Linear Data Structures for Fast Ray-Shooting amidst Convex Polyhedra*

Haim Kaplan, Natan Rubin, and Micha Sharir

School of Computer Science, Tel Aviv University, Tel Aviv, Israel
{haimk,rubinnat,michas}@post.tau.ac.il

Abstract. We consider the problem of ray shooting in a three-dimensional scene consisting of k (possibly intersecting) convex polyhedra with a total of n facets. That is, we want to preprocess them into a data structure, so that the first intersection point of a query ray and the given polyhedra can be determined quickly. We describe data structures that require $\tilde{O}(n\text{poly}(k))$ preprocessing time and storage, and have polylogarithmic query time, for several special instances of the problem. These include the case when the ray origins are restricted to lie on a fixed line ℓ_0, but the directions of the rays are arbitrary, the more general case when the supporting lines of the rays pass through ℓ_0, and the case of rays orthogonal to z-axis with arbitrary origins. In all cases, this is a significant improvement over previously known techniques (which require $\Omega(n^2)$ storage, even when $k \ll n$).

1 Introduction

The general ray-shooting problem can be defined as follows: *Given a collection Γ of n objects in $\mathbb{R}^d$, preprocess Γ into a data structure so that one can quickly determine the first object in Γ intersected by a query ray.*

The ray-shooting problem has received much attention because of its applications in computer graphics and other geometric problems. Generally, there is a tradeoff between query time and storage (and preprocessing), where faster queries require more storage and preprocessing. In this paper, we focus on the case of fast (polylogarithmic) query time, and seek to minimize the storage of the data structures. For a more comprehensive review of the problem and its solutions, see [12]. If Γ consists of arbitrary triangles in $\mathbb{R}^3$, the best known data structure is due to Pellegrini [11]; it requires $O(n^{4+\epsilon})$ preprocessing and storage and has logarithmic query time. If Γ is the collection of facets on the boundary of a convex polyhedron, then an optimal algorithm, with $O(\log n)$ query and

* Work by Haim Kaplan and Natan Rubin has been supported by Grant 975/06 from the Israel Science Fund. Work by Micha Sharir and Natan Rubin was partially supported by NSF Grant CCF-05-14079, by a grant from the U.S.-Israeli Binational Science Foundation, by grant 155/05 from the Israel Science Fund, Israeli Academy of Sciences, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

$O(n)$ storage, can be obtained using the hierarchical decomposition of Dobkin and Kirkpatrick [9]. For the case where Γ consists of the boundary facets of k convex polyhedra, Agarwal and Sharir [4] describe a solution having polylogarithmic query time and $O(n^{2+\epsilon}k^2)$ storage, which is an improvement over [11]. Still, this solution requires $\Omega(n^2)$ storage, even for small k, in contrast to the $O(n)$ storage achieved in [9] for the case $k = 1$.

A wide range of special instances of the ray-shooting problem have been considered, where restrictions are imposed either on the query rays or on the input polyhedra. The most popular are the cases of C-oriented or fat input polyhedra, as well as the case of vertical ray-shooting (amidst arbitrary convex polyhedra). Another case, studied by Bern *et al.* [6], involves rays whose origins lie on a fixed straight line in $\mathbb{R}^3$. Unfortunately, all the known solutions to the restricted problems mentioned above require $\Omega(n^2)$ storage to achieve polylogarithmic query time, even in the case where the input consists of k convex polyhedra, where k is small relatively to n.

Our contribution. In this paper we focus on the case where the input consists of k convex polyhedra with a total of n facets, and implicitly assume that $k \ll n$. Our goal is to derive algorithms with polylogarithmic query time, for which the storage that they require is (nearly) linear in n. We achieve this in several useful special cases, where one case involves ray-shooting with rays whose origins lie on a fixed line. In this case our solution uses $\tilde{O}(nk^5)$ storage and preprocessing, and answers queries in polylogarithmic time. (The notation $\tilde{O}(\cdot)$ hides polylogarithmic factors.) We extend this solution for the case where the rays lie on lines that pass through a fixed line, but their origins are arbitrary, and for the case of rays orthogonal to the z-axis with arbitrary origins. This is achieved at the cost of increasing the preprocessing time to $\tilde{O}(nk^6)$, with similar time and storage complexity. The problems that we study have the common property that the lines containing the query rays have only three degrees of freedom, as opposed to the general case when the lines have four degrees of freedom. If we ignore the issue of making the performance of the algorithm depend on k, there are solutions that require $O(n^{3+\epsilon})$ storage and preprocessing, with polylogarithmic query time (e.g., a simple adaptation of the technique of [3]). No previous algorithm solves these problems with performance that depends subquadratically on n, when $k \ll n$.

Overview. We apply the following uniform approach to both problem instances. Denote by $\mathcal{P}$ the set of input polyhedra. We define a parametric space $\mathbb{S}$ whose points represent lines containing the query rays. Since the lines containing the query rays have three degrees of freedom, we take $\mathbb{S}$ to be $\mathbb{R}^3$. Each polyhedron $P \in \mathcal{P}$ defines a surface σ_P in $\mathbb{S}$, which is the locus of all (points representing) lines that are tangent to P. The arrangement $\mathcal{C}(\mathcal{P})$ of the surfaces σ_P yields a subdivision of $\mathbb{S}$, each of whose cells C is a maximal connected relatively open set of lines which stab the same subset $\mathcal{P}_C$ of $\mathcal{P}$. When the polyhedra of $\mathcal{P}$ are pairwise disjoint, the order in which lines ℓ in any fixed cell C intersect the polyhedra of $\mathcal{P}_C$ is the same for all $\ell \in C$. Let r be a query ray, let ℓ_r

be the line containing r, and let $C \subset \mathbb{S}$ denote the cell of $\mathcal{C}(\mathcal{P})$ containing the point representing ℓ_r. We can answer a query by a binary search with the ray origin among the sorted list of polyhedra of $\mathcal{P}_C$. In Section 2, we modify the analysis of Brönnimann *et al.* [7] to prove that the complexity of $\mathcal{C}(\mathcal{P})$, for lines ℓ_r passing through a fixed line, is $O(nk^2)$. We use the *persistent* data structure technique of [13] to search for the cell of $\mathcal{C}(\mathcal{P})$ containing ℓ_r. This results in an algorithm for ray-shooting from a line into a collection of pairwise disjoint polyhedra, which requires $\tilde{O}(nk^2)$ storage and preprocessing time, and supports queries in polylogarithmic time.

