On Ray Shooting in Convex Polytopes*

Jiří Matoušek

Katedra aplikované matematiky, Universita Karlova
Malostranské nám. 25, 118 00 Praha 1, Czechoslovakia
and
Institut für Informatik, Freie Universität Berlin
Arnimallee 2-6, W 1000 Berlin 33, Germany

Otfried Schwarzkopf

Universiteit Utrecht, Vakgroep Informatica,
Postbus 80·089, 3508 TB Utrecht, the Netherlands

Abstract

Let $\mathcal{P}$ be a convex polytope with n facets in the Euclidean space of a (small) fixed dimension d. We consider the *membership* problem for $\mathcal{P}$ (given a query point, decide whether it lies in $\mathcal{P}$) and the *ray shooting* problem in $\mathcal{P}$ (given a query ray originating inside $\mathcal{P}$, determine the first facet of $\mathcal{P}$ hit by it). It was shown in [AM92] that a data structure for the membership problem satisfying certain mild assumptions can also be used for the ray shooting problem, with a logarithmic overhead in query time. Here we show that some specific data structures for the membership problem can be used for ray shooting in a more direct way, reducing the overhead in the query time and eliminating the use of parametric search.

We also describe an improved static solution for the membership problem, approaching the conjectured lower bounds more tightly.

1 Introduction

Let d be a fixed (small) integer, and let $\mathcal{P}$ be a convex polytope in $\mathbb{E}^d$ with at most n facets, i.e. an intersection of n halfspaces. We will consider the following algorithmic problems for $\mathcal{P}$, listed in the order of increasing generality:

- The *membership problem.* We want to build some data structure (storing some information about $\mathcal{P}$), such that for a given query point, we can quickly decide whether it lies in $\mathcal{P}$.
- The *membership with witness* problem. This is as the membership problem, but if the answer to the query is NO (the query point lies outside $\mathcal{P}$), we require that one of the n halfspaces defining $\mathcal{P}$ is returned, which does not contain the query point.
- The *ray shooting* problem. In this case the query is specified by a ray ρ, originating inside $\mathcal{P}$, and the goal is to find the first facet of $\mathcal{P}$ crossed by the ray[1].

*Part of work on this paper by J.M. was supported by Humboldt Forschungsstipendium. O. S. acknowledges support by the ESPRIT II Basic Research Action of the European Community under contract No. 3075 (project ALCOM). This research was done while he was employed at Freie Universität Berlin. Furthermore, part of this research was done while he visited INRIA-Sophia Antipolis.

[1] We say that a ray ρ *crosses* a hyperplane h if it intersects it but it is not contained in it. Thus, it makes sense to consider also rays which go inside some face of $\mathcal{P}$, and we permit such rays.

The membership problem has been intensively studied in computational geometry (mainly in dimensions 2 and 3), and it falls into the area known as *point location problems*. From another perspective, this problem can be viewed as belonging to *range searching*, and, indeed, the known efficient solutions for this membership problem come from research in range searching (at least those for dimension $d \geqslant 4$). The point location and range searching problems are of some interest for direct practical applications, but they are still more significant as subroutines used in other problems.

Our main goal will be to show that certain data structures for the membership problem for $\mathcal{P}$ can also be used (perhaps slightly augmented) for ray shooting in $\mathcal{P}$. Ray shooting problems (which can also be formulated in many different settings) are also quite significant, e.g. they play an important role in various hidden surface removal algorithms and related problems directly motivated by computer graphics. Ray shooting in a convex polytope can also be applied to solve the so-called post-office problem, see below.

The membership problem with witness is intermediate between membership and ray shooting. Rather surprisingly, the known data structures for the membership problem do not always provide a witness, and it seems nontrivial to find one. Finding a witness is essential in some applications of the membership testing, see [MS92].

In this paper we will review some known data structures for the problem of membership in a convex polytope, and we add one new data structure, which is somewhat more efficient than the previously known ones. Our main goal will be to extend known data structures for the membership problem, so that they are capable of handling the membership with witness and ray shooting.

The observation that ray shooting queries are somewhat related to point location and range searching is implicitly present in many papers. For instance, de Berg et al. [dBH*] developed sophisticated methods for ray shooting using range searching structures. Agarwal and Matoušek [AM92] observed that the transformation of a suitable range searching algorithm into an algorithm for ray shooting queries is possible under quite general conditions, if one employs a sophisticated algorithmic technique, so-called parametric search (due to Megiddo [Meg83]). In particular, they obtained the first nontrivial algorithms for ray shooting in convex polytopes for an arbitrary dimension by this technique.

This application of parametric search makes the resulting ray shooting algorithm somewhat complicated, both conceptually and practically. Moreover, it typically increases the query time by a logarithmic factor.

We will show that for some known data structures for the membership in a convex polytope, one can naturally pass to ray shooting without parametric search, only by a slight modification of the query answering algorithm and/or by augmenting the data structure. We then obtain query time for ray shooting or witness finding of exactly the same order as for the membership problem.

Let us now review the known and new results for the considered problems in a quantitative form (see also Tab. 1). We will distinguish static solutions (where the set of halfspaces defining the polytope $\mathcal{P}$ is fixed once and for all) and dynamic ones (where we allow insertions and deletions of halfspaces defining $\mathcal{P}$). As it is the case with many problems of this type, slightly more efficient solutions are known for the static case than for the dynamic one.

For the membership in a convex polytope, Clarkson [Cla87] gave a (static) solution with $O(\log n)$ query time, which uses $O(n^{\lfloor d/2 \rfloor + \delta})$ space and expected preprocessing time[2]. In [AM91] a dynamic counterpart of this data structure was given, with asymptotically the same space and query time performance, and with $O(n^{\lfloor d/2 \rfloor - 1 + \delta})$ amortized update time[3].

