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

Haim Kaplan† Natan Rubin‡ Micha Sharir§

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 $\tilde{O}(n \cdot \mathrm{poly}(k))$ preprocessing time and storage (where the $\tilde{O}(\cdot)$ notation hides polylogarithmic factors), 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 some fixed line with arbitrary origins and orientations. We also present a simpler solution for the case of vertical ray-shooting 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 [7, 15]. If Γ consists of arbitrary triangles in $\mathbb{R}^3$, the best known data structure is due to Pellegrini [14]; it requires $O(n^{4+\epsilon})$ preprocessing and storage and has logarithmic query time.[1] If Γ is the collection of facets on the boundary of a convex polyhedron, then an optimal algorithm, with $O(\log n)$ query time and $O(n)$ storage, can be obtained using the hierarchical decomposition of Dobkin and Kirkpatrick [12]. For the case where Γ consists of the boundary facets of k convex polyhedra, Agarwal

*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, by a grant from the AFIRST French-Israeli program, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. A preliminary version of this paper appeared in *Proc. 15th Annu. Europ. Sympos. Alg.* (2007), 287–298

†School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: haimk@post.tau.ac.il

‡School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: rubinnat@post.tau.ac.il

§School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA. E-mail: michas@post.tau.ac.il

[1] As is customary, upper bounds of the form $O(f(n) \cdot n^{\epsilon})$ mean that the actual upper bound is $C_\epsilon f(n) \cdot n^{\epsilon}$, which holds for any $\epsilon > 0$, where C_ϵ is a constant the depends on ϵ, and generally tends to infinity as ϵ decreases to 0.

and Sharir [5] describe a solution having polylogarithmic query time and $O(n^{2+\epsilon}k^2)$ storage, which is an improvement over [14] (when $k \ll n$). Still, this solution requires $\Omega(n^2)$ storage, even for small k, in contrast to the $O(n)$ storage achieved in [12] 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.* [9], 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 (or larger) to achieve polylogarithmic query time, even in the case where the input consists of k convex polyhedra, where k is small relative to n. For example, the recent algorithm of Aronov *et al.* [1], for the case of fat convex polyhedra, uses $O(n^{2+\epsilon})$ space and preprocessing time. The algorithm of de Berg [7], for the case of vertical ray-shooting, uses $O(n^{2+\epsilon} + K)$ storage, where K is the total complexity of the arrangement of the input polyhedra, which is known to be $O(nk^2)$; see also [15].

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. 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 a fixed line with arbitrary origins (and orientations). Finally, we present a simpler solution for the case of vertical ray-shooting. The two main 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 where 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 [4]). A plausible goal is thus to design algorithms whose storage is close to $O(nk^2)$. However, as mentioned, no previous algorithm solves these problems with performance that depends subquadratically on n, when $k \ll n$.

In general, we have not yet met the above goal, and we "pay" more in terms of k to achieve linear dependence on n. Specifically, in the case of ray-shooting with rays originating from a fixed line ℓ_0, where the input polyhedra may intersect each other, our solution uses $\tilde{O}(nk^5)$ storage and preprocessing, and answers queries in polylogarithmic time. However, if the input polyhedra are pairwise disjoint, the storage can be reduced to $\tilde{O}(nk^2)$. (As in the abstract, the notation $\tilde{O}(\cdot)$ hides polylogarithmic factors.) Thus, in the latter case, we do achieve the goal mentioned above. In the case involving rays orthogonal to a fixed line, our solution requires $\tilde{O}(nk^5)$ storage, $\tilde{O}(nk^6)$ preprocessing time, and supports queries in polylogarithmic time. (Here we assume the general model, where the polyhedra may intersect one another.) The latter problem is closely related to the more general problem involving rays contained in lines which intersect a fixed line ℓ_0, but whose origins are arbitrary, which we also solve and obtain similar bounds for the query time, storage and preprocessing costs. Our solution for the case of vertical ray-shooting (amidst possibly intersecting convex polyhedra) answers queries in polylogarithmic time and requires only $\tilde{O}(nk^2)$ storage and preprocessing time, which is close to the maximum complexity of an arrangement of k convex polyhedra in $\mathbb{R}^3$.

We note that when k is large, e.g., $k = \Theta(n)$, our solutions, except for the case where we shoot from a line and the given polyhedra are pairwise disjoint, and the case of vertical ray-shooting, are inferior to those obtained earlier. For example, for ray-shooting from a fixed line with possibly intersecting polyhedra, our solution requires $\tilde{O}(n^6)$ storage, as opposed to the $O(n^{3+\epsilon})$ storage used

by the general approach, as mentioned above. Nevertheless, for small values of k, our solutions improve all the previous ones, and extrapolate nicely from the linear storage required for $k = 1$. Thus we (partially) solve a major open problem posed in [5]. We note that the case of $k \ll n$ arises frequently in practice. For example, this is the situation when shooting in a scene consisting of balls or other more complex (convex) shapes, each approximated by a polyhedral surface consisting of many triangles. In spite of the apparent suboptimal dependence on k (we *know* it to be suboptimal only when k is large), we regard the results of the paper to be a first step towards achieving the performance conjectured above, and leave the task of tightening the dependence on k to further research.

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. In the case of ray-shooting from a line, or of ray-shooting along lines passing through a fixed line, or of ray-shooting orthogonal to a fixed line, we have three degrees of freedom, and we take $\mathbb{S}$ to be $\mathbb{R}^3$. Each polyhedron $P \in \mathcal{P}$ defines a two-dimensional 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. Now assume, without loss of generality, that ℓ_0 is disjoint of the union of $\mathcal{P}$. Then there are at most two polyhedra, for each cell C of $\mathcal{C}(\mathcal{P})$, whose boundaries are first hit by such rays r emanating from ℓ_0 and contained in lines $\ell \in C$. Thus, our query is easily answered by locating the cell of $\mathcal{C}(\mathcal{P})$ containing ℓ and then by answering ray-shooting queries amidst at most two polyhedra of $\mathcal{P}$. In Section 2, we modify the analysis of Brönnimann *et al.* [10] 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 [16] 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 $\ell_r \in C$; see Figure 4(a) for an illustration. To handle this difficulty, we refine the technique, by adding to $\mathcal{P}$ all *pairwise intersections* between the polyhedra of $\mathcal{P}$, obtaining 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 $\mathcal{C}(\mathcal{Q})$. 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 $\mathcal{Q}$. In the case of ray-shooting from a fixed line, 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 still supports queries in polylogarithmic time; see Section 3. In the case of ray-shooting with rays orthogonal to a fixed line, we proceed in a similar manner, but the preprocessing time is now $\tilde{O}(nk^6)$, due to the auxiliary structures stored with each cell $C \in \mathcal{C}(\mathcal{Q})$; see Section 4.