The above approach fails when the polyhedra of $\mathcal{P}$ may intersect each other, because the order in which the polyhedra of $\mathcal{P}_C$ are intersected by ℓ_r need not be fixed over $\ell_r \in C$. To handle this difficulty, we apply a more sophisticated technique. We add to $\mathcal{P}$ all *pairwise intersections* between the polyhedra of $\mathcal{P}$, and obtain a set $\mathcal{Q}$, consisting of $O(k^2)$ polyhedra with a total of $O(nk)$ facets. Instead of $\mathcal{C}(\mathcal{P})$ we now consider the arrangement $\mathcal{C}(\mathcal{Q})$, which is a refinement of $\mathcal{C}(\mathcal{P})$. Each cell C in $\mathcal{C}(\mathcal{Q})$ is a maximal connected region of $\mathbb{S}$ such that all lines represented in C stab the same subset $\mathcal{Q}_C$ of $\mathcal{Q}$. We can use essentially the same spatial data structure as in the case of disjoint polyhedra, constructed over $\mathcal{C}(\mathcal{Q})$ instead of over $\mathcal{C}(\mathcal{P})$, to locate the cell of $\mathcal{C}(\mathcal{Q})$ containing a query line ℓ_r. Now the structure requires $\tilde{O}(nk \cdot (k^2)^2) = \tilde{O}(nk^5)$ storage and preprocessing, and supports queries in polylogarithmic time; see Section 3.

Let P and Q be any pair of polyhedra in $\mathcal{P}_C$. If $P \cap Q$ does not belong to $\mathcal{Q}_C$, that is, $P \cap Q$ is not intersected by lines represented in C, then P and Q are intersected in the same order by all lines ℓ represented in C. This induces a *partial order* $\prec_C$ on the polyhedra of $\mathcal{P}$. Clearly, $\Xi \subset \mathcal{P}_C$ is an *antichain* with respect to $\prec_C$ if and only if, for all P and Q in Ξ, the polyhedron $P \cap Q$ is in $\mathcal{Q}_C$. The one-dimensional Helly theorem then implies that the polyhedron $\bigcap \Xi$ (the intersection of all of the polyhedra in Ξ) (a) is nonempty, and (b) is intersected by all lines represented in C.

Using *parametric search*, the ray-shooting problem reduces to testing a query segment for intersection with $\mathcal{P}$; see also [2]. The latter problem can be solved, for segments contained in lines represented in C, using a small number of ray-shooting queries amidst polyhedra of the form $\bigcap \Xi$, where Ξ is a *maximal antichain* with respect to $\prec_C$. Organizing the data to facilitate efficient implementation of these ray shootings requires several additional technical steps, which are detailed in the relevant sections below.

The paper is organized as follows. In Section 2 we describe the solution for ray-shooting *from a fixed line* ℓ_0 amidst *disjoint polyhedra*. In Section 3 we describe the extra machinery used to handle *intersecting* polyhedra. In the full version [10], we generalize our approach to solve the ray-shooting problem for rays with *arbitrary origins* that contained in lines which intersect ℓ_0, and for rays orthogonal to the z-axis. For lack of space some proofs are omitted from this extended abstract.

Preliminaries. Let $\mathcal{P}$ be a set of k polyhedra in $\mathbb{R}^3$. A *support vertex* of a line ℓ is a vertex v of a polyhedron $P \in \mathcal{P}$, so that v lies on ℓ (and ℓ does not meet

the interior of P). A *support edge* of a line ℓ is an edge e of a polyhedron $P \in \mathcal{P}$, so that e intersects ℓ at its relative interior (and ℓ does not meet the interior of P). A *support polyhedron* of a line is a polyhedron that contains a support edge or vertex of the line.

Let S be a set of edges and vertices. A line ℓ is a *transversal of* S if ℓ intersect every edge or vertex in S. Line ℓ is an *isolated* transversal of S if it is a transversal of S and it cannot be moved continuously while remaining a transversal of S. A set S of edges and vertices *admits an isolated transversal* if there is an isolated transversal ℓ of S.

Brönnimann *et al.* [7] prove that there are $O(n^2k^2)$ minimal sets of relatively open edges and vertices, chosen from some of the polyhedra, which admit an isolated transversal that is tangent to these support polyhedra. This bound is an easy consequence of the following main lemma of [7].

Theorem 1. *Let P, Q and R be three convex polyhedra in $\mathbb{R}^3$, having p, q and r facets, respectively, and let ℓ_0 be an arbitrary line. There are $O(p+q+r)$ minimal sets of relatively open edges and vertices, chosen from some of these three polyhedra, which, together with ℓ_0, admit an isolated transversal that is tangent to these support polyhedra (excluding sets that lie in planes which contain ℓ_0 and are tangent to all three polyhedra, if such planes exist).*

Clearly, the number of such transversals, over all triples of polyhedra, is $O(nk^2)$. They can be computed in $O(nk^2 \log n)$ time, as described in [7].