We will describe a static data structure with $O(\log n)$ query time and with the space and preprocessing time requirement improved to $O(n^{\lfloor d/2 \rfloor} / \log^{\lfloor d/2 \rfloor - \delta})$. We apply the technique of

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

[3] By saying that a data structure has $f(n)$ amortized update time we mean that starting with a current structure storing a set of n hyperplanes, an arbitrary sequence of at most n insertions and deletions can be performed in at most $nf(n)$ time.

space; preprocessing	update time (amortized)	ray shooting queries
n; $n \log n$	$\log^2 n$	$n^{1-1/\lfloor d/2 \rfloor+\delta}$
$n^{\lfloor d/2 \rfloor+\delta}$	$n^{\lfloor d/2 \rfloor-1+\delta}$	$\log n$
$m^{1+\delta}$	$m^{1+\delta}/n$	$\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n$
n; $n \log n$	N/A	$n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)}$
$m(\log n)^{O(1)}$	N/A	$\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n$
$n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor-\delta}$	N/A	$\log n$

(m is an adjustable parameter, $n \leqslant m \leqslant n^{\lfloor d/2 \rfloor}$)

Table 1: Summary of data structures for ray shooting in a convex polytope

bootstrapping developed by Chazelle and Friedman [CF] (they used it for point location among hyperplanes).

Why should such a small improvement be interesting? The worst-case combinatorial complexity of a polytope with n facets is $\Theta(n^{\lfloor d/2 \rfloor})$, and this is an intuitive reason to conjecture that for membership queries with logarithmic time, the space requirements should be quite close to $n^{\lfloor d/2 \rfloor}$. A more daring form of this conjecture says that if $S(n)$ denotes the space and $Q(n)$ denotes the worst-case query time of a data structure for the membership problem, then $Q(n) \cdot S(n)^{1/\lfloor d/2 \rfloor} = \Omega(n)$. Nothing even approaching a proof of this conjecture is known, but the work of Chazelle [Cha89] (which established a similar lower bound for simplex range searching, with d replacing $\lfloor d/2 \rfloor$), and of Brönnimann and Chazelle [BC92] (who proved a somewhat weaker lower bound for halfspace range searching) give at least some ideological support to this conjecture. From this point of view, our algorithm would already be quite close to optimal, since it has $Q(n)S(n)^{1/\lfloor d/2 \rfloor} = O(n(\log n)^{\delta})$. Clarkson's solution gives $O(n^{1+\delta})$ for this product, and another solution by Mulmuley [Mul91] yields $O(n(\log n)^c)$ for a (not too small) constant c.

The previous paragraphs consider solutions to the membership problem with a logarithmic query time. At the other end of the spectrum, there is a solution by [Mat92b], with $O(n)$ space, $O(n \log n)$ preprocessing time and $O(n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)})$ query time. A dynamic version with $O(n)$ space, $O(n^{1-1/\lfloor d/2 \rfloor+\delta})$ query time and $O(\log^2 n)$ amortized update time is given in [AM91], as well as a dynamic version of Clarkson's data structure. Finally the linear-space solutions can be easily combined with the previously mentioned large-space ones, obtaining a continuous tradeoff between space and query time, of the following form: Given a parameter m, $n \leqslant m \leqslant n^{\lfloor d/2 \rfloor}$, one can build a static data structure using $O(m(\log n)^{O(1)})$ space and with $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ query time, or a dynamic data structure with $O(m^{1+\delta})$ space and preprocessing time, $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$ query time and $O(m^{1+\delta}/n)$ amortized update time.

Employing parametric search, one can use any of the above mentioned data structures for the membership problem to answer ray shooting queries. The storage, preprocessing and update time remain the same (as well as the whole data structure), but the query time increases by a polylogarithmic factor [AM92]. In this paper, we will obtain ray shooting algorithms for all these data structures, all of whose parameters (query time, storage, preprocessing time and update time) are *of the same order* as the ones for the membership problem (given above). In particular, we can solve the membership problem with witness with these parameters. The bounds are summarized in Table 1.

Note that by a well known lifting transformation (see e.g. [Ede87]), nearest (and furthest) neighbor queries in dimension $d-1$ can be solved by shooting a vertical ray in the intersection of n halfspaces in $\mathbb{E}^d$. We obtain a data structure for the post office problem, which is an improvement over previous results. Even in three-dimensional space, for instance, the best previous solution with space $O(n^2)$ had query time $O(\log^2 n)$, see [Cha85, AESW90]. We improve the query time to

$O(\log n)$.

Let us mention some open problems. In our ray shooting algorithms, we require that the query ray originates within the polytope $\mathcal{P}$ (which is the case in most applications). The general technique with parametric search can handle also rays originating outside $\mathcal{P}$ (see [AM92]); it would be interesting to extend our approach also to this situation.

The passage from membership queries to ray shooting can be viewed as giving one degree of freedom to the (originally fixed) query point, and performing one-dimensional optimization along the query ray. A natural further generalization considered in [MS92] is to consider a query k-flat (for some $k > 1$), and ask whether it intersects the polytope $\mathcal{P}$, or ask for the optimum of some linear function over the intersection of the query k-flat with $\mathcal{P}$. This problem can indeed be solved using a data structure for the membership problem. In [MS92], this is done by a k-wise application of a multidimensional variant of parametric search, which produces a rather mysterious algorithm and quite large overhead in query time. It would thus be interesting to extend the methods of the present paper for handling also the queries with a k-flat and avoid the parametric search also there.

Another problem to consider is the ray shooting in an arrangement of n hyperplanes in $\mathbb{E}^d$. Applying parametric search to a suitable point location algorithm (e.g., that of Chazelle [Cha]), one can get $O(\log^2 n)$ query time with $O(n^d)$ space (or with $O(n^d/\log^{d-\delta})$ space, with some more tricks). It would be interesting to get $O(\log n)$ query time with $O(n^d)$ or smaller space. This problem is solved in [CF] for *vertical* rays, but the solution does not extend to general ray shooting (see also [Mat]).

2 Algorithms with almost linear space

In this section it will be more convenient to consider dual versions of the problems discussed in the introduction. Under a suitable duality transform, the set H of hyperplanes extending the facets of the polytope $\mathcal{P}$ becomes a set P of points. The membership problem then translates into the following query problem: Given a query halfspace γ, decide whether it is disjoint from P. The witness in the membership problem becomes simply a witness point of P contained in γ. We call this the *halfspace emptiness* problem.

We consider the linear-space data structures for the halfspace emptiness problem of [Mat92b] and its dynamization given in [AM91]. Neither of these data structures provides a witness directly. We show how to find one, and how to handle the (dual version of) ray shooting.

