factors. Also, in the case of ray shooting in a convex polytope, we can remove some of the extra logarithmic factors in the query time arising by a straightforward application of the parametric search method; see [25], [28].

Finally, we conclude by mentioning some open problems.

(i) Can the ray shooting algorithm for half planes be extended to triangles in $\mathbb{R}^3$? The currently best known algorithm admits the standard trade-off with $b = 4$.

(ii) What about ray shooting in a collection of spheres or other nonlinear objects in $\mathbb{R}^3$? The authors are not aware of any efficient procedure for ray shooting or even intersection detection algorithms for nonlinear objects.

(iii) Can the running time of k -nearest neighbor searching be improved to

$$O\left(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k\right)?$$

Appendix. Multilevel partition trees. In some of our ray shooting problems (e.g., §§3.3 and 5), we needed “tailored” range searching structures for the corresponding segment emptiness problems. These particular data structures are not explicitly described in the literature, but they can be easily designed using *multilevel (recursive) partition trees*. Multilevel partition trees were originally introduced by Dobkin and Edelsbrunner [16] and they have been successfully applied to several geometric problems [12], [22], [4].

Let us start by describing a partition tree for the half-space range searching. We will consider only some abstract properties without going into specific implementation details. Thus our description encompasses the partition scheme of [12], as well as a somewhat simpler and better partition scheme of [22].

Let S be a set of n points in $\mathbb{R}^d$. A partition tree on S consists of a collection of certain distinguished subsets of S , called the *canonical subsets*. The task of a partition tree is to express the intersection of S with a given half space γ as a disjoint union of canonical subsets; such a decomposition of $S \cap \gamma$ is called a *canonical decomposition*. A partition tree is implemented as a tree-like data structure, each of whose node is associated with canonical subsets. At each node, we also store certain auxiliary information that allows us to find a canonical decomposition of $S \cap \gamma$ efficiently.

In order to answer the queries efficiently, we need a worst-case upper bound on the maximum number of canonical subsets used in a canonical decomposition of $S \cap \gamma$. On the other hand, the sum of sizes of all canonical subsets essentially determines the space requirements of the partition tree. The results of [12], [22] show that, for any $n \leq m \leq n^d$, we can build a partition tree of size (i.e., the sum of the size of its canonical subsets is) $O(m^{1+\epsilon})$, which can decompose $S \cap \gamma$ into $O(\frac{n}{m^{1/d}} \log n)$ canonical subsets, for any query half space γ .

Depending on the purpose of a partition tree, we also store some kind of additional information along with each canonical subset. For example, we need to store the cardinality of each canonical subset if we want to use it for answering queries of the form “how many points of S are contained in a query half space.”

To introduce the concept of a multilevel partition tree, let us consider a slightly more complicated example, where we want to count the number of points of S in a query wedge, i.e., in an intersection of two half-spaces⁵. We thus take every canonical subset C in our partition tree, and build a “secondary” partition tree for C ; this data structure corresponds to the auxiliary information associated with C in the “primary” partition tree. The auxiliary information stored at each node of the secondary partition tree is still the cardinality of the

⁵The partition trees of [22] allow us to handle such a query directly, since the points of S in every *simplex* can be decomposed into the appropriate number of canonical subsets, but we want to illustrate the principle here.

corresponding secondary canonical subset. Given a wedge $w = \gamma \cap \gamma'$, where γ, γ' are two half-spaces, we first obtain a canonical decomposition of $S \cap \gamma$ with respect to the primary tree. Then, for every canonical subset C in this decomposition, we use the corresponding secondary data structure to count the number of points in $C \cap \gamma'$. Finally we sum up all the answers to compute the value of $|S \cap w|$. Hence, the first level of our data structure sifts out the points contained in γ , and the second level sifts the points contained in γ' . Thus, the two levels together filter out the points of S that lie in $w = \gamma \cap \gamma'$.