2 Ray-Shooting from a Line: Disjoint Polyhedra

In this section we present an algorithm for the case where we shoot from a fixed line ℓ_0, and the input consists of pairwise-disjoint polyhedra. Our approach somewhat resembles that of [7]. We parameterize the set of planes containing ℓ_0 by fixing one such plane Π_0 in an arbitrary manner, and by defining Π_t, for $0 \leq t < \pi$, to be the plane obtained by rotating Π_0 around ℓ_0 by angle t.

We represent a line ℓ passing through ℓ_0 by the pair $(t(\ell), D_t(\ell))$, where $t(\ell)$ is such that $\Pi_{t(\ell)}$ contains ℓ, and $D_t(\ell)$ is the point dual to ℓ in $\Pi_{t(\ell)}$. For convenience, we choose $D_t(\cdot)$ to be a planar duality transform which excludes points corresponding to lines parallel to ℓ_0. That is, if we choose in Π_t a coordinate frame in which ℓ_0 is the y-axis and the origin is a fixed point on ℓ_0, then the duality maps each point (ξ, η) to the line $y = \xi x - \eta$ and vice versa. As above, we denote by $\mathbb{S}$ the resulting 3-dimensional parametric space of lines.

We use the arrangement $\mathcal{C}(\mathcal{P})$ defined in the introduction, and recall that each cell C of $\mathcal{C}(\mathcal{P})$ is a maximal connected region in $\mathbb{S}$, such that all lines in C stab the same subset of $\mathcal{P}$. For a cell $C \in \mathcal{C}(\mathcal{P})$, we denote by $\mathcal{P}_C \subseteq \mathcal{P}$ the set of the polyhedra stabbed by the lines in C. Clearly, the polyhedra of $\mathcal{P}_C$ are intersected in the same order by all lines in C (this follows by a simple continuity argument, using the fact that C is connected). This allows us to reduce ray-shooting queries with rays emanating from ℓ_0 to point location queries in $\mathcal{C}(\mathcal{P})$, in the manner explained in the introduction.

Without loss of generality we assume that the interiors of all the polyhedra in $\mathcal{P}$ are disjoint from ℓ_0. Otherwise, we can cut each of the polyhedra, whose interior is intersected by ℓ_0, into two sub-polyhedra, by some plane through ℓ_0. This will not affect the asymptotic complexity of the solution.

This allows us, for each cell C, to order the polyhedra in $\mathcal{P}_C$, and the line ℓ_0, according to their intersections with a line $\ell \in C$, where the order is independent of ℓ. In fact, it suffices to store, for each $C \in \mathcal{C}(\mathcal{P})$, the two polyhedra of $\mathcal{P}_C$ that are adjacent to ℓ_0 in this order. We denote these polyhedra by P_C^+ and P_C^-.

Next we describe how to construct the point location data structure over $\mathcal{C}(\mathcal{P})$, and how to compute P_C^+ and P_C^-, as defined above, for each $C \in \mathcal{C}(\mathcal{P})$.

Point location in $\mathcal{C}(\mathcal{P})$. Consider a plane Π_t, for some fixed value of the rotation angle t. For each polyhedron $P \in \mathcal{P}$, let P_t denote the intersection polygon $P \cap \Pi_t$, and let $\mathcal{P}_t = \{P_t \mid P \in \mathcal{P},\ P_t \neq \emptyset\}$ be the set of all resulting (non-empty) polygons. For each polygon $P_t \in \mathcal{P}_t$ we denote by $\mathcal{U}(P_t)$ (resp., $\mathcal{L}(P_t)$) the upper (resp., lower) envelope of all the lines dual to the vertices of P_t in the plane Π_t. The following observation is elementary.

Observation: *Let ℓ be a line contained in Π_t, and let $D_t(\ell)$ be the point dual to ℓ in the dual plane. Then ℓ does not intersect P if and only if $D_t(\ell)$ lies above $\mathcal{U}(P_t)$ or below $\mathcal{L}(P_t)$. Points on $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$, are dual to lines tangent to P_t (and, therefore, also to P).* (See Figure 1 for an illustration.)

We denote by $\mathcal{A}_t^*$ the arrangement of the monotone piecewise-linear unbounded curves $\{\mathcal{L}(P_t), \mathcal{U}(P_t)\}_{P_t \in \mathcal{P}_t}$. Clearly, regions in $\mathcal{A}_t^*$ correspond to maximal connected sets of lines passing through ℓ_0, such that all lines in the same region stab the same subset of $\mathcal{P}$.[1]

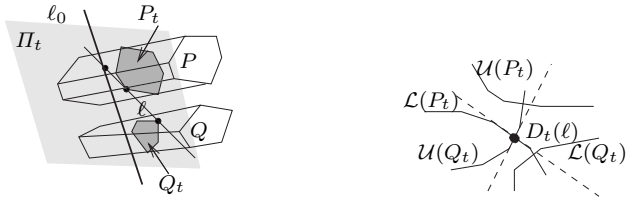

Fig. 1. ℓ is a common tangent of P_t and Q_t in the primal plane Π_t (left). The dual arrangement $\mathcal{A}_t^*$ (right). $D_t(\ell)$ is an intersection of two lines, which are dual to the support vertices of ℓ in Π_t.

Hence, for each cell $C \in \mathcal{C}(\mathcal{P})$, the intersection of C with the surface $\{t(\ell) = t\}$ consists of one or several connected cells in $\mathcal{A}_t^*$. Thus, answering a point-location query in $\mathcal{C}(\mathcal{P})$, for a query point $(t(\ell), D_t(\ell))$, is reduced to answering a point-location query in $\mathcal{A}_{t(\ell)}^*$, with the point $D_t(\ell)$.