We start with several definitions. Let P be an n-point set in $\mathbb{E}^d$. A *simplicial partition* of P is a collection

$$\Pi = \{(P_1, \Delta_1), \ldots, (P_m, \Delta_m)\},$$

where the P_i's are nonempty sets (called the *classes* of Π) forming a partition of P and each Δ_i is a relatively open simplex (not necessarily full-dimensional) containing P_i. The number m is the *size* of the partition.

The basis of the efficient algorithm for the halfspace emptiness problem in [Mat92b] is the so-called Partition theorem for shallow hyperplanes. We say that a hyperplane h is *k-shallow* (relative to P), if one of the halfspaces bounded by h contains no more than k points of P in its interior. We say that a hyperplane h *crosses* a simplex Δ if $\Delta \cap h \neq \emptyset$ but $\Delta \not\subseteq h$. For our purposes here, we use the following form of the theorem:

Theorem 1 [Mat92b] (Partition theorem for shallow hyperplanes) *Let P be a set of n points in $\mathbb{E}^d$, $d \geqslant 3$, and r a parameter, $1 \leqslant r \leqslant n/2$. Then there exists a simplicial partition Π of P having size $O(r)$ with classes of size between n/r and $2n/r$, and such that any (n/r)-shallow hyperplane h crosses at most $O(r^{1-1/\lfloor d/2 \rfloor} + \log r)$ simplices of Π. For $r \leqslant n^{\alpha}$, where $\alpha > 0$ is a certain constant (depending on the dimension d), such a simplicial partition can be found in time $O(n \log r)$.*

The preprocessing for a basic version of the halfspace emptiness algorithm consists of building a *partition tree* using the above theorem. Each node v of the tree corresponds to some subset $P_v \subseteq P$, the root corresponds to the whole set P. We choose a suitable large constant r, we find a simplicial partition of P of size $O(r)$ as in the above theorem and we store the description of the simplices of this partition in the root of the tree. For every subset P_i of this partition, we then build one subtree of the root in the same manner. The construction ends in the leaves of the tree, where the size of the subsets drops below some constant; then we explicitly store all the points of such a subset in the corresponding leaf.

The query answering algorithm deciding the emptiness of an open halfspace γ with a bounding hyperplane h then uses the partition tree as follows: We start in the root of the tree, and we determine the simplices of the simplicial partition stored there intersecting γ. If the number of such simplices is greater than the guaranteed maximal number $\kappa = O(r^{1-1/\lfloor d/2 \rfloor} + \log r)$ of simplices crossed by any (n/r)-shallow hyperplane, then γ is nonempty, in fact it contains at least n/r points of P (since either there is a simplex completely contained in γ, or more than κ simplices cross h, implying that h is not (n/r)-shallow). Otherwise we proceed recursively down the tree into the at most κ children of the current node for which the corresponding simplices intersect γ. This recursion terminates in leaves of the partition tree, where we simply check if any of the points stored there is contained in γ.

A straightforward analysis of this data structure shows that it requires $O(n)$ space, $O(n \log n)$ preprocessing time and $O(n^{1-1/\lfloor d/2 \rfloor + \delta})$ query time, where $\delta > 0$ tends to 0 with increasing r. A dynamic version of this data structure is described in [AM91]. First, it is shown that one can perform deletions of points from this data structure (rebuilding subtrees from which many points have been deleted). Insertions are then handled in a standard way (the "binary counting" pattern of Bentley and Saxe [BS80]), using the decomposability of the halfspace emptiness problem. The resulting (amortized) update time is $O(\log^2 n)$.

We will now discuss several ways to find a witness using this data structure; later we will generalize the methods also to ray shooting. The problematic situation is the following:

> In the query answering algorithm, we are at an inner node v, and we find that γ intersects more than κ simplices of the simplicial partition in v.

We say that such a halfspace γ is *deep* for v. By the argument above, we have $|\gamma \cap P_v| > n_v/r$, $n_v = |P_v|$, for any deep halfspace γ. In the dynamic version of [AM91], the points get deleted from P_v, but after at most $n_v/2r$ deletions from P_v the subtree rooted at v is reconstructed from scratch, so in this case a deep halfspace always contains at least $n_v/2r$ points of P_v. We describe three methods for finding a witness point of P_v contained in a deep query halfspace.

A simple randomized algorithm. When we reach a node v for which γ is deep, we pick a random point p in P_v. By the above, the probability that $p \in \gamma$ is at least $1/2r$, which implies that the probability that we do not hit a point in γ after $O(r \log n)$ trials is very small (less than $1/n^c$, for a constant $c > 0$ that can be made arbitrarily large). A random point in P_v can be found in time $O(\log n_v)$ without having to store the set P_v explicitly: if we store the number of points in P_v with each node, we can just choose a random index in $\{1, \ldots, n_v\}$ and track down the point in the corresponding subtree, in $O(\log n_v)$ time. The time for witness finding is thus dominated by the total query time with high probability. We needed no auxiliary data structures (except for the point counts in nodes, whose maintenance is trivial), and so this query answering algorithm can be used for the dynamic version as well.

A deterministic algorithm (static version). The key to a deterministic query algorithm providing a witness is the notion of ε-nets. A $(1/r)$-*net* for an n-point set P is a subset $R \subset P$ with the property that every halfspace γ with $|\gamma \cap P| \geqslant n/r$ contains a point of R. We use the following fact.

Fact 2 *Let P be a set of n points in $\mathbb{E}^d$, $r \leqslant n$ a parameter. There exists a $(1/r)$-net R of P with*

$|R| = O(r \log r)$ [HW87], *and it can be computed in time* $O(n \log r)$ *time if* $r \leqslant n^{\alpha}$, *where* $\alpha > 0$ *is certain constant, depending on the dimension* [Mat92a].

We augment every inner node v of the partition tree as follows: We find a $(1/r)$-net R_v of size $O(r \log r)$ for P_v, and store it at v. When we reach a node v in a query with γ for which γ is deep, we test all points $p \in R_v$ for containment in γ. Since γ contains at least n_v/r points of P_v, it must contain some point of R_v. Such a point, the witness, is thus found in additional time $O(|R_v|) = O(r \log r)$. Query time and storage of the structure thus remain the same as before, and this also holds for the preprocessing time (since R_v can be computed in time linear in the size of P_v).

