Ray shooting and Parametric Search *

Pankaj K. Agarwal† and Jiří Matoušek‡

Abstract

We present efficient algorithms for the *ray shooting* problem: Given a collection Γ of objects in $\mathbb{R}^d$, build a data structure, so that one can quickly determine the first object of Γ hit by a query ray. Using the parametric search technique, we reduce this problem to the *segment emptiness* problem. For various ray shooting problems, we achieve space/query time tradeoffs of the following type: for some integer b and a parameter m ($n \leq m \leq n^b$) the queries are answered in time $O(\frac{n}{m^{1/b}} \log^{O(1)} n)$, with $O(m^{1+\varepsilon})$ space and preprocessing time ($\varepsilon > 0$ is arbitrarily small but fixed). We get $b = \lfloor d/2 \rfloor$ 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 $\mathbb{R}^3$. Next we apply the ray shooting algorithms to several problems including reporting k-nearest (or k-farthest) neighbors, hidden surface removal, computing convex layers, and computing levels in arrangements of planes. All the algorithms described here either give the first non-trivial solutions to these problems, or improve the previously best known solutions significantly.

*Work by the first author has been supported by National Science Foundation Grant CCR-91-06514.

†Computer Science Department, Duke University, Durham, NC 27706.

‡Department of Applied Mathematics, Charles University, Praha.

24th ANNUAL ACM STOC - 5/92/VICTORIA, B.C., CANADA
© 1992 ACM 0-89791-512-7/92/0004/0517...$1.50

1 Introduction

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

The ray shooting problem has received a lot of attention in the last few years because of its applications in graphics and other geometric problems [11, 20, 1, 5, 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. 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 preprocessing[1] [1, 5]. Although no lower bound is known for this case, it is conjectured that this bound is close to optimal. But the problem is far from being solved in three and higher dimensions. For example, no efficient ray shooting algorithm has been known for a convex d-polytope for $d > 3$. Even in $\mathbb{R}^3$, non-trivial solutions have been obtained only very recently (cf. [5, 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

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

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 [25], 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.

The range searching algorithms, we have at disposal, usually admit time/space tradeoffs: the more space (and preprocessing time) we use, the faster the queries can be answered. Such a tradeoff then transfers to the ray shooting results. A usual form of this tradeoff 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 $O(\frac{n}{m^{1/b}} \log^c n)$. Note that since we require $m \geq n$, the smallest amount of space we consider in this tradeoff 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 tradeoff* with certain value of b. It is understood that c is always a (reasonably small) constant.

We first describe the general ray shooting algorithm (Section 2), and then apply this technique to obtain fast procedures for the following specific instances (Section 3): **Hyperplanes in $\mathbb{R}^d$.** Previously, efficient solutions were known only for $d \leq 3$ [5]. Moreover, they did not support insertion/deletion of planes for $d = 3$. We obtain a data structure with query time $O(\frac{n}{m^{1/d}} \log^4 n)$.

Convex d-polytope. We assume that the polytopes is defined as the intersection of n half-spaces. For $d = 2$ a straight-forward binary search can answer a ray shooting query in $O(\log n)$ time, and for $d = 3$ one can use Dobkin-Kirkpatrick hierarchical representation of a 3-polytope to obtain an optimal algorithm [17]. But for $d > 3$, 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 present a data structure with query time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^5 n)$.

Half-planes in $\mathbb{R}^3$. The previously best known result answered a query in time $O(\frac{n^{16/15+\varepsilon}}{m^{4/15}})$ using $O(m^{1+\varepsilon})$ space and preprocessing [5]. We present a data structure with query time $O(\frac{n}{m^{1/3}} \log^3 n)$.

Next, we apply the above ray shooting algorithms for various other problems:

Proximity problems. Clarkson showed that if we allow $O(n^{\lceil d/2 \rceil + \varepsilon})$ space, a nearest or farthest neighbor of a query point in S can be computed in time $O(\log n)$. But it neither admits space/query time tradeoff nor allows insertion/deletions of points. Recently Mulmuley showed that a set of n points can be stored into a data structure of size $O(n^{\lceil d/2 \rceil + \varepsilon})$, so that k-nearest neighbors can be reported in expected time $O(k \log n)$, and a random sequence of insertions/deletions can be performed in $O(n^{\lceil d/2 \rceil - 1 + \varepsilon})$ amortized time with high probability. But his algorithm does not perform well on an arbitrary sequence of insertions/deletions. Moreover, it also does not admit space query time tradeoff.