To perform point location queries in $\mathcal{A}_t^*$, for all $t \in [0, \pi]$, we employ an adapted version of the technique of Preparata and Tamassia [13] for point loca-

[1] We regard the polyhedra $P \in \mathcal{P}$ as closed, so tangency of a line to some P also counts as intersection. This requires some (routine) care in handling lower-dimensional faces of $\mathcal{A}_t^*$, which we omit.

tion in a dynamically changing monotone planar subdivision. There are certain events, of two different kinds, at which $\mathcal{A}_t^*$ changes combinatorially. Events of the first kind occur when the sweep plane Π_t passes through a vertex v of some $P \in \mathcal{P}$. It can easily be argued that all such events cause $O(nk)$ topology changes to $\mathcal{A}_t^*$ in total (at each of there events, we either get $O(k)$ new vertices of $\mathcal{A}_t^*$, or have to delete $O(k)$ vertices from $\mathcal{A}_t^*$). In events of the second kind, each of the envelopes in $\{\mathcal{L}(P_t),\mathcal{U}(P_t)\}_{P_t \in \mathcal{P}_t}$ is combinatorially unchanged. Events of this kind either involve a concurrent triple of envelopes (where the intersection point is dual to a line tangent to three polygons of $\mathcal{P}_t$), or involve a vertex of one envelope lying on another envelope (which is dual to a line supporting a vertex of one P_t and tangent to another Q_t). That is, each of the listed events corresponds to a set of polyhedra edges and vertices which (together with ℓ_0) admit an isolated transversal. Hence, by Theorem 1, the number of such events is $O(nk^2)$. In summary, we have:

Theorem 2. *Let $\mathcal{P}$ be a set of k convex polyhedra with a total of n facets, let ℓ_0 be a fixed line, and let $\mathcal{C}(\mathcal{P})$ be the corresponding cell decomposition of the parametric space $\mathbb{S}$, as defined above. We can construct, in $O(nk^2 \log^2 n)$ time, a data structure which supports point location queries in $\mathcal{C}(\mathcal{P})$ in $O(\log^2 n)$ time, and requires $O(nk^2 \log^2 n)$ storage.*

Note that this theorem does not require the input polyhedra to be disjoint. We will apply it also in Section 3, in contexts involving intersecting polyhedra.

To compute, for each cell $C \in \mathcal{C}(\mathcal{P})$, the pair of polyhedra P_C^+ and P_C^- "nearest" to ℓ_0, we note that the enumeration of the elements of $\mathcal{P}_C \cup \{\ell_0\}$, ordered according to their intersections with lines $\ell \in C$, changes in at most one position as we pass between neighboring cells of $\mathcal{C}(\mathcal{P})$ (across a common 2-face), where in each such change we have to either insert a new polyhedron P into $\mathcal{P}_C$, or remove a polyhedron from this list. So we maintain this list in a persistent search tree, as follows. After $\mathcal{C}(\mathcal{P})$ has been constructed, we construct a spanning tree of the adjacency graph of its cells, and turn it into a linear-size sequence of cells, each consecutive pair of which are adjacent. We then traverse the sequence, and when we pass from a cell C to the next (adjacent) cell C', we update $\mathcal{P}_C$ in a persistent manner, and construct from it $\mathcal{P}_{C'}$, by the corresponding insertion or deletion of a polyhedron.

Theorem 3. *Let $\mathcal{P}$ be a set of k pairwise disjoint convex polyhedra in $\mathbb{R}^3$ with a total of n facets, and let ℓ_0 be a fixed line. Then we can construct, in $O(nk^2 \log^2 n)$ time, a data structure of size $O(nk^2 \log^2 n)$, which supports ray-shooting queries from ℓ_0 in $O(\log^2 n)$ time.*

3 Ray-Shooting from a Line: Handling Intersecting Polyhedra

The solution described in the previous section fails for the case of intersecting polyhedra because lines which belong to the same cell C may intersect the

boundaries of the polyhedra of $\mathcal{P}_C$ in different orders. In this section we consider this case, and derive a solution which is less efficient, but still nearly linear in n.

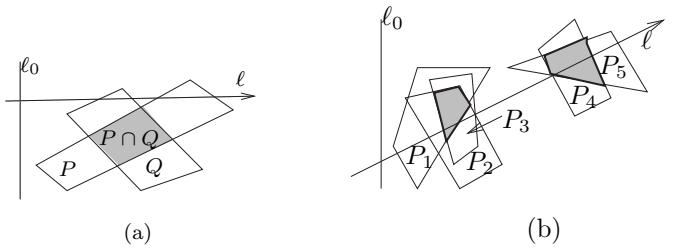

Fig. 2. (a) $P \prec_C Q$, for the cell C containing ℓ; the polyhedron $P \cap Q$ is not in $\mathcal{Q}_C$. (b) Illustrating $\overline{Front}(C)$, for the cell C containing ℓ. We have $Front(C) = \{P_1, P_2\}$ and $\overline{Front}(C) = \{P_1, P_2, P_3\}$. The maximal antichains are $\{P_1, P_2, P_3\}$ and $\{P_4, P_5\}$.

As described in Section 1, we use parametric search to reduce the problem to segment intersection detection. Specifically, the problem is reduced to testing a segment $s = o_1o_2$, such that o_1 is on ℓ_0, for intersection with the polyhedra of $\mathcal{P}$. We begin with the following trivial observation.