For a static data structure, the query time can be slightly improved, as in [Mat92b]. We will consider the case $d \geqslant 4$. Instead of setting r to a large enough constant, we set $r = n_v^{\beta}$ in a node v, where $\beta > 0$ is a suitable sufficiently small constant. Otherwise the data structure remains exactly the same. The space is still linear, the preprocessing time is $O(n \log n)$ and the query time $O(n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)})$, also when we count the additional witness finding time (this can be derived by analyzing simple recurrences, see [Mat92b]). We do not know how to dynamize this improved data structure efficiently, so in the next algorithm we will return to constant values of r at every node.

A deterministic algorithm (dynamic version). In order to use the previous idea with ε-net in a dynamic setting, we would need to maintain an ε-net dynamically. This is nontrivial, since if we delete a point belonging to the current ε-net, this ε-net has to be repaired somehow. Fortunately we can replace ε-nets by suitable simplicial partitions, which behave as "robust ε-nets" in some respect.

Above we have considered simplicial partitions for which every k-*shallow* hyperplane crosses possibly few simplices, for a suitable k. Here we will need simplicial partitions for which *any* hyperplane crosses few simplices only. The following is shown in [Mat92a]:

Lemma 3 [Mat92a] *For every* n*-point set* P *in* $\mathbb{E}^d$ *and a parameter* t $(1 < t \leqslant n/2)$, *there exists a simplicial partition* Ψ *of size* $O(t)$ *for* P, *with class sizes between* n/t *and* $2n/t$, *and such that every hyperplane crosses at most* $\xi = O(t^{1-1/d})$ *simplices of* Ψ. *For* t *bounded by a constant, such a* Ψ *can be computed in time* $O(n)$.

Let us consider such a simplicial partition Ψ for P, and let γ be an open halfspace which contains none of the simplices of Ψ completely. Since the bounding hyperplane of γ crosses at most ξ simplices of Ψ, all but ξ simplices of Ψ are disjoint from γ. Hence γ contains at most $k = 2n\xi/t = O(n/t^{1/d})$ points of P. In other words, an open halfspace containing more than k points of P completely contains some simplex of Ψ.

We now return to our partition tree and apply this as follows. In an inner node v, set $t = Cr^d$, for a large enough constant C, and build an auxiliary simplicial partition Ψ_v for the set P_v, as in Lemma 3. The constant C is chosen in such a way that every open halfspace γ containing at least $n_v/2r$ points of P_v completely contains some simplex of Ψ_v. The simplicial partition Ψ_v will be our auxiliary data structure stored at v.

Suppose that we came to a node v with a query halfspace γ which turned out to be deep for v (so γ contains some simplices of Ψ). Let $\gamma' \subseteq \gamma$ be the open halfspace whose boundary is parallel to the boundary of γ, and such that it does not contain the closure of any simplex of Ψ_v but it contains some (relatively open) simplex Δ of Ψ_v. Thus, γ' arises by shifting γ inwards, until the boundary hits the last simplex of Ψ_v remaining in the shifted halfspace. Obviously γ' is nonempty, since it contains the points of P_v in Δ. On the other hand, $|P_v \cap \gamma'| < n_v/2r$: if it were not the case, then we might shift the (open) halfspace γ' a little bit more, obtaining a halfspace γ'' with $\gamma'' \cap P_v = \gamma' \cap P_v$ but such that no simplex of Ψ_v is contained in γ'', which is impossible.

Thus having a deep halfspace γ for a node v, we find γ' as above (in $O(t \log t) = O(1)$ time), and we continue the query with γ'. This is contained in γ, intersects P_v but it is not deep for v anymore, so it intersects only $\kappa = O(r^{1-1/\lfloor d/2 \rfloor} + \log r)$ simplices of Π_v, and we can continue with the recursion in the appropriate children of v. Eventually we reach a leaf with a nonempty query halfspace, and we find the desired witness there.

It remains to show how to maintain the simplicial partitions Ψ_v under deletions of points. This is not difficult: Whenever at least $n_v/2t$ points of P_v are deleted in a node v, we recompute the simplicial partition Ψ_v anew, in $O(n_v)$ time. The amortized time is constant per deletion in one node, and $O(\log n)$ for the whole data structure. At the same time, these reconstructions guarantee that none of the simplices of Ψ_v becomes empty, which is sufficient for a correct function of the algorithm. As for insertions, these need not be considered (since they are handled at a higher level of the dynamic data structure of [AM91]), but we could handle them easily anyway.

Note that we could also store the list of points of P_v belonging to every simplex of the auxiliary simplicial partition Ψ_v, and thus find a witness for a deep halfspace without further recursion. But storing the lists of points in every node would raise the space requirements to $O(n \log n)$. Also, the above approach generalizes to ray shooting, as we will now explain.

Ray shooting algorithms. For a point set P and a nonvertical hyperplane h, let us call a point $p \in P$ *h-extreme* if there is no point of P above the hyperplane passing through p and parallel to h (let us denote this hyperplane by $h(p)$). Let us consider the problem of finding an h-extreme point of P for a query hyperplane h. This is the dual counterpart of the ray shooting problem with vertical rays emanating from $+\infty$.

It turns out that some of the data structures developed for halfspace emptiness queries with witness can be used to find an h-extreme point quite easily. Let us consider the data structure for the static deterministic algorithm (the one with ε-nets). Given a query hyperplane h, we start in the root of the partition tree. In an inner node v, we proceed as follows: we find an h-extreme point q of R_v (R_v is the $(1/r)$-net at v). Then we recursively find the h-extreme points for all children of v corresponding to simplices of Π_v intersecting the halfspace above $h(q)$ (if there is no such simplex, then q itself is h-extreme). Among these, we select an h-extreme one. In a leaf, an h-extreme point is selected trivially. This algorithm is obviously correct, and the $(1/r)$-net property of R_v guarantees that $h(q)$ is an n/r-shallow hyperplane (relative to P_v). Hence we recurse in at most κ children of v. Thus the running time analysis is the same as for the basic halfspace emptiness algorithm and we also get the same query time.