By a well-known reduction, we reduce the problem to a suitable ray shooting algorithm which 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 tradeoff with $b = \lceil d/2 \rceil$ [18]. It can also report the k-nearest (or k-farthest) neighbors, for any given $k \leq n$, at an additional cost of $k \log^2 n$ in the query time. As usual, the data structure can be updated in $O(m/n^{1-\varepsilon})$ amortized time per insert/delete operation. (Section 4.)

Hidden surface removal. 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 in the last few years [28, 5, 8, 7], only very recently de Berg et al. [7] presented an output-sensitive algorithm for the general case, whose time complexity is $O(n^{1+\varepsilon}\sqrt{k} + k)$, where k is the output size. We improve the running time of their algorithm to $O(n^{2/3+\varepsilon}k^{2/3} + n^{1+\varepsilon})$. (Section 5.)

Convex layers in 3D. Given a set S of n points in $\mathbb{R}^3$, the convex layers of S are defined iteratively as follows: Compute the convex hull of S, delete the points of S lying on the convex hull, and repeat the same step until S becomes empty. Although an optimal $O(n \log n)$ time algorithm has been known for a long time in the plane [9], the best known algorithm in 3D runs in time $O(n^{3/2} \log n)$. We present an algorithm for computing convex layers of n points in $\mathbb{R}^3$, whose time complexity is $O(n^{1+\varepsilon})$. (Section 6.)

Levels in arrangements of planes in 3D. Let H be a set of n planes in $\mathbb{R}^3$. For an integer $k \leq n$, the *k-level* in arrangement of H is the closure of faces f of arrangement of H, $\mathcal{A}(H)$, such that k planes of H lie strictly above f. Edelsbrunner and Welzl [19] presented an algorithm for computing the k-level in arrangements of n lines in the plane, whose time complexity was $O(n \log n + b \log^2 n)$, where b is the size of the level.

We present an output-sensitive algorithm for computing a level in $\mathcal{A}(H)$, whose time complexity is $O((b+n)n^\varepsilon)$, where b is the size of the k-level. This is the first known output-sensitive algorithm for computing a level in 3D. (Section 7.)

Our algorithm for computing a level also yields an $O(n^{1+\varepsilon}k)$ time deterministic algorithm for computing the k^{th}-order Voronoi diagram of a set of n points in the plane.

New developments. Recently, a further progress was made in the construction of (static) range searching algorithms (see [23] for simplex range searching algorithms and multilevel range searching structures and [30] 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, one can remove some of the extra logarithmic factors in the query time arising by a straightforward application of the parametric search method, see [24, 30].

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

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

Suppose that we have an efficient algorithm that, given a segment $ox \in \mathrm{Seg}(\mathcal{R})$, decides whether it intersects some objects $g \in \Gamma$. 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 $x < \varphi(\Gamma, \rho)$, and otherwise $x > \varphi(\Gamma, \rho)$. As it is often the case in similar situations, the *parametric search technique* due to Megiddo [25] 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 computation 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 of 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 $t^* \notin I$, we can conclude that t^* does not exist, and we stop. 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 $I \cap (\beta_i, \beta_{i+1})$ and 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^*, A has to make some comparison whose polynomial vanishes at t^* (see [4] for a proof), which will cause the computation to 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, the sequential algorithm A is replaced 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). This 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. An improvement of this technique by Cole [16] 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 $\mathrm{Seg}(\mathcal{R})$. Let A_p be a parallel algorithm for answering a segment emptiness query, which uses p processors and runs in T_A parallel steps, 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 query segment intersects less than, or equal to, or more than k objects of Γ. 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.

3.1 Ray shooting among hyperplanes

In this subsection 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 [5] have given an algorithm that gives a standard tradeoff for ray shooting among hyperplanes with $b = d$. We obtain similar bounds in 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 (resp. segment) in $\mathbb{R}^d$ is a point (resp. double-wedge),[2] 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 one wants to report or count the points contained in a query simplex.