Let C be a cell in $\mathcal{C}(\mathcal{Q})$. Since $\mathcal{P} \subseteq \mathcal{Q}$, $\mathcal{C}(\mathcal{Q})$ is a *refinement* of $\mathcal{C}(\mathcal{P})$. Therefore, the set $\mathcal{P}_C$ of polyhedra stabbed by any line represented in C is well defined. 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 $\mathcal{P}$, consisting of all pairs (P, Q) as above, with P hit before Q along lines in C. Clearly, $\Xi \subseteq \mathcal{P}_C$ 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 [3]. 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. Overall, we obtain the performance bounds mentioned earlier.

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. Then, in Section 4 we apply the same approach (with a few modifications) to solve the ray-shooting problem involving rays with *arbitrary origins* but contained in lines which intersect ℓ_0. We show a simple reduction from the ray-shooting problem involving rays orthogonal to a fixed line with arbitrary origins amidst (possibly intersecting) polyhedra, to the latter problem. We also briefly outline a relatively simple solution for the vertical ray-shooting problem (amidst possibly intersecting polyhedra).

Preliminaries. We briefly describe some of the techniques used in this paper.

Fully dynamic point-location in monotone planar subdivisions. In [16], Preparata and Tamassia describe a data structure for point-location in a dynamic monotone planar subdivision. In addition to point location queries, it supports the following set of update operations (where it is assumed that the map remains monotone after each update).

- $Insert(v_1, v_2, r; e, r_1, r_2)$: inserts an edge e between vertices v_1 and v_2 inside region r, which is decomposed into regions r_1 and r_2 to the left and to the right of e, respectively.
- $Delete(v_1, v_2, e, r_1, r_2; r)$: removes an edge e between vertices v_1 and v_2, and merges, into a common region r, the two regions r_1 and r_2 formerly to the left and to the right of e, respectively.
- $Expand(v, r_1, r_2; v_1, v_2, e)$: expands vertex v into two vertices v_1 and v_2 connected by edge e, which has regions r_1 and r_2 to the left and to the right, respectively.
- $Contract(v_1, v_2, e; v)$: contracts edge e between vertices v_1 and v_2 into a single vertex v.

The data structure in [16] supports each of these operations in $O(\log^2 n)$ time, where n is the (maximal) complexity of the subdivision. We assume that a point location query can detect situations where the query point lies on an edge or at a vertex, and, if so, returns the corresponding feature of the planar map.

Using persistence, we can extend this data structure to answer point location queries in an xy-monotone subdivision $\mathcal{C}$ of $\mathbb{R}^3$. For that we have to be able to sweep $\mathbb{R}^3$ with a plane orthogonal to the y-axis, while maintaining the cross-section of $\mathcal{C}$ with the sweep plane in a planar point location data structure as above. We need to be able to keep track of toplogical changes in the cross-section, and, when such a change occurs, to update the planar point location data structure using the update operations listed above. The resulting (static) data structure in $\mathbb{R}^3$ requires $O(N \log^2 N)$ storage and preprocessing, and answers point-location queries in $O(\log^2 N)$ time, where N is the number of updates to the planar point location data structure over the entire sweeping process. For more details see [16].

Tangent lines in an arrangement of polyhedra. Let $\mathcal{P}$ be a set of k polyhedra in $\mathbb{R}^3$, and let ℓ_0 be a fixed line. 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).[2]

Let S be a set of edges and vertices (possibly including ℓ_0). A line ℓ is a *transversal* of S if ℓ intersects every element of 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.

Let $\ell \neq \ell_0$ be an isolated transversal through ℓ_0, and let S be the respective minimal set of relatively open edges and vertices, chosen from some of the polyhedra, such that ℓ is an isolated transversal to $S \cup \{\ell_0\}$. Let $\mathcal{T}$ be the respective set of polyhedra of $\mathcal{P}$ which contain an element of S (on their boundary). Let Π be the plane containing ℓ and ℓ_0. We say that ℓ is a *generic* isolated transversal of S through ℓ_0 if, for each $P \in \mathcal{T}$, ℓ is tangent to $P \cap \Pi$ in Π. In particular, ℓ is tangent to each polyhedron in $\mathcal{T}$.

Brönnimann *et al.* [10] have proved 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 polyhedra. This bound is a consequence of the following main theorem of [10].

Theorem 1.1. *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 admit a generic isolated transversal through ℓ_0.*

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 [10]. Repeating the analysis for each of the $O(n)$ lines that contain edges of the polyhedra (and applying some additional arguments), the bound $O(n^2k^2)$ follows.

Parametric search. Our most general ray-shooting scheme in Sections 3 and 4 is based on the parametric searching technique of Agarwal and Matoušek [3]. In this technique we build a data structure for solving segment intersection detection queries, each asking whether a query segment s intersects (the boundary of) any polyhedron of $\mathcal{P}$. Given a ray ρ, we replace it by the segment o_1o_2, where o_1 is the origin of ρ and o_2 is the first (unknown) intersection point between ρ and the given polyhedra. Since we do not know o_2, we feed into our data structure a generic, unspecified input o_2. As we will see below, each decision step of the detection algorithm, which depends on o_2, is easy to implement generically. The cost of a ray-shooting query is thus quadratic in the cost of a segment-intersection query. As we will show, the cost of the latter query is polylogarithmic, so this technique answers ray-shooting queries in polylogarithmic time as well.

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 [10]. 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, in some direction, 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

[2] A line ℓ can also support a polyhedron P by overlapping a facet of P. In this case ℓ has two distinct support vertices and/or edges of P.

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, nor its correctness.

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.)