Observation: *Let $\mathcal{P}$ be a set of convex polyhedra, let $\mathcal{A}$ denote the arrangement of their boundaries, and let s be a segment. If the endpoints of s belong to* distinct cells *of $\mathcal{A}$ then s intersects the boundary of a polyhedron of $\mathcal{P}$.*

The first ingredient of our algorithm is thus a data structure for point location in $\mathcal{A}$. For example, we can use the structure described in [13]. For this, we note that $\mathcal{A}$ is not *xy-monotone*, as required in [13]. Nevertheless, we can replace each polyhedron $P \in \mathcal{P}$ by two semi-unbounded polyhedra P^+, P^-, where P^+ (resp., P^-) consists of all the points that lie in P or above (resp., below) it. We obtain $2k$ convex polyhedra, still with a total of $O(n)$ facets, whose arrangement is xy-monotone, and is in fact a refinement of $\mathcal{A}$. Thus, locating a point in the new arrangement, using the algorithm in [13], gives us the cell of $\mathcal{A}$ that contains it. The complexity of the (extended) $\mathcal{A}$ is $O(nk^2)$—an easy and well known fact. Hence, the resulting data structure requires $O(nk^2 \log^2 n)$ storage and preprocessing, and answers point-location queries in $O(\log^2 n)$ time.

The more difficult part is to test whether s intersects a polyhedron of $\mathcal{P}$ when the endpoints of s belong to the same cell of $\mathcal{A}$. In this case, it is sufficient to test s for intersection with the polyhedra of $\mathcal{P}$ which do not contain o_1. Indeed, if s intersects a polyhedron P containing o_1 then the other endpoint o_2 is outside of P, and, therefore, o_1 and o_2 belong to different cells of $\mathcal{A}$.

As in the previous section, we cut each of the polyhedra, whose interior is intersected by ℓ_0, into two sub-polyhedra, by some plane through ℓ_0, and leave rest of the polyhedra unchanged. Note, that we have just added artificial facets to the polyhedra of $\mathcal{P}$, whose affine extensions contain ℓ_0.

We replace $\mathcal{P}$ with the set $\mathcal{Q} = \{P \cap Q \mid P, Q \in \mathcal{P}\}$; note that the original polyhedra of $\mathcal{P}$ are also in $\mathcal{Q}$. The new set contains $O(k^2)$ convex polyhedra of

total complexity $O\left(\sum_{P,Q\in\mathcal{P}}(n_P+n_Q)\right) = O(nk)$, where n_P is the number of facets of polyhedron P. As above, let $\mathcal{C}(\mathcal{Q})$ denote the 3-dimensional arrangement in $\mathbb{S}$ defined by the surfaces of tangents to the polyhedra in $\mathcal{Q}$.

We construct a point location structure over $\mathcal{C}(\mathcal{Q})$ as we did for $\mathcal{C}(\mathcal{P})$ in Section 2. Each cell $C \in \mathcal{C}(\mathcal{Q})$ is a maximal connected region with the property that all lines in C stab the same subset of polyhedra of $\mathcal{Q}$, which we denote by $\mathcal{Q}_C$. This point location data structure supports queries in $O(\log^2(nk)) = O(\log^2 n)$ time, and requires $O(nk \cdot (k^2)^2 \log^2(nk)) = O(nk^5 \log^2 n)$ storage and preprocessing time. The following is an obvious but crucial observation, which justifies the introduction of $\mathcal{Q}$.

Observation: *Let C be a cell of $\mathcal{C}(\mathcal{Q})$, and let ℓ be a line in C. For any pair of distinct polyhedra $P, Q \in \mathcal{P}_C$, the segments $P \cap \ell$ and $Q \cap \ell$ intersect if and only if $P \cap Q$ is in $\mathcal{Q}_C$.*

We consider a query ray in a line ℓ to be *positive* if it is contained in the halfplane of $\Pi_{t(\ell)}$ which lies to the right of ℓ_0 (in an appropriate coordinate frame, as described above). We focus below on the case of positive rays, since negative rays can be handled in a fully symmetric manner. We denote by ℓ^+ the positive ray (emanating from ℓ_0) contained in the line ℓ, and, for a cell $C \in \mathcal{C}(\mathcal{Q})$, we denote by $\mathcal{P}_C^+ \subseteq \mathcal{P}$ the subset of polyhedra of $\mathcal{P}$ intersected by ℓ^+, for any line ℓ in C; since we have assumed that no polyhedron of $\mathcal{P}$ intersects ℓ_0, this property is indeed independent of the choice of ℓ in C.

We define $Front(C)$ to be the set of all polyhedra $P \in \mathcal{P}_C^+$ for which there exists a line ℓ in C, such that an endpoint of $P \cap \ell$ is the closest to ℓ_0, among all the boundary points, along ℓ. That is, $Front(C)$ consists of all candidate polyhedra to be the first intersected by positive rays ℓ^+, for $\ell \in C$.

We define a partial order $\prec_C$ on the polyhedra of $\mathcal{P}_C$, as follows. For $P, Q \in \mathcal{P}_C$, we say that $P \prec_C Q$ if $P \cap Q \notin \mathcal{Q}_C$ (the polyhedron $P \cap Q$ is not stabbed by any line in C), and the segment $P \cap \ell$ appears before $Q \cap \ell$ along ℓ, for any line ℓ in C. See Figure 2 (a) for an illustration.

Lemma 1. *The order $P \prec_C Q$ (a) is well defined and is independent of the choice of ℓ in C, and (b) is indeed a partial order.*

Recall that a subset $\mathcal{T} \subseteq \mathcal{P}_C$ is called an *antichain* of $\prec_C$ if any two distinct polyhedra $P, Q \in \mathcal{T}$ are unrelated under $\prec_C$; $\mathcal{T}$ is called a *maximal antichain* if no (proper) superset of $\mathcal{T}$ is an antichain.