The h-extreme point problem can also be solved using the data structure for the dynamic deterministic algorithm (the one using auxiliary simplicial partitions). In fact, this is exactly what the above described query answering algorithm does, only we need to select the h-extreme point from all the leaves reached in the recursion.

As we pointed out, the h-extreme point problem is equivalent to vertical ray shooting from the inside of a convex polytope. The general ray shooting problem dualizes to the following problem: We are given a query (nonvertical) $(d-2)$-flat a, such that there is a hyperplane h_0 passing through a and having no points of P above it. We want to find an *a-rotation extreme point* of P, which is a point $p \in P$ such that there are no points of P above the hyperplane passing through p and a. The previously considered problem corresponds to the situation when a is formally at infinity. One can find an a-rotation extreme point in almost the same way as we did for an h-extreme point; we leave details to the reader. Hence we can solve the ray shooting problem with the same complexity as the halfspace emptiness problem.

We have thus given a new proof of the following result of [AM91, AM92]:

Theorem 4 *The ray shooting problem for a convex polytope with n facets in $\mathbb{E}^d$ can be solved with $O(n)$ space, $O(n \log n)$ preprocessing, $O(\log^2 n)$ amortized update time and $O(n^{1-1/\lfloor d/2 \rfloor + \delta})$ query time. For a static data structure, the query time can be improved to $O(n^{1-1/\lfloor d/2 \rfloor}(\log n)^{O(1)})$.*

3 Ray shooting with logarithmic query time

Another data structure for the polytope membership problem is a specialization of a result due to Clarkson [Cla88]; let us recall that it requires $O(n^{\lfloor d/2 \rfloor + \delta})$ space and preprocessing time, and achieves $O(\log n)$ query time. A dynamic counterpart of this data structure was given in [AM91]. We will explain a static data structure (similar to Clarkson's one) and the way we can find a witness point using this structure. Then we generalize to the ray shooting case.

We return to the primal setting of the problems, as discussed in the introduction. We may and will assume that the polytope $\mathcal{P}$ is the upper unbounded cell in an arrangement of a set H of n hyperplanes, i.e. the set of points lying above all hyperplanes of H. In this situation the membership problem takes the following special form: "Given a query point, determine whether it lies *above* all hyperplanes of H".

Let $r \leqslant n$ be a parameter. For the sake of simplicity, we will assume that the hyperplanes of H are in general position. Let the *level* of a point $p \in \mathbb{E}^d$ with respect to H be the number of hyperplanes $h \in H$ lying strictly above it. A collection Ξ of simplices with disjoint interiors will be called a $(1/r)$-*cutting for the* $(\leqslant 0)$-*level of* H, provided that the simplices of Ξ cover all points of level 0 (with respect to H, i.e. all points with no hyperplanes of H lying strictly above them), and that each simplex of Ξ is intersected by at most n/r hyperplanes of H. We will apply the following result:

Theorem 5 (Shallow cutting lemma, [Mat92b]) *Let H, r be as above. Then there exists a $(1/r)$-cutting Ξ for the $(\leqslant 0)$-level of H, consisting of $O(r^{\lfloor d/2 \rfloor})$ simplices. For $r \leqslant n^{\alpha}$ (where $\alpha > 0$ is a certain constant, dependent on the dimension), such a cutting can be computed in $O(n \log r)$ time.*

For every simplex $\Delta \in \Xi$, we say that a hyperplane $h \in H$ is *relevant* for Δ if it lies above Δ or intersects Δ. Let H_{Δ} denote the collection of hyperplanes relevant for Δ.

A slightly modified Clarkson's structure for the membership problem is a tree-like structure, built recursively as follows: If the number of hyperplanes in H is smaller than a suitable constant, then one simply stores the list of hyperplanes of H; this will be a leaf node. A query is answered by testing the query point against each hyperplane of H.

If, on the other hand, H is larger, one chooses a suitable parameter r (a sufficiently large constant in Clarkson's original construction), and computes a $(1/r)$-cutting Ξ for the $(\leqslant 0)$-level of H, consisting of $O(r^{\lfloor d/2 \rfloor})$ simplices. We store the cutting Ξ in the root of the data structure, and for every $\Delta \in \Xi$, we recursively build a subtree corresponding to the data structure for H_{Δ}. The space $S(n)$ occupied by this data structure obeys the recursion

$$S(n) \leqslant O(r^{\lfloor d/2 \rfloor}) + O(r^{\lfloor d/2 \rfloor}) S(n/r)\,,$$

which for a sufficiently large but constant value of r solves to $O(n^{\lfloor d/2 \rfloor + \delta})$.

We now describe a query answering algorithm finding a witness as well. A query with a point q on H is answered as follows: We begin at the root. Being in a non-leaf node v, we determine whether q belongs to some simplex of Ξ_v, the cutting stored at v, and if there is such a simplex, we proceed recursively into the corresponding child of v. If there is no such simplex, it means that there is a hyperplane strictly above q.

In this situation, we shoot a vertical ray ρ from q upwards, we find the first simplex Δ of Ξ_v intersected by ρ and we recurse in the child corresponding to Δ. We claim that some of the hyperplanes relevant for Δ must lie above q. Indeed, let us take the last hyperplane h encountered along ρ in the upward direction. The point $q \cap \rho$ has level 0, and so it has to be contained in some simplex. Therefore Δ contains this point or some point below it, and so h is relevant for Δ, showing the correctness of this step.

In a leaf node, we solve the query by inspecting all the hyperplanes in that node. This finishes the query algorithm. Since we spend a constant time in every node, the query time is $O(\log n)$.