In other words, the cross section of the surface σ_P of all tangents to P within Π_t consists of the union of $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$. We denote by $\mathcal{A}_t^*$ the arrangement of the curves $\{\mathcal{L}(P_t), \mathcal{U}(P_t)\}_{P_t \in \mathcal{P}_t}$, all of which are monotone, piecewise-linear, and unbounded. Clearly, $\mathcal{A}_t^*$ is a cross-section of $\mathcal{C}(\mathcal{P})$ at the chosen value of 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 $\mathcal{A}_t^*$. Conversely, each connected cell in $\mathcal{A}_t^*$ corresponds to maximal connected sets of lines passing through ℓ_0, such that all lines in the same region stab the same subset of $\mathcal{P}$ (in the same order).[3]

We will use the persistent approach of [16] for point-location in $\mathcal{C}(\mathcal{P})$ (see the review in the introduction). Specifically, we maintain $\mathcal{A}_t^*$, as t varies continuously from 0 to π, in a dynamic planar point location data structure for monotone subdivisions, and we update this structure persistently at each t where there is a topological change in $\mathcal{A}_t^*$. Persistence allows us to store all the resulting instances of the data structure in a compact manner for answering queries efficiently.

$\mathcal{A}_t^*$ changes topologically either when a single envelope undergoes a combinatorial change, or when an intersection point of two envelopes lies at a vertex of one of them, or when three envelopes meet at a common point. We keep track of these topological and combinatorial changes of $\mathcal{A}_t^*$, and update the corresponding point location data structure. We can use the four special operations to manipulate the data structure for dynamic point-location in a planar monotone subdivision, as reviewed in the introduction.

[3] 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.

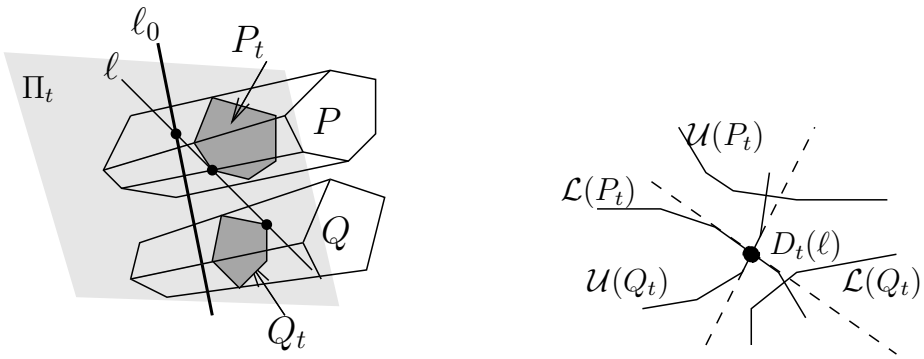

Figure 1: ℓ is a common tangent of P_t and Q_t in the primal plane Π_t (left). The dual arrangement $\mathcal{A}_t^*$ (right); $D_t(\ell)$ is an intersection of two lines, which are dual to the support vertices of ℓ in Π_t.

We assume, for simplicity, general position of the given polyhedra. In particular, we assume that each isolated transversal through ℓ_0 passes through exactly three edges, or through a vertex and an edge, of the given polyhedra, and that no edge of any polyhedron is coplanar with ℓ_0 or with a facet of another polyhedron.

Handling the critical events depends on their type, and proceeds as follows.

(i) A polyhedron $P \in \mathcal{P}$ is first intersected by the rotating plane Π_t at time $t = t^*$. By the general position assumption, Π_{t^*} meets P at a single vertex v. In Π_{t^*} both envelopes $\mathcal{U}(P_{t^*})$ and $\mathcal{L}(P_{t^*})$ are the single line λ dual to v.

The new edges that emerge in the planar embedding of $\mathcal{A}_t^*$, at time t^*, are determined as follows. First, we intersect each polyhedron Q_t that appears in $\mathcal{A}_t^*$ immediately before time t^*, with Π_{t^*}, and then compute the corresponding envelopes $\mathcal{U}(Q_{t^*})$ and $\mathcal{L}(Q_{t^*})$. We trace the intersection points of λ with each envelope and sort them along λ. Clearly, this way we spend a total of $O(n \log n)$ time to find the sequence $e_1, e_2, \ldots, e_m$, ordered from left to right, of the $m \leq k+1$ new edges (along λ) inserted to $\mathcal{A}_{t^*}^*$, at time t^*. Using the current version of the point-location structure (which is consistent with $\mathcal{A}_t^*$, for $t < t^*$ arbitrary close to t^*), we can locate in $\mathcal{A}_t^*$ the edges containing the endpoints of the newly inserted edges. We split each such edge, by applying an $Expand(\cdot)$ operation on one of its endpoints. Similarly, we find for each edge e_i, the region r_i in $\mathcal{A}_t^*$ containing it. We then apply $Insert(\cdot)$ operations to insert the newly created edges into the point location data structure of $\mathcal{A}_t^*$. By inserting an edge e_i, for $1 \leq i \leq m$, we split the region r_i, into a pair of new regions r_i' and r_i'' above and below e_i, respectively. In $\mathcal{A}_{t^*}^*$ the regions r_i' and r_i'' belong to the cross section of the same cell of $\mathcal{C}(\mathcal{P})$ as r_i in $\mathcal{A}_t^*$ for $t < t^*$.[4]

As we rotate further "into" P, the lines forming the envelopes are dual to the vertices of P_t that lie on the edges of P emanating from v. Assume $t > t^*$ is arbitrary close to t^*, then $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ almost coincide with λ. (For an illustration, see Figure 2). Let h_v be the degree of v in P. We compute the envelopes $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ in $O(h_v \log h_v)$ time, where h_v is the number of edges of P incident to v.