Lemma 2. *A subset $\mathcal{T} \subseteq \mathcal{P}_C$ is an antichain under $\prec_C$ if and only if every line ℓ in C intersects the polyhedron $\bigcap \mathcal{T} = \bigcap_{P\in\mathcal{T}} P$.*

Proof. Let $\mathcal{T}$ be an antichain under $\prec_C$. For any $P, Q \in \mathcal{T}$, the polyhedra P and Q are unrelated under $\prec_C$, so $P \cap Q \in \mathcal{Q}_C$. Therefore, for every line ℓ in C, the segments $P \cap \ell$ and $Q \cap \ell$ intersect. Hence, by the one-dimensional Helly theorem, $\bigcap_{P\in\mathcal{T}}(P \cap \ell) = (\bigcap \mathcal{T}) \cap \ell$ is not empty, for every $\ell \in C$.

To prove the converse statement, let $\mathcal{T}$ be a subset of $\mathcal{P}_C$ such that $\bigcap \mathcal{T}$ is stabbed by every line in C. In particular, for any pair of polyhedra P and Q

in $\mathcal{T}$, the polyhedron $P \cap Q$ is stabbed by every line in C. Thus, P and Q are unrelated under $\prec_C$. □

We define $\overline{Front}(C) \subseteq \mathcal{P}_C$ to be the set of all minimal polyhedra under $\prec_C$. It is easy to see that $Front(C) \subseteq \overline{Front}(C)$. (A proper inclusion is possible—see Figure 2 (b).) Since $\overline{Front}(C)$ is a maximal antichain under $\prec_C$, we obtain the following corollary. See Figure 2(b) for an illustration.

Corollary 1. *Every line in C intersects the polyhedron $\bigcap \overline{Front}(C)$.*

For each cell C we store a pointer to a data structure which supports ray-shooting queries at the polyhedron $\bigcap \overline{Front}(C)$ with positive rays contained in lines $\ell \in C$. Details about the structure, and the analysis of its storage and construction cost, will be presented later in this section.

Answering Queries. The query algorithm is based on the following lemma. See Figure 3(a) for an illustration.

Lemma 3. *Let $s = o_1o_2$ be a query segment (where $o_1 \in \ell_0$), contained in a positive ray. Let ℓ be the line containing s, and let C be the cell of $\mathcal{C}(\mathcal{Q})$ that contains ℓ. Then s intersects a (non-artificial) facet on the boundary of some polyhedron of $\mathcal{P}$ if and only if one of the following two conditions holds:*
(i) o_1 and o_2 belong to distinct cells of $\mathcal{A}$.
(ii) s intersects the boundary of the polyhedron $\bigcap \overline{Front}(C)$ and o_2 is not contained in $\bigcap \overline{Front}(C)$.

Proof. If the first one of the two conditions holds then s clearly intersects the boundary of some polyhedron of $\mathcal{P}$. Now assume s intersects the boundary of $\bigcap \overline{Front}(C)$ and o_2 is not contained in $\overline{Front}(C)$. In particular, there is $P \in \overline{Front}(C)$ such that o_2 is not contained in P, and s intersects P. Hence, s intersects a non-artificial facet on the boundary of P.

For the converse statement, assume that s intersects a non-artificial facet on the boundary of some polyhedron P of $\mathcal{P}$. If o_2 is contained in P then, by assumption, o_1 and o_2 belong to distinct cells of $\mathcal{A}$. Otherwise, let P be the first polyhedron intersected by s, and let ℓ^+ denote the ray which contains s. Since o_2 is not contained in P, the segment $P \cap \ell^+$ is contained in s. By definition, P belongs to $\overline{Front}(C)$. By Corollary 1, ℓ^+ intersects $\bigcap \overline{Front}(C)$. Since $\overline{Front}(C) \subset P$, the intersection of ℓ^+ with $\bigcap \overline{Front}(C)$ is also contained in s. Hence, o_2 is not contained in $\bigcap \overline{Front}(C)$, and s intersects $\bigcap \overline{Front}(C)$. □

Now we are ready to describe our query algorithm. First, we use the data structure for point location in $\mathcal{A}$ to test whether the endpoints of s belong to different cells of $\mathcal{A}$. If this is the case, we report intersection and finish. Otherwise, we query the spatial point location structure of $\mathcal{C}(\mathcal{Q})$ to find the cell C which contains the line ℓ containing s. Then, we test whether s intersects $\bigcap \overline{Front}(C)$, and whether o_2 is not contained in $\overline{Front}(C)$, and report intersection if and only if the answer is positive. The correctness of this procedure follows from the preceding analysis.

Query Time and Storage. For any collection of polyhedra $\mathcal{P}' \subseteq \mathcal{P}$, and a face f of $\mathcal{A}$, we say that f is *good* for $\mathcal{P}'$ if $\mathcal{P}'$ is the set of all (closed) polyhedra in $\mathcal{P}$ that contain f.

Lemma 4. *Let C be a cell of $\mathcal{C}(\mathcal{Q})$. If a line ℓ in C intersects $f \in \mathcal{A}$, and f is on $\partial(\bigcap \overline{Front}(C))$, then f is good for $\overline{Front}(C)$.*

Proof. We have to prove that the set of polyhedra that contain f is equal to $\overline{Front}(C)$. Since f is on $\partial(\bigcap \overline{Front}(C))$, it is clear that every polyhedron in $\overline{Front}(C)$ contains f. Conversely, let P be a polyhedron that contains f. We claim that P is minimal in $\mathcal{P}_C^+$ under $\prec_C$, and therefore $P \in \overline{Front}(C)$. Indeed, if P is not minimal under $\prec_C$, then there exists $Q \in \overline{Front}(C)$ such that $Q \prec_C P$. It follows that $Q \cap \ell$ is disjoint from $P \cap \ell$, but this contradicts the fact that ℓ intersects f, which clearly lies in $P \cap Q$. □

Therefore, it suffices to solve, for each cell C, the ray-shooting problem only amidst the faces of $\mathcal{A}$ that lie on the boundary of $\bigcap \overline{Front}(C)$ and are good for $\overline{Front}(C)$. See Figure 3(b) for an illustration.