The same method works also for the dynamic data structure of [AM91] and provides a witness hyperplane, while retaining the same $(O(\log n))$ query time. (The reader has to be familiar with the dynamic data structure in order to check our claim, since we will not explain the dynamic data structure. We apologize for withholding this information, but the data structure in question is technically fairly complicated.) Since the dynamic data structure requires a larger (nonconstant) value of r, we need some auxiliary data structure to find the first simplex of a cutting Ξ crossed by the vertical ray ρ. This is done by point location in the projection of the cutting Ξ on a horizontal hyperplane. The combinatorial complexity of this projection is bounded by a polynomial in the size of Ξ, and since this size is a very small fractional power of n, the complexity is still small compared to n. We can thus afford quite a wasteful approach: we extend every $(d-2)$-dimensional feature in this projection into a full hyperplane, and we use an algorithm for point location among hyperplanes with a logarithmic query time and a polynomial space (see e.g., [CF]). This additional data structure does not increase the asymptotic space or query time requirements.

Let us now turn to the ray shooting problem. We use the same data structure as for the membership problem, but with a slightly modified query algorithm. For the static case, assume we are given a query ray ρ with origin p lying above all hyperplanes in H. We begin the query answering at the root of the data structure. Being in a non-leaf node v, we determine the *last* simplex Δ of Ξ_v intersected by ρ. Since p lies above all hyperplanes of H, it is contained in a simplex of Ξ_v, hence such a last simplex Δ exists. We proceed recursively into the child of v corresponding to Δ. In a leaf node, we simply test all hyperplanes to find the first one crossed by the ray ρ.

To prove the correctness of this algorithm, we have to show that the first hyperplane h crossed by ρ is relevant for Δ. The argument is similar to the one employed above: Consider the point $\rho \cap h$. It has level 0 and must thus be contained in some simplex. Thus, Δ contains this point or a point on ρ lying *behind* it (in the order along the ray ρ). Since h contains the point $\rho \cap h$, h must be relevant for Δ.

In the dynamic data structure, the size of the cuttings used is larger and we need a secondary point location structure to find the last simplex of Ξ_v intersected by a query ray. It is sufficient to have a data structure with $O(\log s)$ query time and space and preprocessing time polynomial in s, the size of Ξ. This is discussed in the Appendix. We thus get

Theorem 6 *The ray shooting problem for a convex polytope with n facets in $\mathbb{E}^d$ can be solved with $O(n^{\lfloor d/2 \rfloor + \delta})$ space and preprocessing time, $O(\log n)$ query time, and $O(n^{\lfloor d/2 \rfloor - 1 + \delta})$ amortized update time.*

4 Reducing storage for static data structures with logarithmic query time

If we do not need a dynamic data structure, we can use the static data structure described in the previous section, but with the parameter r set to m^α in an m-point node, where $\alpha > 0$ is a suitable small constant. If α is chosen small enough, the storage requirement $S(n)$ follows the following recurrence: $S(n) = O(1)$ for $n \leqslant n_0 = O(1)$, and

$$S(n) = c_1 r^{\lfloor d/2 \rfloor} S(n/r) + c_2 n^{\lfloor d/2 \rfloor}$$

for some constants c_1, c_2. This solves to $S(n) = O(n^{\lfloor d/2 \rfloor} \log^c n)$, where the constant c depends on c_1, c_2. The query time follows a recursion of the form $Q(n) = Q(n^{1-\alpha}) + O(\log n)$, which gives $Q(n) = O(\log n)$.

Let us now turn to the preprocessing time. The shallow cutting Ξ can be constructed in time $O(n \log r)$ according to Theorem 5. Identification of the sets H_Δ can trivially be done in time $O(nr^{\lfloor d/2 \rfloor})$ by testing every hyperplane with every simplex of the cutting.

The time to construct the secondary structure on Ξ according to Lemma 11 (which is given in the Appendix) is polynomial in $r = n^\alpha$, and for sufficiently small $\alpha > 0$ all this is bounded by $O(n^{\lfloor d/2 \rfloor})$ again. Thus we get the same recursion as for the space requirement and find that the preprocessing time is also bounded by $O(n^{\lfloor d/2 \rfloor} \log^c n)$. We obtain

Lemma 7 *The (static) ray shooting problem in a convex polytope with n facets in $\mathbb{E}^d$ can be solved with preprocessing time and storage $O(n^{\lfloor d/2 \rfloor} (\log n)^{O(1)})$ and with query time $O(\log n)$.*

The partition tree data structures for ray shooting with linear space (as in Section 2) can be combined with the data structures with logarithmic query time to obtain a continuous space-query time tradeoff. Namely, we build only an upper part of the partition tree (as in the linear-space algorithm), stopping the recursive construction as soon as the size of the point set in a node drops below a suitably chosen number s, where $1 < s < n$. For such a node v, instead of building the subtree for P_v, we store the data structure with logarithmic query time for P_v at the node v. A similar construction appears in many previous papers (e.g., [CSW90]), so we omit the details. A somewhat worse tradeoff for the ray shooting problem is given in [AM92]. We obtain the following:

Theorem 8 *Given a parameter m ($n \leqslant m \leqslant n^{\lfloor d/2 \rfloor}$), there is a dynamic data structure for ray shooting queries in a convex polytope with n facets in $\mathbb{E}^d$ with space and preprocessing time $O(m^{1+\delta})$, amortized update time $O(m^{1+\delta}/n)$ and query time $O(\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log n)$. For a static data structure, the space and preprocessing time can be improved to $O(m(\log n)^{O(1)})$.*

We will now show that the idea of bootstrapping used by Chazelle and Friedman in [CF] can be used to improve the storage in Lemma 7 to $O(n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor - \delta})$. While [CF] needs four bootstrapping steps, we employ somewhat more powerful tools do obtain our result with a single such step. This idea could be used for the point location in hyperplane arrangements as well, and, in fact, the time bound given in [CF] can be slightly improved by following exactly the same ideas as presented here.

We need a lemma on random sampling due to Clarkson [Cla87], see also [CS89] . For a collection H of hyperplanes, let $\mathcal{U}(H)$ denote the set of all vertices of the upper envelope of H.

Lemma 9 *Let H be a set of n hyperplanes in $\mathbb{E}^d$ and $\beta > 0$ a constant. For a random sample R of r hyperplanes from H, with probability at least $1/2$ the following is true: For all vertices $v \in \mathcal{U}(R)$ the number n_v of hyperplanes $h \in H$ lying above v is bounded by $O(n \log r/r)$, and the sum of $\sum_{v \in \mathcal{U}(R)} n_v^\beta$ is bounded by $O(r^{\lfloor d/2 \rfloor}(n/r)^\beta)$.*

The improved structure can now be derived as follows: We take a random sample R of H of size $r = n/\log^k n$ for some parameter k (which is determined later), and build a structure according to Lemma 7 for ray shooting in the upper cell of the arrangement of R. Furthermore, for every vertex v of the upper envelope of R we identify the set H_v of hyperplanes $h \in H$ lying above v. Let $n_v = |H_v|$.