We then insert $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ (instead of λ) into $\mathcal{A}_t^*$ as follows. We traverse the edges e_i, for $1 \leq i \leq m$, of λ in $\mathcal{A}_t^*$ from left to right. We replace each such edge e_i by two monotone chains of edges, one is a sequence of edges of $\mathcal{U}(P_t)$ that replaces e_i, and the other is a sequence of edges of $\mathcal{L}(P_t)$ that replaces e_i. To actually perform the insertion, we first replicate the $m-1$

[4]We maintain the labels of the regions in $\mathcal{A}_t^*$, for $0 \leq t < \pi$, in a Union-Find data structure, such that at the end of the sweep the labels of all regions associated with a particular cell of $\mathcal{C}(\mathcal{P})$ are in the same set. Here we insert r_i' and r_i'' into the same set containing r_i. This Union-Find structure allows us to identify all cells of $\mathcal{C}(\mathcal{P})$ and to map regions of $\mathcal{A}_t^*$ to the cells they belong to.

endpoints[5] of the edges $e_1, \ldots, e_m$ on λ using $Expand(\cdot)$ operations, and insert a copy e'_i of each edge e_i immediately above it, for every $1 \leq i \leq m$. The insertion of e'_i, for $1 \leq i \leq m$, splits the region r'_i adjacent to e_i, into two regions, the upper of which, which belongs to a cross section of the same cell as r'_i, is also given the label r'_i. The lower region is given a new name. We then apply a sequence of $Expand(\cdot)$ operations on the endpoints of e'_i and e_i, to insert the vertices of $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ into $\mathcal{A}^*_t$, respectively. Clearly, all this takes, in addition to the cost $O(h_v \log h_v)$ of constructing $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$, $O((h_v + k)\log^2 n)$ time, where the cost is dominated by $O(h_v + k)$ updates of the dynamic point location structure.

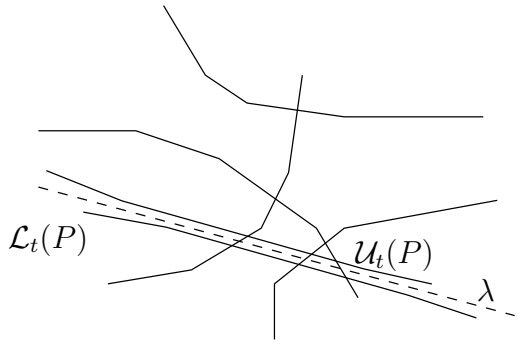

Figure 2: For $t > t^*$ close enough to t^*, the envelopes $\mathcal{U}(P_t)$, $\mathcal{L}(P_t)$ are very close to λ.

Since there are at most k events of type (i), the total cost of processing them is thus $O(nk \log^2 n)$.

(ii) A polyhedron $P \in \mathcal{P}$ leaves the rotating plane at time $t = t^*$. These events are handled in a symmetric manner to those of type (i). Here we have to delete the edges of the envelopes $\mathcal{U}(P_{t^*})$ and $\mathcal{L}(P_{t^*})$ from $\mathcal{A}^*_t$, using the $Contract(\cdot)$ and $Delete(\cdot)$ operations on the point location structure. The total cost of processing events of type (ii) is thus also $O(nk \log^2 n)$.

(iii) Π_t passes, at time t^*, through a vertex v of some $P \in \mathcal{P}$, which is not extreme in the rotation order. As t sweeps past t^*, a sequence of vertices of P_t that correspond to the "backward" edges, incident to v and intersected by Π_t immediately before t^*, is replaced by a sequence of new vertices that correspond to the "forward" edges of P incident to v. Let h_v^- and h_v^+ be the "backward" (resp., "forward") degree of v in P (with respect to the chosen sweep direction).

In between, at $t = t^*$, the envelopes, $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$, degenerate so that their edges that lie on lines dual to the intersections of "backward" edges, incident to v, with Π_t become collinear and lie on the line, λ, dual to v, at $t = t^*$.[6] We compute these edges of $\mathcal{U}(P_t) \cup \mathcal{L}(P_t)$ in a brute-force manner, in $O(h_v^- \log h_v^-)$ time. Since λ has $O(k)$ intersections with the other envelopes $\mathcal{U}(Q_{t^*})$ and $\mathcal{L}(Q_{t^*})$, it takes $O((k + h_v^-)\log^2 n)$ time to trace the degenerating edges of $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ in $\mathcal{A}^*_t$ (by querying the point location structure of $\mathcal{A}^*_t$). We have thus found $O(h_v^- + k)$ edges in $\mathcal{A}^*_t$ that become collinear, as t tends to t^*.

We remove each vertex of $\mathcal{U}(P_t)$ (resp., $\mathcal{L}(P_t)$), which becomes redundant at time t^*, from the point-location structure in $\mathcal{A}^*_t$, by applying a single $Contract(\cdot)$ operation on one of its incident edges. (By the general position assumption, there are two such edges, both on $\mathcal{U}(P_t)$ or both on $\mathcal{L}(P_t)$.) Clearly, this takes $O(h_v^- \log^2 n)$ time.

Symmetrically, as v leaves Π_t, a segment of $\mathcal{U}(P_t)$ (resp., $\mathcal{L}(P_t)$), which is contained in λ, splits into a sequence of edges (that become collinear as $t > t^*$ tends to t^*). We compute it in $O(h_v^+ \log h_v^+)$

[5] The edges e_1 and e_m are unbounded edges (rays) having a single endpoint.

[6] If v is a silhouette vertex of P_t (with respect to the direction orthogonal to ℓ_0) and therefore belongs to both the "upper" and the "lower" hulls of P, then each of the envelopes $\mathcal{U}(P_t)$ and $\mathcal{L}(P_t)$ contains a connected sequence of edges merging into a segment of λ. Otherwise, this happens only on one of the envelopes $\mathcal{U}(P_t)$ or $\mathcal{L}(P_t)$.

time, and then apply $O(h_v^+)$ $Expand(\cdot)$ operations, to insert the new vertices of $\mathcal{U}(P_t)$ (resp., $\mathcal{L}(P_t)$) into the point-location structure. Clearly, this takes a total of $O(h_v^+ \log^2 n)$ time.

To recap, we spend a total of $O((h_v^+ + h_v^- + k)\log^2 n)$ at each event of this type. Since the sum of the vertex degrees, over all polyhedra vertices, is $O(n)$, we spend $O(nk \log^2 n)$ time at all the events of type (iii).

(iv) There is a pair of polyhedra P, Q, such that an intersection between two respective envelopes, say $\mathcal{L}(P_t)$ and $\mathcal{U}(Q_t)$, coincides with a vertex of, say $\mathcal{L}(P_t)$, at time $t = t^*$. See Figure 3.