Chazelle et al. [13] have shown that using $O(m^{1+\varepsilon})$, $(n \leq m \leq n^d)$ space and preprocessing, one can answer a double wedge range query in time $O(\frac{n}{m^{1/d}} \log^2 n)$. With a knowledge of this data structure, it is straight-forward 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

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

[5, 21] that the data structure can be dynamically maintained under insertions and deletions of points. Hence, in view of Theorem 2.1, we have

Theorem 3.1 *Given a set H of n hyperplanes in $\mathbb{R}^d$, a parameter m, $n \leq m \leq n^d$, one can build, in time $O(m^{1+\varepsilon})$, a data structure of size $O(m^{1+\varepsilon})$ that supports ray shooting queries in time $O(\frac{n}{m^{1/d}} \log^4 n)$. This data structure can be maintained dynamically in $O(m^{1+\varepsilon}/n)$ amortized time per insertion/deletion of a hyperplane.*

Remark 3.2: The above theorem can be extended to report in time $O(\frac{n}{m^{1/d}} \log^4 n + k)$ the first k hyperplanes intersected by the query ray.

3.2 Ray shooting in a convex polytope

Next, 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.

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 (this is just for the sake of explanation, we can actually avoid performing any transformation by modifying the algorithm suitably). 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 [22] it is shown that the half-space emptiness queries can be answered in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ using $O(m^{1+\varepsilon})$ space and preprocessing (the algorithm includes the data structure due to Clarkson [14] for the "large space" case). A parallel implementation with $O(\log n)$ time and number of processors bounded by the sequential running time is quite straight-forward. 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, 22]). We thus have

Lemma 3.3 *Given a convex polytope in $\mathbb{R}^d$, described as the intersection of n half-spaces, and a parameter m, $n \leq m \leq n^{\lfloor d/2 \rfloor}$, one 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 $\mathcal{P}$, one can determine the first point of the polytope boundary hit by the ray in $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$ time.*

This result has one surprising consequence. Namely, the half-space emptiness algorithm of [22] 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 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 $(h_1^*, \ldots, h_{d-1}^*, +\infty)$ (where $(h_1^*, \ldots, h_d^*)$ 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

Corollary 3.4 *Given an n point set S in $\mathbb{R}^d$, one 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, one can determine, in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^3 n)$, 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 3.4, 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 3.4 and Theorem 2.1, a ray shooting query can be answered in time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^5 n)$.

In [3], we presented a dynamic data structure of size $O(m^{1+\epsilon})$ for the half-space range searching, which could insert or delete a point in $O(m/n^{1-\epsilon})$ amortized time. The query time of this structure is the same as that of the static structures. Hence, we can conclude

Theorem 3.5 *Given a convex polytope in $\mathbb{R}^d$ described as the intersection of n half-spaces, and a parameter m ($n \leq m \leq n^{\lfloor d/2 \rfloor}$), one can preprocess it in time $O(m^{1+\epsilon})$ into a data structure of size $O(m^{1+\epsilon})$, so that the first point of the polytope boundary hit by a query 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 in amortized time $O(m^{1+\epsilon}/n)$ per insert/delete operation.*

3.3 Ray shooting among half-planes

In this subsection we consider the case when Γ is a set of half-planes in $\mathbb{R}^3$. Although it is already known that a ray shooting query among planes in $\mathbb{R}^3$ can be answered in roughly $n/m^{1/3}$ time, this procedure does not extend to half-planes. The query time of the best known algorithm is close to $n^{16/15}/m^{4/15}$, which is the same as the query time for ray shooting among triangles in $\mathbb{R}^3$ [5]. Here we improve the query time to $O(\frac{n}{m^{1/3}} \log^4 n)$; as usual, we will describe an efficient procedure for the segment emptiness problem. We use a "multi-level" partition trees tailored to this specific application (cf. [5, 13]).

Let H denote the set of planes supporting the half-planes of Γ. We construct a partition tree $\mathcal{T}$ on the point set H^*. Each node v of $\mathcal{T}$ is associated with a subset of points in H^*. Let Γ_v be the set of half-planes corresponding to the points 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.