We need not limit ourselves to just two levels; for instance, we need $d + 1$ levels to handle simplex range counting queries in this spirit. The examples described in this paper illustrate that the secondary structure need not deal with the points of canonical subsets directly, but rather it can be set up for some objects in one-to-one correspondence with these points (e.g., in §5, the primary partition tree is constructed on the left endpoints of some set of segments, the secondary partition trees are constructed on right endpoints of the appropriate segments, and the tertiary partition trees are constructed on the points dual to some set of lines).

In order to analyze the performance of such multilevel structures, we apparently need more information about the distribution of sizes of the canonical subsets. From the analysis in [12], [22], we can obtain the following convenient form.

THEOREM A.1. *For an n -point set in $\mathbb{R}^d$ and a parameter $m, n \leq m \leq n^d$, we can construct, in time $O(m^{1+\varepsilon})$, a partition tree that contains at most $O(\max\{\frac{n}{s}, \frac{m^{1+\varepsilon}}{s^d}\})$ canonical subsets of size exceeding s for any $s \leq n$ and, for any half-space γ , the canonical decomposition of $\gamma \cap S$ contains at most*

$$O\left(\min\left\{\frac{n^{1-1/d+\varepsilon}}{s^{1-1/d}}, \frac{n}{m^{1/d}} \log n\right\}\right)$$

canonical subsets of size exceeding s .

Using these estimates, the analysis of multilevel partition trees becomes straightforward, especially if we ignore $O(n^\varepsilon)$ factors. Let us consider the above outlined two-level example for wedge range counting. For the sake of simplicity, let us assume that $m = n$ for the primary partition tree, and that, for every canonical subset C of size c , we build a secondary partition tree with $m = c$. The secondary structure constructed on C requires $O(c^{1+\varepsilon})$ space. The total space required by this two-level structure is thus at most

$$\sum_{i=0}^{\log_2 n} 2^{i(1+\varepsilon)} O\left(\max\left\{\frac{n}{2^i}, \frac{n^{1+\varepsilon}}{2^{di}}\right\}\right) = O(n^{1+\varepsilon}),$$

and the number of second-level canonical subsets used for decomposition of a wedge is at most

$$\sum_{i=0}^{\log_2 n} 2^{i(1-1/d)} \log n \cdot O\left(\min\left\{\frac{n^{1-1/d+\varepsilon}}{2^{i(1-1/d)}}, n^{1-1/d} \log n\right\}\right) = O(n^{1-1/d} \log^2 n).$$

As for other values of m , a similar calculation shows that the total space is $O(m^{1+\varepsilon})$ and a query output consists of $O(\frac{n}{m^{1/d}} \log^2 n)$ canonical subsets. In general, the space requirements and the query complexity of a multilevel partition tree are essentially determined by the least efficient level. The basic analysis of this kind is given in [4], [12], and finer complexity questions (the tuning of subpolynomial factors) are touched in [22].

Acknowledgments. The authors thank Marco Pellegrini for pointing out an error in an earlier version of the paper, Micha Sharir for several helpful discussions, and the referees for useful comments.