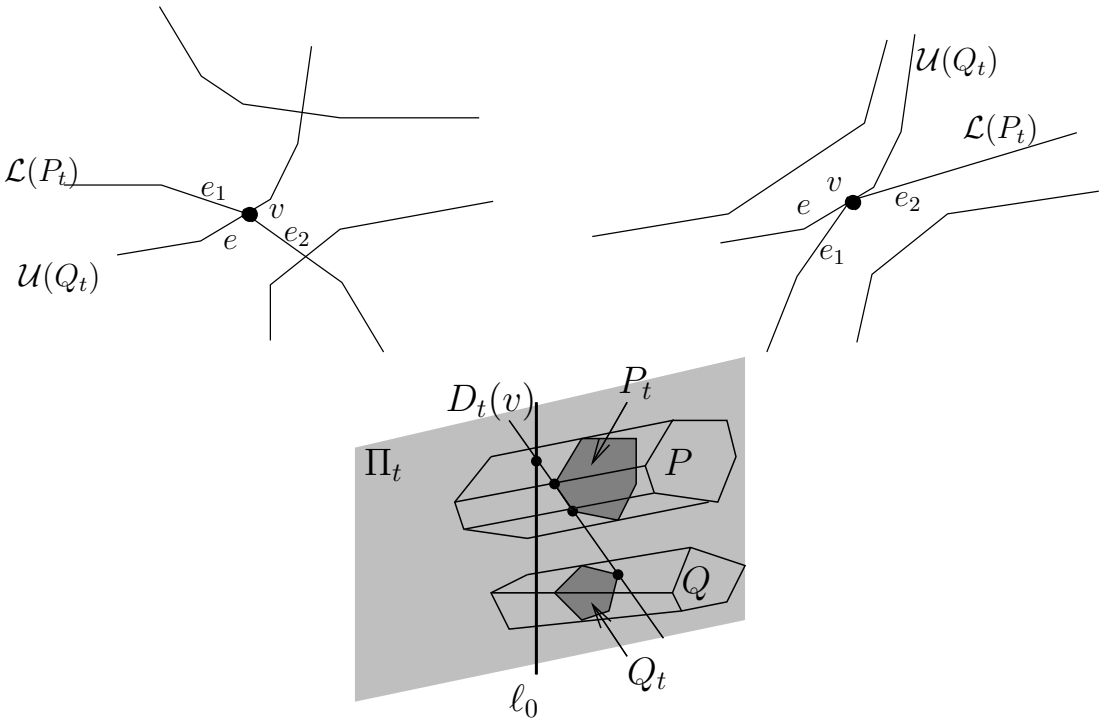

Figure 3: An edge e of $\mathcal{U}(Q_t)$ coincides with a vertex v of $\mathcal{L}(P_t)$. v is dual to a line $D_t(v)$ which is tangent to P and Q at a facet and an edge, respectively.

Let e be the edge of $\mathcal{U}(Q_t)$ incident to the intersection point, let v be the corresponding vertex of $\mathcal{L}(P_t)$, and let e_1 and e_2 be the two edges incident to v on $\mathcal{L}(P_t)$. Assume that the values of t^*, e, e_1, e_2 and v are already known to us. (We describe how to get them shortly.) We query the point location data-structure $\mathcal{A}_t^*$ for t right before t^* with v and $e \cap e_1$, to determine the features of $\mathcal{A}_t^*$ involved in this event.

We distinguish between two possible cases. Consider the case where e intersects e_1 in $\mathcal{A}_t^*$ for $t < t^*$ right before t^*, and e intersects e_2 for $t > t^*$ right after t^*. We update the point location structure by first contracting an edge of $\mathcal{A}_t^*$ which connects v to the intersection of e_1 and e. This gives us the point location data structure for $\mathcal{A}_{t^*}^*$. Then we expand the vertex v so that the newly created edge connects the intersection of e_2 and e with v to obtain the data structure for $\mathcal{A}_t^*$ where $t > t^*$. See Figure 3 (left).

Now consider the case when e intersects both e_1 and e_2 immediately before time t^*, and does not intersect e_1 and e_2 immediately after time t^* (or vice versa). Consider $\mathcal{A}_t^*$ immediately before time t^*. We delete the edge between $e_1 \cap e$ and $e_2 \cap e$. The new region created by this deletion is given the same name as for the region on the side of the deleted edge which is opposite to v. In addition we contract the edge connecting v with $e_1 \cap e$ and the edge connecting v with $e_2 \cap e$. Now the point location data structure stores $\mathcal{A}_{t^*}^*$ Clearly, $\mathcal{A}_{t^*}^*$ contains four regions adjacent to v: one

region on one side of e and three regions on the other side. Let the latter regions be r_1, r_2, and r_3, ordered in counter-clockwise order around v. To modify the point location data structure so that it represents $\mathcal{A}_t^*$ for t right after t^*, we expand v into two vertices v_1 and v_2, so that the new edge is incident to r_1 and r_3, v_1 incident to two pieces of e and v_2 is incident to e_1 and e_2. Subsequently we delete the newly created edge, thereby merging r_1 and r_3 into a new region r.[7] We also contract one of the two edges incident to v_1. If e does not intersect e_1 or e_2 immediately before time t^*, and intersects both of them immediately afterwards, we act symmetrically. See Figure 3 (right).

Thus, knowing t^*, v, e, e_1, and e_2, we spend $O(\log^2 n)$ time on each event of this type.

To find these quantities, we note that, at t^*, $\mathcal{L}(P_{t^*})$ and $\mathcal{U}(Q_{t^*})$ intersect at v. So at t^*, v is dual to a line $D_{t^*}(v)$ tangent to both polyhedra and to ℓ_0. The line containing e is dual to an intersection point of an edge of P with Π_{t^*}, so $D_{t^*}(v)$ is tangent to that edge. Similarly, $D_{t^*}(v)$ is tangent to a pair of adjacent edges of Q, corresponding to e_1 and e_2, so $D_{t^*}(v)$ is tangent to the facet of Q bounded by e_1 and e_2. Hence, $D_{t^*}(v)$ is tangent (in Π_{t^*}) to P_{t^*} and Q_{t^*} at a vertex and at an edge, respectively. In other words, $D_{t^*}(v)$ passes through ℓ_0, and is tangent to P at one edge and to Q at two edges, both lying on a common facet. It then follows that this set of supports admits a generic isolated transversal through ℓ_0 and is minimal with this property. We can thus use the algorithm of [10] (see also Theorem 1.1) to find all the $O(nk^2)$ minimal sets of polyhedra vertices and edges that admit an isolated transversal of this kind, in time $O(nk^2 \log n)$. This gives us the list of critical events of type (iv), and associates, with each such event, the corresponding values of t^*, v, e, e_1, and e_2.