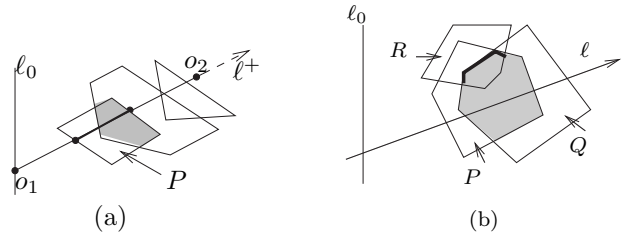

Fig. 3. (a) Query algorithm: case (ii). o_1 and o_2 lie in the same cell of A, s intersects $\bigcap \overline{Front}(C)$ and o_2 lies out of $\bigcap \overline{Front}(C)$. (b) Good facets. For the cell C which contains ℓ, $\bigcap \overline{Front}(C) = P \cap Q$. The facets of $P \cap Q$ which are contained in R (bold) are not good and are not intersected by lines in C.

Let $\mathcal{P}' \subseteq \mathcal{P}$ be a set of polyhedra. Let f be a good facet for $\mathcal{P}'$, let h_f be the plane containing f, and let h_f^+ be the halfspace bounded by h_f and containing $\bigcap \mathcal{P}'$. We define $G(\mathcal{P}')$ to be the polyhedron obtained by intersecting the halfspaces h_f^+, over all good faces f for $\mathcal{P}'$. Note that $\bigcap P' \subseteq G(P')$, and that every good face for P' lies on $\partial G(P')$. In particular, we have the following property (brought without proof).

Lemma 5. *Let C be a cell in $\mathcal{C}(\mathcal{P})$, let ℓ be a line in C, and let ℓ^+ be the positive ray of ℓ, as defined above. Then $\bigcap \overline{Front}(C)$ and $G(\overline{Front}(C))$ have the same intersection with ℓ^+.*

Hence, s intersects $\bigcap \overline{Front}(C)$, and o_2 is not in $\overline{Front}(C)$ if and only if it intersects $G(\overline{Front}(C))$, and o_2 is not in $G(\overline{Front}(C))$. It suffices to test s for the latter property.

For each cell $C \in \mathcal{C}(\mathcal{Q})$, we construct a data structure which supports ray-shooting queries at $G(\overline{Front}(C))$. If implemented as in [9], the structure uses storage linear in the complexity of $G(\overline{Front}(C))$, and can answer queries in $O(\log n)$ time. With each cell $C \in \mathcal{C}(\mathcal{Q})$ we store a pointer to the ray-shooting structure at $G(\overline{Front}(C))$. As each face of $\mathcal{A}$ is good for exactly one subset $\mathcal{P}'$ of polyhedra, the total storage and preprocessing required for all these auxiliary ray-shooting data-structures is $O(nk^2 \log n)$.

Hence, the resulting data structure uses $O(nk^5 \log^2 n)$ storage (the major part of which is consumed by the point location structure in the decomposition $\mathcal{C}(\mathcal{Q})$). Since we use parametric search, the actual query time is quadratic in the cost of a segment intersection query, and is thus $O(\log^4 n)$.

Preprocessing. The only nontrivial part of the preprocessing is the construction of the ray-shooting structures at the polyhedra $G(\overline{Front}(C))$, for all the cells $C \in \mathcal{C}(\mathcal{Q})$. First, for each face f of $\mathcal{A}$, we compute the unique subset $\mathcal{P}_f \subseteq \mathcal{P}$ for which f is good, which is done by testing, for each polyhedron $P \in \mathcal{P}$, whether P contains some fixed point $p_f \in f$. Since the complexity of $\mathcal{A}$ is $O(nk^2)$, this can be done in total time $O(nk^3 \log n)$. We represent each subset $\mathcal{P}_f$ by a bitvector in $\{0,1\}^k$. Next, we collect the faces f having the same set $\mathcal{P}_f$. This can be done by sorting the $O(nk^2)$ k-long bitvectors $\mathcal{P}_f$, in time $O(nk^3 \log n)$. We construct for each of the resulting equivalence classes $\mathcal{T}$ a ray-shooting structure on the polyhedron $G(\mathcal{T})$.

To supply each cell $C \in \mathcal{C}(\mathcal{Q})$ with a pointer to the ray-shooting structure corresponding to $G(\overline{Front}(C))$, we proceed as follows. Recall that $\mathcal{C}(\mathcal{Q})$ is a refinement of $\mathcal{C}(\mathcal{P})$. For a cell $\tilde{C} \in \mathcal{C}(\mathcal{P})$, the cells $C \in \mathcal{C}(\mathcal{Q})$ contained in $\tilde{C}$ form a connected portion of the adjacency structure induced by the arrangement $\mathcal{C}(\mathcal{Q})$. Let C and C' be a pair of adjacent cells in $\mathcal{C}(\mathcal{Q})$ which are contained in the same cell of $\mathcal{C}(\mathcal{P})$. If the set $\mathcal{P}$ of original polyhedra is in general position, there is a unique pair of polyhedra P and Q such that $\{P \cap Q\} = \mathcal{Q}_C \triangle \mathcal{Q}_{C'}$. Therefore, (P, Q) is the only pair of polyhedra related under $\prec_C$ and unrelated under $\prec_{C'}$ (or vice versa). Thus in C we have, say, $P \prec_C Q$, so Q is not in $\overline{Front}(C)$. Since all other pairs do not change their status in $\prec$, the only possible change from $\overline{Front}(C)$ to $\overline{Front}(C')$ is that Q may be added to the latter set, which is the case if and only if the following property holds. Let ℓ be a line on the common boundary between C and C', and let q be the singleton point in $P \cap Q \cap \ell$. Then Q is added to $Front(C')$ if and only if $q \in \bigcap \overline{Front}(C)$; In other words, we have argued that $|\overline{Front}(C) \triangle \overline{Front}(C')| \leq 1$.