We then construct, for every such vertex v, a secondary structure as in Theorem 8 with storage $s = O(n_v^\beta (log n_v)^\gamma)$ (γ a constant) and query time $O(n_v^{1-\beta/\lfloor d/2 \rfloor})$ ($1 \leqslant \beta < \lfloor d/2 \rfloor$ is yet another constant to be chosen later[4]). By Lemma 9 we can choose R such that $n_v = O(\log^{k+1} n)$ for all vertices $v \in \mathcal{U}(R)$, and at the same time the total storage for the secondary structures is

$$\begin{aligned} \sum_{v \in \mathcal{U}(R)} n_v^\beta (\log n_v)^\gamma &= O(r^{\lfloor d/2 \rfloor}(n/r)^\beta (\log\log n)^\gamma) = \\ &= O(n^{\lfloor d/2 \rfloor}(\log n)^{k\beta - k\lfloor d/2 \rfloor}(\log\log n)^\gamma)\,. \end{aligned} \tag{1}$$

Consider now a query with a ray ρ with origin p above the upper envelope of H. Let h be the first hyperplane in R crossed by ρ, and let $q = \rho \cap h$. We determine a set V of vertices of the upper envelope of R such that q lies in the convex hull of V. This can be done by repeated ray shooting queries in the upper cell of the arrangement of R: We first find q and h by shooting with the ray ρ. Take then an arbitrary vertex p_1 of the upper envelope of R which is incident to h, and shoot with a ray ρ_1 with origin p_1 and going through q. This ray lies in h. Let h_1 be the first hyperplane in R crossed by ρ_1, and let q_1 be the point of intersection. Take an arbitrary point p_2 incident to h and h_1. Shoot again with the ray ρ_2 with origin p_2 and going through q_1, finding a new hyperplane $h_2 \in R$ and a point q_2. In this fashion we proceed until the intersection of $h \cap h_1 \cap h_2 \cap \cdots \cap h_i$ is a vertex p_{i+1}. Then we can take $V = \{p_1, p_2, \ldots, p_{i+1}\}$. The size of V is at most d.

[4] A weaker form of the tradeoff with storage $O(m^{1+\delta})$ would suffice, such as in [AM92].

We now query the secondary structures associated with the at most d vertices in V with the ray ρ, and output the first hyperplane crossed in any of the subproblems.

It remains to show that the first hyperplane h' in H crossed by ρ appears in H_v for some vertex $v \in V$. If the solution hyperplane h' is not identical to h, it must lie above the point q where ρ intersects h. Since q lies in the convex hull of the $v \in V$, h' lies above at least one of the vertices v.

Since $n_v = O(\log^{k+1} n)$, the query time of our combined structure is bounded by

$$O(\log n + (\log n)^{(k+1)(1-\beta/\lfloor d/2 \rfloor)}).$$

We choose $\beta = \lfloor d/2 \rfloor \frac{k}{k+1}$. This implies that the query time is bounded by $O(\log n)$. The storage for the primary data structure for ray shooting in the upper cell of the arrangement of R is (by Lemma 7) $O(r^{\lfloor d/2 \rfloor}(\log n)^{c_1})$, for some constant c_1. For k large enough, this will not exceed $O((n/\log n)^{\lfloor d/2 \rfloor})$. The storage for the secondary structures is bounded by

$$O(\frac{n^{\lfloor d/2 \rfloor}}{(\log n)^{\lfloor d/2 \rfloor - \lfloor d/2 \rfloor/(k+1)}}(\log \log n)^{\gamma}),$$

as we find from (1) and the choice of β. Choosing k large enough we get the total storage of order $O(n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor - \delta})$.

Let us turn to the cost of preprocessing. The problem is to find a sample R which conforms to the requirements of Lemma 9. Those can be checked once we know the number of hyperplanes in the lists H_v for every vertex $v \in \mathcal{U}(R)$ (these lists have to be computed anyway). We just take a random sample R of the appropriate size, and construct its upper envelope $\mathcal{U}(R)$. Then we sequentially take all n hyperplanes h from H and find all $v \in \mathcal{U}(R)$ which lie below h. It is well known that this can be done in time linear in the number of such vertices once one such vertex is known (see e.g. [CS89]). On the other hand, we get such a vertex by linear programming on R in time $O(r)$ (see [Sei91]), so the total time for this step is $O(nr)$ plus the total number of vertices found. We stop this algorithm if it tries to report more than $O(nr^{\lfloor d/2 \rfloor - 1})$ such vertices (because then the sample R is bad), otherwise we check whether it fulfills the requirements of Lemma 9. If not, we discard R and take a new random sample. By Lemma 9, the expected number of such trials is constant, so we can find R in expected time $O(nr^{\lfloor d/2 \rfloor - 1})$ (and identify the lists H_v in the same time). The secondary structures can be computed in time linear in their storage, so we have again the same recursion as for the space. We have thus proven

Theorem 10 *The (static) ray shooting problem in a convex polytope with n facets in $\mathbb{E}^d$ can be solved with preprocessing time and storage $O(n^{\lfloor d/2 \rfloor}/(\log n)^{\lfloor d/2 \rfloor - \delta})$ and with query time $O(\log n)$.*

Acknowledgments: Many thanks go to Emo Welzl for lots of helpful discussions. We would also like to thank Pankaj Agarwal for his comments on previous versions of this work. Furthermore, we are grateful to the anonymous referees for their constructive criticism.

References

[AESW90] Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proc. 6th ACM Symposium on Computational Geometry*, pages 203–210, 1990.

[AM91] Pankaj K. Agarwal and Jiří Matoušek. Dynamic half-space range reporting and its applications. Tech. Report CS-1991-43, Duke University 1991.