(v) There is a triple of polyhedra P, Q, R, such that some corresponding triple of envelopes, say $\mathcal{U}(P_t)$, $\mathcal{U}(Q_t)$ and $\mathcal{U}(R_t)$ intersect at a single point q at time $t = t^*$. As in the previous case, knowing the edges of $\mathcal{U}(P_t)$, $\mathcal{U}(Q_t)$ and $\mathcal{U}(R_t)$ involved in the intersection, we can update the point location structure at each event of type (v), in $O(\log^2 n)$ time, by a constant number of operations. We leave it to the reader to verify the details.

Clearly, the intersection point v is tangent (in Π_{t^*}) to each one of P_{t^*}, Q_{t^*} and R_{t^*} at a respective vertex. Equivalently, v is dual to a common tangent line ℓ to P, Q and R, which passes through ℓ_0 and is supported by an edge on each of these polyhedra. The set of support edges for $D_{t^*}(v)$ admits a generic isolated transversal through ℓ_0 and is minimal with this property. By Theorem 1.1, there is a total of $O(nk^2)$ such sets of edges. They can be computed, using the algorithm of [10], in $O(nk^2 \log n)$ time, and this gives us an enumeration of the events of type (v), and also associates with each event the edges of the corresponding envelopes that become concurrent.

In summary, plugging all these into the machinery of [16], we have:

Theorem 2.1. *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.

We say that a pair of cells $C, C' \in \mathcal{C}(\mathcal{P})$ are neighbors if C' can be reached from C by crossing a single 2-surface σ_P (as defined in the introduction), for some $P \in \mathcal{P}$. 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 list of polyhedra in $\mathcal{P}_C \cup \{\ell_0\}$, 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 lower-dimensional face in $\mathcal{C}(\mathcal{P})$). In

[7] When this happens we unite the sets containing r_1 and r_3 in our union-find data structure and we insert r to the resulting set as they all represent the same cell of $\mathcal{C}(\mathcal{P})$.

each such change we either insert a new polyhedron P into the list, or remove a polyhedron from this list. We maintain the lists in a persistent search tree, as follows.

During the sweep algorithm of Theorem 2.1, we construct the adjacency graph on the three-dimensional cells of $\mathcal{C}(\mathcal{P})$. We detect that a pair of cells C and C' are adjacent, when their cross-sections, say C_t and C'_t, become adjacent for the first time in $\mathcal{A}^*_t$. This can only happen when we insert an edge e which separates C_t and C'_t in $\mathcal{A}^*_t$. Let $P \in \mathcal{P}$ be the polyhedron for which e lies in either $\mathcal{L}(P_t)$ or $\mathcal{U}(P_t)$. Then C can be reached from C' across σ_P. So each time we insert a new edge e into $\mathcal{A}^*_t$ we store the tuple (ℓ, P) where ℓ is a line (represented by a point in $\mathcal{A}^*_t$) in the interior of e, and P is as above. After we have completed the construction of the point-location structure for $\mathcal{C}(\mathcal{P})$ (and we know all its cells which are represented as sets in our Union-Find data structure), we scan the list of all the previously stored tuples (ℓ, P). For each tuple (ℓ, P), we find a pair of cells C and C' containing ℓ on their boundary, and connect them by an edge (if they are not already connected). We also store the values of P and ℓ with the corresponding edge. Clearly, the construction takes $O(nk^2 \log^2 n)$ time.

We next claim that the adjacency graph of the three-dimensional cells of $\mathcal{C}(\mathcal{P})$ is connected. Indeed, by the general position assumption, the intersection of every pair of curves σ_P and σ_Q, of points representing tangents to polyhedra $P \in \mathcal{P}$ and $Q \in \mathcal{P}$, respectively, is a (possibly disconnected) one-dimensional curve in $\mathbb{S}$. Accordingly, the intersection of every triple of curves σ_P, σ_Q, and σ_R, is a set of points in $\mathbb{S}$. Hence, every three-dimensional cell can be reached across a two-dimensional face.

We then traverse the above adjacency graph in breadth-first fashion, and when we move from a cell C to an adjacent cell C', we construct $\mathcal{P}_{C'}$ by updating $\mathcal{P}_C$ in a persistent manner. To obtain $\mathcal{P}_{C'}$ from $\mathcal{P}_C$ we have to insert or delete a polyhedron (we always know what action to take).

To insert P into the ordered list $\mathcal{P}_C \cup \{\ell_0\}$, we perfrom a binary search along the line ℓ on the common boundary of C and C', which is stored with the edge connecting them. The binary search may require $O(\log k)$ ray-shooting queries at individual polyhedra of $\mathcal{P}_C$ and P. Deletions are done symmetrically. Thus, passing from one cell of $\mathcal{C}(\mathcal{P})$ to another takes $O(\log^2 n)$ time. It follows that we need $O(nk^2 \log^2 n)$ extra storage and preprocessing time to persistently store the lists P_C, over all cells $C \in \mathcal{C}(\mathcal{P})$.