At each node v we construct a secondary 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 5 dimensions[3]. Combining their reduction with the already mentioned results of [22] for the half-space emptiness problem, we get a data structure for the segment emptiness problem (with respect to half-planes in 3-D) that admits a standard tradeoff with $b = 2$; see [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, such that within each canonical subset, either p lies below all the planes containing the half-planes of Γ_v and q lies above all of them, or vice-versa. Let Γ_v be a first-level canonical subset of the query output, and 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 ℓ is in clockwise (resp. counter-clockwise) direction to the line $\partial\gamma$, then ℓ intersects γ if and only if ℓ lies below (resp. above) $\partial\gamma$, i.e., ℓ has negative orientation with respect to $\partial\gamma$. 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 tradeoff with $b = 3$ for the segment emptiness problem. We thus obtain

Theorem 3.6 *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 struc-*

[3] The reduction uses so-called Plücker coordinates of lines. Every line ℓ' of the set L_v is mapped to a Plücker point $\pi(\ell')$ in projective 5-space, and a query line ℓ is mapped to a Plücker hyperplane $\varpi(\ell)$ in projective 5-space. The query line ℓ has positive (resp. negative) orientation with respect to all lines in L_v if and only if the hyperplane $\varpi(\ell)$ lies above (resp. below) the Plücker points of all lines in L_v, see [31, 10]. The problem thus reduces to determining whether the half-space lying above (resp. below) $\varpi(\ell)$ is empty, which can be done using the data structure of Clarkson [14].

ture can be maintained dynamically in amortized time $O(m^{1+\epsilon}/n)$ per insert/delete operation.

4 Nearest and Farthest Neighbors Searching

We now turn our attention to the neighbor searching problems: *Given a set S of n points in $\mathbb{R}^d$, store S into a data structure so that, for a query point ξ and an integer $k \leq n$, one can quickly compute k-nearest (or k-farthest) neighbors of ξ in S.*

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

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

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

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

(see [18]). 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 $(\xi_1, \xi_2, \ldots, \xi_d, +\infty)$ 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 3.3. A farthest neighbor query can also be answered using the same approach. Hence, we obtain

Theorem 4.1 *Given a set S of n points in $\mathbb{R}^d$ and a parameter $n \leq m \leq n^{\lceil d/2 \rceil}$, one can preprocess S, in time $O(m^{1+\epsilon})$, into a data structure of size $O(m^{1+\epsilon})$ so that, for a query point ξ, one 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 in amortized time $O(m^{1+\epsilon}/n)$ per insert/delete 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 $(\xi_1, \xi_2, \ldots, \xi_d, +\infty)$. Therefore, by Remark 3.2, one 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 a suitable reporting algorithm. A result of [22] in a dual setting shows that, in $\mathbb{R}^d$, all s hyperplanes lying above a query point can be reported in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log n + s)$ using $O(m^{1+\epsilon})$ space and preprocessing. Such a reporting algorithm can be turned into an algorithm that checks in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log n + k)$ whether there are at most k hyperplanes above a query point; see [2]. A parallel implementation of the reporting algorithm with $O(\log n)$ parallel steps is again straightforward. Hence we obtain

Theorem 4.2 *Given a set S of n points in $\mathbb{R}^d$, one can preprocess it, in time $O(m^{1+\epsilon})$, into a data structure of size $O(m^{1+\epsilon})$ so that, for a query point ξ, one can compute its k closest (or farthest) neighbors in S in time $O(\frac{n}{m^{1/\lceil d/2 \rceil}} \log^3 n + k \log^2 n)$.*

Remark 4.3: If $m = n^{\lceil d/2 \rceil}$, the query time of the above theorem can be improved to $O(k \log n)$, which matches the bound of Mulmuley's algorithm [27].

5 Hidden Surface Removal

Consider the following problem: "Given a set T of n triangles in $\mathbb{R}^3$ and a view point 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 from all points of a face." Note that if we are given an arbitrary view point p, we can apply an appropriate transformation so that p maps to $z = +\infty$.

De Berg et al. [7] have shown that the hidden surface removal problem can be reduced to shooting $O(k)$ rays (k is the size of the visibility map) among a collection of n curtains; a *curtain* is an unbounded vertical triangle with two of its edges being vertical rays extending to $-\infty$. 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, it suffices to 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. [5]) that $\bar{e}$ intersects $\bar{\gamma}$ if and only if $\bar{\ell}$ separates the endpoints of $\bar{\gamma}$ and $\bar{e}$ intersects $\bar{\ell}_\gamma$. We construct a four-level partition tree (similar to the one used in [5]) on $\bar{\Gamma}$ that can answer a ray shooting query in time $O(\frac{n}{\sqrt{m}} \log^{O(1)} n)$.