Let $\tilde{C}$ be a cell in $\mathcal{C}(\mathcal{P})$. We pick a representative cell $C_0 \in \mathcal{C}(\mathcal{Q})$ contained in $\tilde{C}$, and compute $\overline{Front}(C_0)$ by brute force, as described above. Next we trace the adjacency structure of $\mathcal{C}(\mathcal{Q})$ within $\tilde{C}$ in breadth-first fashion, and find all the cells $C \in \mathcal{C}(\mathcal{Q})$ contained in $\tilde{C}$. For each newly encountered cell C', reached via a previous cell C, we already have, by construction, the pair $P, Q \in \mathcal{P}$ such that $\{P \cap Q\} = \mathcal{Q}_C \triangle \mathcal{Q}_{C'}$. We can now compute $\overline{Front}(C')$ from $\overline{Front}(C)$ as follows. We take a line ℓ on the common boundary of C and C', compute $P \cap \ell$ and $Q \cap \ell$, obtain their common endpoint q, verify that, say, $P \cap \ell$ precedes $Q \cap \ell$ along ℓ, and test whether $q \in \bigcap \overline{Front}(C')$. If so, $\overline{Front}(C')$ is obtained

by either adding or removing Q from $\overline{Front}(C)$ (we know which action to take); otherwise $\overline{Front}(C) = \overline{Front}(C')$.

To make the transition from $\overline{Front}(C)$ to $\overline{Front}(C')$ efficient, we precompute a table on the equivalence classes $\mathcal{T}$ that were constructed at the preliminary stage. For each class $\mathcal{T}$ and each polyhedron P we store a pointer to $\mathcal{T}' = \mathcal{T} \triangle \{P\}$.

To bound the total preprocessing time, we note that the brute force method for finding $\overline{Front}(C)$ takes $O(k \log n)$ time and is applied to $O(nk^2)$ representatives of cells in $\mathcal{C}(\mathcal{P})$. Also observe that in our traversal of $\mathcal{C}(\mathcal{Q})$, over all cells $\tilde{C} \in \mathcal{C}(\mathcal{P})$, we make a total of $O(nk^5)$ transitions, by the upper bound on the overall complexity of $\mathcal{A}(\mathcal{Q})$. Each such transition takes $O(\log n)$ time. In summary, we have:

Theorem 4. *Let $\mathcal{P}$ be a set of k possibly intersecting convex polyhedra with a total of n facets, and a let ℓ_0 be a fixed line in $\mathbb{R}^3$. Then we can construct a data structure supporting ray-shooting queries with rays emanating from ℓ_0, in $O(\log^4 n)$ time. The structure uses $O(nk^5 \log^2 n)$ storage and preprocessing.*

References

1. B. Aronov, M. de Berg and C. Gray, Ray shooting and intersection searching amidst fat convex polyhedra in 3-space, *Proc. 22nd Annu. ACM Sympos. Comput. Geom.* (2006), 88–94.
2. P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* 22 (1993), 794–806.
3. P. K. Agarwal and J. Matoušek, Range searching with semialgebraic sets, *Discrete Comput. Geom.* 11 (1994), 393–418.
4. P. K. Agarwal and M. Sharir, Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions, *SIAM J. Comput.* 25 (1996), 100–116.
5. B. Aronov, M. Pellegrini and M. Sharir, On the zone of a surface in a hyperplane arrangement, *Discrete Comput. Geom.* 9 (1993), 177–186.
6. M. Bern, D. P. Dobkin, D. Eppstein, and R. Grossman, Visibility with a moving point of view, *Algorithmica* 11 (1994), 360–378.
7. H. Brönnimann, O. Devillers, V. Dujmovic, H. Everett, M. Glisse, X. Goaoc, S. Lazard, H.-S. Na and S. Whitesides, Lines and free line segments tangent to arbitrary three-dimensional convex polyhedra, *SIAM J. Comput.* 37 (2007), 522–551.
8. R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symb. Comput.* 7 (1989), 11–30.
9. D. P. Dobkin and D. Kirkpatrick, A linear algorithm for determining the separation of convex polyhedra, *J. Algorithms* 6 (1985) 391–395.
10. H. Kaplan, N. Rubin, M. Sharir, Linear Data Structures for Fast Ray-Shooting amidst Convex Polyhedra, `http://www.cs.tau.ac.il/~rubinnat/fastRaySh.pdf`.
11. M. Pellegrini, Ray Shooting on Triangles in 3-Space. *Algorithmica* 9 (1993), 471–494.
12. M. Pellegrini, Ray shooting and lines in space, in *Handbook of Discrete and Computational Geometry*, 2nd edition, J. E. Goodman and J. O'Rourke (eds.), Chapman & Hall/CRC Press, Boca Raton, FL, 839–856, 2004.
13. F. P. Preparata and R. Tamassia, Efficient point location in a convex spatial cell complex, *SIAM J. Comput.* 21 (1992), 267–280.