Theorem 2.2. *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 per query.*

So far, we have assumed general position of $\mathcal{P}$. In Section 3, we will apply this machinery in a degenerate situation, where the set of input polyhedra is $\mathcal{Q} = \mathcal{P} \cup \{P \cap Q \mid P, Q \in \mathcal{P}\}$. Hence, a generic isolated transversal that emanates from ℓ_0 can meet more than three edges or vertices of polyhedra, and the polyhedra can have any number of collinear or coplanar vertices and edges. The algorithms of Theorem 2.1, and of Theorem 2.2 also work in degenerate settings, by introducing routine (but tedious) modifications needed for handling degenerate vertices, edges and faces of $\mathcal{C}(\mathcal{P})$ (which may now be contained in any number of envelopes of the form $\mathcal{U}(P_t)$ or $\mathcal{L}(P_t)$); for more details see [8]. In these cases, although multiple topological changes happen at the same time t, they are ordered consistently, by the above modification, so as to enable us to process one event at a time.[8]

As one can easily verify, we still spend $O(nk \log^2 n)$ time to process (topology changes caused by) events of types (i)-(iii). Each of the remaining events is uniquely charged to a minimal (constant-

[8]Multiple simultaneous sweep events may induce degenerate features in the cross-section $\mathcal{A}^*_{t^*}$, at a critical time t^*. When we process topology events at t^* we first create the data structure of $\mathcal{A}^*_{t^*}$ and only then create the data structure of $\mathcal{A}^*_t$, for $t > t^*$. That is, each event is split into two parts, one contains the changes required at time t^* and the other contains the changes for $t > t^*$. We perform the first part of all events first and only then the second part of the events.

size) set S of relatively open edges or vertices, of the input polyhedra which admits a generic isolated transversal through ℓ_0. Since a single line may now be an isolated transversal to any number of edges and vertices of the input polyhedra, it might yield multiple sweep events of type (iv)-(v), which occur simultaneously. Fortunately, Theorem 1.1, also holds in degenerate settings. This implies that the $O(nk^2)$ bound on the number of events of type (iv)-(v) still holds. A consistently ordered list of those events can be computed in $O(nk^2 \log n)$ time, by carefully applying the algorithm of [10]. We process all these events in $O(nk^2 \log^2 n)$ time.

3 Ray-Shooting from a Line: Handling Intersecting Polyhedra

The data structure. 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 (see Figure 4 (a)). In this section we consider this case, and derive a solution which is less efficient, but still nearly linear in n.

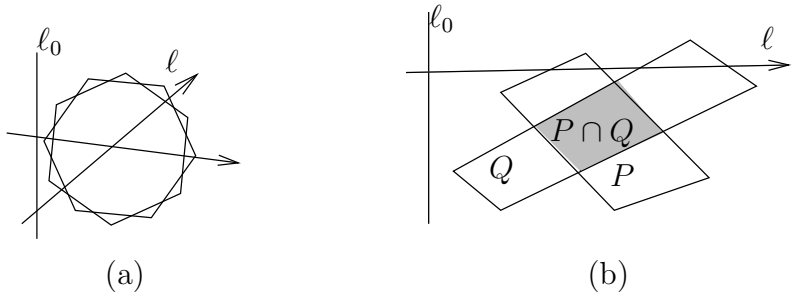

Figure 4: The case of intersecting polyhedra. (a) As we rotate ℓ, the first polyhedron boundary intersected by ℓ changes $\Theta(n)$ times. (b) $P \prec_C Q$, for the cell C containing ℓ; the polyhedron $P \cap Q$ is not in $\mathcal{Q}_C$.

As described in Section 1, we use parametric search, in the style of [3], to reduce the problem to segment intersection detection. Specifically, the problem is reduced to testing a segment $s = o_1 o_2$, such that o_1 is on ℓ_0, for intersection with the boundaries of 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 [16]. We note, though, that $\mathcal{A}$ is not *xy-monotone*, as required in [16]. 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 [16], 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 (see, e.g., [2]). 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 the boundary of 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 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 the other polyhedra unchanged. The polyhedra of $\mathcal{P}$ may now contain *artificial facets*, tangent to ℓ_0. Since we are interested in testing s for intersection with non-artificial faces of polyhedra of $\mathcal{P}$, the artificial faces will require special treatment throughout the intersection-testing algorithm.[9]

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. 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 (in a nonartificial face) 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 such boundary points along ℓ. That is, $Front(C)$ consists of all polyhedra that are first intersected by positive rays ℓ^+, for $\ell \in C$.

We define a strict 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 4 (b) for an illustration. In this section we only use the restriction of $\prec_C$ to the polyhedra of $\mathcal{P}_C^+$.

Lemma 3.1. *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 strict partial order.*

Proof. Let $P, Q \in \mathcal{P}_C$ be a pair of polyhedra such that $P \cap Q \notin \mathcal{Q}_C$. Since $P \cap Q$ is not stabbed by any line in C, the segments $P \cap \ell$ and $Q \cap \ell$ are disjoint for all lines ℓ in C. A simple continuity argument, which exploits the connectivity of C, the fact that continuous motion in C corresponds to continuous motion of the line in 3-space, and the fact that no point of C represents a line parallel to ℓ_0, implies that this order of $P \cap \ell$ and $Q \cap \ell$ is the same for all lines ℓ in C.

This establishes part (a), from which part (b) is immediate. □

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 3.2. *A (non-empty) 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 $\bigcap \mathcal{T} = \bigcap_{P\in\mathcal{T}} P$.*

[9] Note that the arrangement $\mathcal{A}$, and the point-location structure in $\mathcal{A}$, are still defined with respect to the original set of polyhedra.

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 5 (a).) Since $\overline{Front}(C)$ is a maximal antichain under $\prec_C$, we obtain the following corollary.

Corollary 3.3. *Assume $\mathcal{P}_C^+ \neq \emptyset$. Then $\overline{Front}(C) \neq \emptyset$, and every line in C intersects the polyhedron $\bigcap \overline{Front}(C)$.*

See Figure 5 (a) for an illustration.