Plugging the resulting ray shooting procedure into the hidden-surface removal algorithm of [7] and choosing $m = \lceil n^{2/3}/k \rceil$, 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 have to guess an initial value of m and update it as the algorithm proceeds, as in [7, 28]; see any of these papers for details. Hence, we can conclude

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 Computing Convex Layers

Let S be a set of n points in $\mathbb{R}^3$. Let $CH(S)$ denote the boundary of the convex hull of S. Then the convex layers $\mathcal{C}(S) = \{C_1, C_2, \ldots, C_m\}$ of S can be defined as: $S_0 = S$ and, for $1 \leq i \leq m$ $C_i = CH(S_{i-1}) \cap S_{i-1}$, $S_i = S_{i-1} - C_i$.

Let H denote the set of n planes dual to the points in S. For the sake of simplicity we assume that the planes of H are in the general position. In the dual setting, the problem of computing the convex layers of S reduces to computing the upper and lower envelopes of H (the cells in the arrangement of H that lie above and below all planes of H) repeatedly. That is, compute the planes of H appearing in the upper and lower envelopes of H, delete these planes from H, and repeat the above two steps until H becomes empty.

We will describe how to compute the upper envelope; the lower envelope can be computed analogously. For each plane $h \in H$, we pick up the half-space lying above h, and preprocess the intersection of the resulting half-spaces for efficient ray shooting using Theorem 3.5. We will compute the 1-skeleton of the upper envelope (i.e., the graph formed by the vertices and the edges of the upper envelope) by traversing its edges in the depth-first manner. Suppose we are at a vertex v of the upper envelope. We want to determine the other endpoints of the edges incident to v. We shoot a ray ρ from v along each edge e incident to v; ρ can be computed in $O(1)$ time. If ρ does not intersect any plane of H (other than the ones that contain ρ), e is unbounded. Otherwise, if ρ intersects a plane h, then $v' = \rho \cap h$ is the other endpoint of v. Since we want to find the first plane intersecting ρ that does not contain ρ, we delete the two planes containing ρ before answering the ray shooting query and insert them back after answering the query. By Theorem 3.5, we can answer a ray shooting query in $O(\log^5 n)$ time, and can insert/delete a half-space in $O(n^\epsilon)$ time, so we spend $O(n^\epsilon)$ time at v. If v' has not been visited earlier, we recursively search at v'. We repeat this procedure for all edges incident to v. Let n_u denote the number of planes of H appearing in its upper envelope. Since each vertex is visited only once, and the number of vertices is $O(n_u)$, the total time spent is $O(n_u n^\epsilon)$.

Similarly, we compute the lower envelope of H. After having computed the upper and lower envelopes, we delete the planes that appear in the lower and upper envelopes. We repeat the above procedure until H becomes empty. Summing the time complexity over all layers, we can conclude

Theorem 6.1 *The convex layers of a set of n points in $\mathbb{R}^3$ can be computed in $O(n^{1+\epsilon})$ time.*

7 Computing Levels in Plane Arrangements

Levels in hyperplane arrangements play an important role in several geometric problems, including higher order Voronoi diagrams [18] and half-space range searching [12].

It is well known that $\sum_{j=1}^{k} |\Pi_j| = O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ for an arrangement of n hyperplanes in $\mathbb{R}^d$ [15], and that this bound is tight in the worst case. Hence, the expected complexity of j-level with j randomly chosen in range from 1 to k is no worse than $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil - 1})$, but only little is known about the worst-case complexity of a single level (see [18] for older results and references for dimension 2, and [6, 29, 32] for recent results).