[AM92] Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. In *Proc. 24th ACM Symposium on Theory of Computing*, pages 517–526, 1992. Full version as Tech. Report CS-1991-22, Duke University 1991.

[BS80] Jon L. Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[dBH*] Mark de Berg, Dan Halperin, Mark Overmars, Jack Snoeyink, and Marc van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica.* To appear.

[BC92] Hervé Brönnimann and Bernard Chazelle. How hard is halfspace range searching? In *Proc. 8th Symp. on Computational Geometry*, pages 271–275, 1992.

[CEGS89] Bernard Chazelle, Herbert Edelsbrunner, Leonidas Guibas and Micha Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. In *Proc. 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 179–192, 1989.

[CF] Bernard Chazelle and Joel Friedman. Point location among hyperplanes and vertical ray shooting. *Computational Geometry: Theory and Applications.* To appear.

[Cha85] Bernard Chazelle. How to search in history. *Information and Control*, 64:77–99, 1985.

[Cha89] Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc*, 2(4):637–666, 1989.

[Cha] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry.* To appear.

[Cla87] Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.

[Cla88] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17:830–847, 1988.

[CS89] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[CSW90] Bernard Chazelle, Micha Sharir, and Emo Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proc. 6th Symp. on Computational Geometry*, pages 23–33, 1990.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[HW87] David Haussler and Emo Welzl. ϵ-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[Mat92a] Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[Mat92b] Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory & Applications*, 2:169–186, 1992.

[Mat] Jiří Matoušek. On vertical ray shooting in arrangements. *Computational Geometry: Theory & Applications.* To appear.

[Meg83] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.

[MS92] Jiří Matoušek and Otfried Schwarzkopf. Linear optimization queries. In *Proc. 8th ACM Symposium on Computational Geometry*, pages 16–25, 1992.

[Mul91] Ketan Mulmuley. Randomized multidimensional search trees: Further results in dynamic sampling. In *32nd Symp. Foundations of Computer Science*, pages 216–227, 1991.

[Sei91] Raimund Seidel. Low dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(5):423–434, 1991.

A Appendix: Finding the last simplex intersected by a query ray

Here we treat the problem arising in Section 3: Given a set of s simplices in $\mathbb{E}^d$ with disjoint interiors, preprocess it in polynomial time and space so that, given a query ray ρ, the last simplex intersected by ρ can be detected in $O(\log s)$ time.

Lemma 11 *Given a set Ξ of s simplices with disjoint interiors in $\mathbb{E}^d$. We can build in time $O(s^{8d+2})$ a data structure of size $O(s^{8d-6+\delta})$ that permits queries of the form "Given a query ray ρ, find the last simplex Δ in Ξ intersected by ρ" with query time $O(\log s)$.*

Proof: We apply a standard locus approach. We consider the space of all rays. The loci corresponding to rays with the same last simplex induce a subdivision of this space. We can do point location in this subdivision using the following result by Chazelle et al. [CEGS89].

Lemma 12 ([CEGS89]) *Let $f_1, \ldots, f_N$ be D-variate polynomials of degrees bounded by a constant (D is also considered a constant). For a D-dimensional variable $x := (x_1, x_2, \ldots, x_D)$, let $sign_i(x)$ be the sign of $f_i(x)$, and $sign(x) := (sign_1(x), \ldots, sign_N(x))$ be the sign vector of x with respect to the family (f_i). Assume we are given a label for every possible value of $sign(x)$.*

There is a data structure with storage and preprocessing time $O(N^{2D-3+\delta})$ which, given a D-dimensional vector x, finds the label associated with $sign(x)$ in time $O(\log N)$.

This result is applied as follows. We represent the query ray ρ by its origin p and by a direction vector u. We thus have a $D := 2d$ dimensional[5] representation for ρ. Consider now a fixed simplex Δ. We compute a constant number of polynomials in the $D = 2d$ variables representing ρ to test whether ρ intersects Δ. We first observe that ρ intersects Δ if either the origin p of ρ lies in Δ, or if some facet f of Δ is visible from p and the direction u lies in the convex cone with apex p spanned by f. The first case can easily be tested by $d+1$ linear inequalities (polynomials of degree one). For the second case we first identify the facets f visible from p using the same inequalities as in the first case.

We now have a facet f of Δ visible from p and wish to test whether u lies in the cone spanned by f. This can be done by checking the orientation of u with respect to the d hyperplanes that pass through p and the d ridges ($(d-2)$-faces) of f. For every ridge, we choose $d-1$ affinely independent points $q_1, q_2, \ldots, q_{d-1}$. The orientation of u with respect to the hyperplane spanned by p and $q_1, q_2, \ldots, q_{d-1}$ can be tested by computing the orientation of the simplex spanned by u and the $d-1$ vectors $q_1 - p$, $q_2 - p$, $\ldots$, $q_{d-1} - p$. This orientation can be found by computing the determinant of the matrix whose rows are these d vectors, which corresponds to the evaluation of a polynomial of degree d.

In this fashion we have determined a set of $O(s)$ polynomials. The signs of these polynomials for the variable representing a given ray ρ give us complete information as to which simplices are intersected by ρ, and, if a simplex Δ is intersected, through which facet f of Δ the ray ρ enters the simplex Δ. It remains to determine the *last* simplex intersected by ρ.

To this end, we add another set of $O(s^2)$ polynomials. Assume that ρ intersects two simplices Δ_1 and Δ_2, and enters them through their facets f_1 and f_2, resp. Let h_1 and h_2 be the hyperplanes containing f_1 and f_2. We can then test which of the two simplices is intersected first by testing which of the two intersections of ρ with h_1 and h_2 lies closer to p. But this can again be tested

[5] This is admittedly not optimal.

by determining the orientation of u with respect to the hyperplane through p and the $(d-2)$-flat $h_1 \cap h_2$. This can be dealt with as above. In this fashion we can add a polynomial for every pair of facets of different simplices.

The sign sequence of the resulting set of polynomials determines uniquely which simplex is the last one intersected by ρ. We can therefore appeal to Lemma 12, with parameters $N \in O(s^2)$ and $D = 2d$, to obtain a search structure with the time bounds claimed above. Given a ray ρ, this structure returns the label of the last simplex Δ intersected by ρ. $\square$