REFERENCES

- [1] P.K. AGARWAL, *Ray shooting and other applications of spanning trees with low stabbing number*, SIAM J. Comput., 21 (1992), pp. 540–570.
- [2] P. K. AGARWAL AND J. MATOUŠEK, *Dynamic half-space range reporting and its applications*, Tech. Report CS-1991-43, Dept. of Computer Science, Duke University, Durham, NC, 1991; Algorithmica, to appear.
- [3] P. K. AGARWAL AND M. SHARIR, *Planar geometric location problems*, Tech. Report 90-58, DIMACS, Rutgers University, August, 1990. (Also to appear in Algorithmica.)
- [4] ———, *Applications of a new partitioning scheme*, Discrete Comput. Geom., 9 (1993), pp. 11–38.
- [5] B. ARONOV, M. PELLEGRINI, AND M. SHARIR, *On the zone of a surface in a hyperplane arrangement*, Discrete Comput. Geom., to appear.
- [6] R. BAR YEHUDA AND S. FOGEL, *Good splitters with applications to ray shooting*, in Proc. 2nd Canadian Conf. on Computational Geometry, 1990, pp. 81–85.
- [7] M. DE BERG, D. HALPERIN, M. OVERMARS, J. SNOEYINK, AND M. VAN KREVELD, *Efficient ray shooting and hidden surface removal*, in Proc. 7th Symposium on Computational Geometry, 1991, pp. 51–60.
- [8] M. DE BERG AND M. OVERMARS, *Hidden surface removal for axis-parallel polyhedra*, Comput. Geom.: Theory and Appl., 1 (1992), pp. 247–268.
- [9] B. CHAZELLE, H. EDELSBRUNNER, M. GRIGNI, L. GUIBAS, J. HERSHBERGER, M. SHARIR, AND J. SNOEYINK, *Ray shooting in polygons using geodesic triangulations*, Proc. 17th International Colloquium on Automata, Languages and Programming, 1991, pp. 661–673.
- [10] B. CHAZELLE, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND J. STOLFI, *Lines in space: Combinatorics and algorithms*, Proc. 21st ACM Symposium on Theory of Computing, 1989, pp. 382–393. Full version: Tech. Report 491, Dept. of Computer Science, New York University, New York, February, 1990.
- [11] B. CHAZELLE AND L. GUIBAS, *Visibility and intersection problems in plane geometry*, Discrete Comput. Geom., 4 (1989), pp. 551–589.
- [12] B. CHAZELLE, M. SHARIR, AND E. WELZL, *Quasi-optimal upper bounds for simplex range searching and new zone theorems*, Algorithmica, 8 (1992), pp. 407–430.
- [13] S.W. CHENG AND R. JANARDAN, *Space efficient ray shooting and intersection searching: Algorithms, dynamization, and applications*, J. Algorithms, 3 (1992), pp. 670–692.
- [14] K. CLARKSON, *A randomized algorithm for closest point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.
- [15] R. COLE, *Slowing down sorting networks to obtain faster sorting algorithms*, J. Assoc. Comput. Mach., 31 (1984), pp. 200–208.
- [16] D. DOBKIN AND H. EDELSBRUNNER, *Space searching for intersecting objects*, J. Algorithms, 8 (1987), pp. 348–361.
- [17] D. DOBKIN AND D. KIRKPATRICK, *Determining the separation of preprocessed polyhedra: a unified approach*, Proc. 17th International Colloquium on Automata, Languages and Programming, 1990, pp. 400–413.
- [18] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, New York, 1987.
- [19] H. EDELSBRUNNER AND R. SEIDEL, *Voronoi diagrams and arrangements*, Discrete Comput. Geom., 1 (1985), pp. 25–44.
- [20] L. GUIBAS, M. OVERMARS, AND M. SHARIR, *Ray shooting, implicit point location, and related queries in arrangements of segments*, Tech. Report 433, Courant Institute, New York University, New York, 1989.
- [21] J. MATOUŠEK, *Approximations and optimal geometric divide-and-conquer*, Proc. 23rd ACM Symposium on Theory of Computing, 1991, pp. 506–511.
- [22] ———, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.
- [23] ———, *Reporting points in halfspaces*, Comput. Geom.: Theory and Appl., 2 (1992), pp. 169–186.
- [24] ———, *Range searching with efficient hierarchical cuttings*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 276–285.
- [25] J. MATOUŠEK AND O. SCHWARZKOPF, *Linear optimization queries*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 16–25.
- [26] N. MEGIDDO, *Applying parallel computation algorithms in the design of serial algorithms*, J. Assoc. Comput. Mach., 30 (1983), pp. 852–865.
- [27] M. OVERMARS AND M. SHARIR, *Output-sensitive hidden surface removal*, Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, pp. 598–603.
- [28] O. SCHWARZKOPF, *Ray shooting in convex polytopes*, Proc. 8th ACM Symposium on Computational Geometry, 1992, pp. 286–295.
- [29] D. SOMMERVILLE, *Analytical Geometry in Three Dimensions*, Cambridge University Press, London, 1951.