Since the complexity of a k-level varies a lot, a natural question is whether it can be computed in an output sensitive fashion. Edelsbrunner and Welzl [19] showed that a level in arrangement of n lines in the plane can be computed in time $O(n \log n + b \log^2 n)$, where b is the actual size of the level. Recently Mulmuley [26] presented a randomized algorithm for com-

puting all levels $\Pi_1, \ldots, \Pi_k$, for a given k, whose expected running time is $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ for $d \geq 4$ and $O(nk^2 \log \frac{n}{k})$ for $d = 3$. We are not aware of any efficient algorithm for computing a single level in arrangements of planes in $\mathbb{R}^3$. Here we present an output sensitive algorithm with $O(n^\epsilon)$ cost for every feature of the k-level. Our technique also extends to higher dimensions, though we can only obtain somewhat worse bounds:

Theorem 7.1 *Given a set H of n planes in $\mathbb{R}^3$ and an integer $k < n$, one can compute the k-level in the arrangement of H in time $O((n+b)^{1+\epsilon})$, where b is the actual size of the level.*

It is well known that computing the k^{th}-order Voronoi diagram in $\mathbb{R}^d$ can be reduced to computing the k-level in an arrangement of n hyperplanes in $\mathbb{R}^{d+1}$. Thus, we get

Corollary 7.2 *The k-th order Voronoi diagram of a set of n points in the plane can be computed deterministically in time $O((b+n)n^\epsilon) = O(n^{1+\epsilon}k)$, where b is the actual size of the diagram.*

Proof: We basically follow the same approach as in the previous section for computing the convex layers. That is, we traverse the 1-skeleton of Π_k by following its edges in a depth first manner. Assuming that the planes of H are in general position, every vertex of $\mathcal{A}(H)$ is the intersection point of three planes. At every vertex of the k-level, there are three edges of the level. Hence, having arrived at a vertex v through one of its edges, it suffices to determine the other endpoints of the remaining two edges and recursively search from these vertices on, provided that they have not been visited earlier.

For a vertex $v = h_1 \cap h_2 \cap h_3$ of the level, let $H_l(v)$ (resp. $H_u(v)$) denote the set of planes of H lying strictly below (resp. above) v (thus, by our general position assumptions, each plane of H except for h_1, h_2 and h_3 appears in $H_l(v)$ or $H_u(v)$). Throughout the depth-first search algorithm, we will maintain the following invariant:

> Whenever we visit a vertex v, we have at our disposal a data structure $\Lambda_l(v)$ for answering ray shooting queries inside the upper envelope of $H_l(v)$, and a similar data structure $\Lambda_u(v)$ for the lower envelope of $H_u(v)$.

Suppose that we have arrived at a vertex $v = h_1 \cap h_2 \cap h_3$ along one of its edges, $e_{1,2} \subset h_1 \cap h_2$. We query both $\Lambda_l(v)$ and $\Lambda_u(v)$ with a ray $\rho_{1,3}$ originating at v and going inside $h_1 \cap h_3$ in an appropriate direction. This gives us the first plane h_2' hit by the ray $\rho_{1,3}$ (if no such plane exists, $\rho_{1,3}$ determines an unbounded edge of the 1-skeleton). We then check whether the vertex $v' = h_1 \cap h_2' \cap h_3$ has already been visited. If the answer is 'no', we recursively search at v'. After returning to the vertex v, we perform a similar action with the ray $\rho_{2,3}$ originating in v and going within $h_2 \cap h_3$. After returning to v again, we go back to the vertex from which we originally came to v.

It remains to show how to maintain the invariant (the ray shooting data structures). This is quite simple: any two vertices v and v' joined by an edge share two of the triple of defining planes. Thus, when passing from a vertex $v = h_1 \cap h_2 \cap h_3$ to $v' = h_1' \cap h_2 \cap h_3$, we delete h_1' from either $\Lambda_l(v)$ (if h_1' is below v) or $\Lambda_u(v)$ (if h_1' is above v), and we insert h_1 to the appropriate one of $\Lambda_l(v)$, $\Lambda_u(v)$, obtaining $\Lambda_l(v')$ and $\Lambda_u(v')$. Hence, for the depth-first search we need to perform at most 4 ray shooting queries and at most 4 insert/delete operations at each vertex of the 1-skeleton.

Again, we can find in $O(n \log n)$ time the first vertex on Π_k from which we initiate the depth-first search. Hence the total running time of the algorithm is $O((n+b)n^\epsilon)$, which proves part (i) of Theorem 7.1.

The above algorithm can be extended to higher dimensions. In particular, the k-level in an arrangement of n hyperplanes in $\mathbb{R}^d$ can be computed in time $O\left(n^{1+\epsilon} + bn^{1-\frac{2}{\lfloor d/2 \rfloor + 1}+\epsilon}\right)$. In fact, if the value of b is large, one can improve the running time to $O(n^{\frac{d}{d+1}+\epsilon} b^{\frac{d}{d+1}} + n^{1+\epsilon})$ using a different data structure, see [3].

References

[1] P. K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *SIAM J. Computing* 21 (1992), in press.

[2] P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, Technical Report CS-1991-22, Dept. Computer Science, Duke University, 1991.

[3] P. K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, Technical Report, CS-1991-43, Dept. Computer Science, Duke University, 1991.

[4] P. K. Agarwal and M. Sharir, Planar geometric location problems, Tech. Rept. 90-58, DIMACS, Rutgers

University, August 1990. (Also to appear in *Algorithmica.*)

[5] P. K. Agarwal and M. Sharir, Applications of a new partitioning scheme, *Proc. 2nd Workshop on Algorithms and Data Structures*, 1991, pp. 379–392.

[6] B. Aronov, B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and R. Wenger, Points and triangles in the plane and halving planes in the space, *Discrete & Computational Geometry*, 6 (1991), 435–442.

[7] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld, Efficient ray shooting and hidden surface removal, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 51–60.

[8] M. de Berg and M. Overmars, Hidden Surface Removal for Axis-Parallel Polyhedra, *Proceedings* 31^{st} *Annual IEEE Symposium on Foundations of Computer Science*, 1990, pp. 252–261.

[9] B. Chazelle, On the convex layers of a planar set, *IEEE Trans. Information Theory* IT–31 (1985), 509–517.

[10] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir and J. Stolfi, Lines in space: Combinatorics and algorithms, *Proc. 21. ACM Symposium on Theory of Computing*, 1989, pp. 389–392. Full version: Tech. Rept. 491, Dept. of Computer Science, New York University, February 1990.

[11] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Discrete Comput. Geom.* 4 (1989), 551–589.

[12] B. Chazelle and F. P. Preparata, Halfspace range searching: An algorithmic application of k-sets, *Discrete & Computational Geometry*, 1 (1986), 83–93.

[13] B. Chazelle, M. Sharir and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, *Proc. 6th ACM Symp. on Computational Geometry*, 1990, pp. 23–33.

[14] K. Clarkson, A randomized algorithm for closest point queries, *SIAM J. Computing* 17 (1988), 830–847.

[15] K. L. Clarkson and P. Shor, New applications of random sampling in computational geometry II, *Discrete & Computational Geometry*, 4, 1989.

[16] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *J. ACM* 31 (1984), 200–208.

[17] D. Dobkin and D. Kirkpatrick, Determining the separation of preprocessed polyhedra: a unified approach, *Proceedings* 17^{th} *International Colloquium on Automata, Languages and Programming*, 1990, pp. 400–413.

[18] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[19] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, *SIAM J. Computing* 15 (1986), 271–284.

[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, 1989.

[21] J. Matoušek, Efficient partition trees, *Proc. 7th ACM Symp. on Computational Geometry*, 1991, pp. 1–9.

[22] J. Matoušek, Reporting points in halfspaces, *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, 1991.

[23] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th ACM Symposium on Computational Geometry*, 1992. To appear.

[24] J. Matoušek and O. Schwarzkopf. Linear optimization queries. In *Proc. 8th ACM Symposium on Computational Geometry*, 1992. To appear.

[25] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM* 30 (1983), 852–865.

[26] K. Mulmuley, On levels in arrangements and Voronoi diagrams, *Discrete & Computational Geometry*, 6 (1991), 307–338.

[27] K. Mulmuley, Randomized multidimensional search trees: Further results in dynamic sampling, *Proceedings 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991, pp. 216–27.

[28] M. Overmars and M. Sharir, Output-sensitive hidden surface removal, *Proc. 30th IEEE Symp. on Foundations of Computer Science*, 1989, pp. 598–603.

[29] J. Pach, W. Steiger, and E. Szemerédi, An upper bound on the number of planar k-sets, *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 72–79.

[30] O. Schwarzkopf. Ray shooting in convex polytopes. Technical Report B–91–18, FB Mathematik, Freie Universität Berlin, 1991.

[31] D. Sommerville, *Analytical Geometry in Three Dimensions*, Cambridge, 1951.

[32] S. Vrećica and R. Živaljević, The colored Tverberg's problem and complexes of injective functions, Manuscript, 1991.