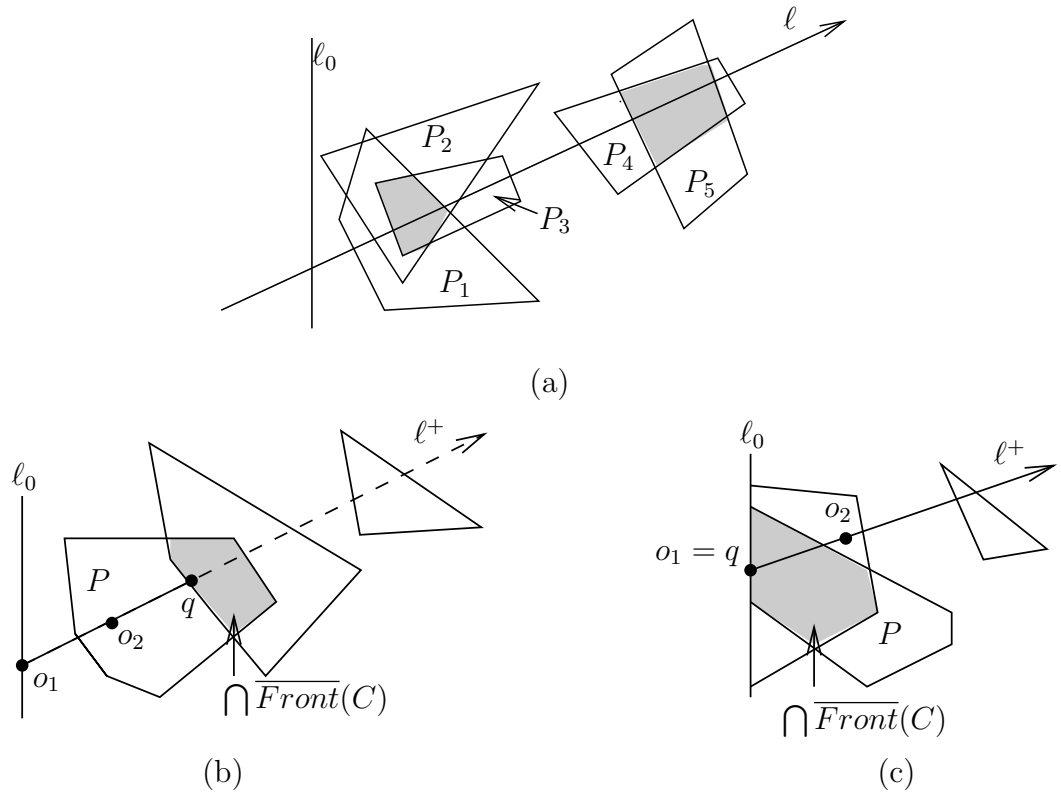

Figure 5: (a) Illustrating $\overline{Front}(C)$, for the cell C containing ℓ. We have $Front(C) = \{P_1, P_2\}$ and $\overline{Front}(C) = \{P_1, P_2, P_3\}$. The maximal antichains are $\{P_1, P_2, P_3\}$ and $\{P_4, P_5\}$. (b) Testing $o_1 o_2$, where $o_2 \in o_1 q$, for intersection with polyhedra boundaries (case (i)). (c) Testing $o_1 o_2$, where $o_2 \in o_1 q$, for intersection with polyhedra boundaries (case (ii)).

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 $\ell \in C$. We present the details about this structure, and the analysis of its storage and construction cost, later in this section.

Answering Segment Intersection Detection Queries.

Lemma 3.4. *Let ℓ be a line through ℓ_0, let ℓ^+ be the positive ray of ℓ with origin $o_1 \in \ell_0$ not contained in any (non-artificial) facet on the boundary of any polyhedron of $\mathcal{P}$, and let C be the cell of $\mathcal{C}(\mathcal{Q})$ that contains ℓ. Assume $P_C^+ \neq \emptyset$. Let q be the first intersection of ℓ^+ with $\bigcap \overline{Front}(C)$. (i) If $q \neq o_1$ then, for all points o_2 in the segment o_1q of ℓ^+, the segment $s = o_1o_2$ intersects a (non-artificial) facet on the boundary of some polyhedron of $\mathcal{P}$ if and only if o_1 and o_2 belong to distinct cells of $\mathcal{A}$. (ii) If $q = o_1$ then the above property holds for all points $o_2 \in \ell^+$.*

See Figure 5 (b and c) for an illustration. Note that, by Corollary 3.3, ℓ^+ intersects $\bigcap \overline{Front}(C)$. We have $q = o_1$ if and only if all the polyhedra in $\overline{Front}(C)$ contain artificial facets (all of which contain o_1); see Figure 5 (c) for an illustration.

Proof. If o_1 and o_2 belong to distinct cells of $\mathcal{A}$, then s clearly intersects the boundary of some polyhedron of $\mathcal{P}$. For the converse statement, assume that s intersects a non-artificial facet on the boundary of some polyhedron P of $\mathcal{P}$, at some point w, and take P to be that polyhedron for which w is closest to o_1 (if there is more than one such polyhedra, take P to be any of these polyhedra). Suppose first that w is an entry point into P. In this case, it is easily checked that $P \in \overline{Front}(C)$, and thus $q \neq o_1$. Moreover, as we follow ℓ^+ from w to q we cannot exit P, for then we would have also exited $\bigcap \overline{Front}(C)$, contradicting the definition of q. Hence, o_2, which clearly belongs to wq, lies in P, while o_1 lies outside P, so o_1 and o_2 lie in distinct cells of $\mathcal{A}$.

Suppose next that w is an exit point from P. In this case, each of the polyhedra in $\overline{Front}(C)$ has an artificial facet containing o_1. Since o_2 clearly lies beyond w, o_2 lies outside P while o_1 lies inside P (before splitting it by an artificial facet). Hence, they again lie in distinct cells of $\mathcal{A}$.

Note that in both cases if $o_2 = w$ then then o_2 resides in a lower dimensional cell different from the cell of o_1. □

Now we are ready to describe our query algorithm which tests, for a query segment $s = o_1o_2$ contained in some positive ray ℓ^+ with origin $o_1 \in \ell_0$, whether s intersects a (non-artificial) facet on the boundary of some polyhedron of $\mathcal{P}$. First, we test if o_1 is contained in a (non-artificial) facet of some polyhedron of $\mathcal{P}$ using the data structure for point location in $\mathcal{A}$, and report intersection if the answer is positive. Then we query the spatial point location structure of $\mathcal{C}(\mathcal{Q})$ to find the cell C which contains the line ℓ containing s. If $\mathcal{P}_C^+$ is empty then we return and report no intersection. Otherwise, we perform a ray-shooting query at $\bigcap \overline{Front}(C)$ to find the first intersection point q of ℓ^+ with the boundary of $\bigcap \overline{Front}(C)$. If $q \neq o_1$, and o_2 appears along ℓ^+ after q, we report intersection and finish. Otherwise (i.e., either $q \neq o_1$ and o_2 precedes q, or $q = o_1$ and o_2 is arbitrary), we search the data structure for point location in $\mathcal{A}$ with o_2 to test whether o_1 and o_2 belong to different cells of $\mathcal{A}$. We report intersection if and only if the answer is positive. The correctness of this procedure follows from Lemma 3.4.

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 3.5. *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. 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 f is on $\partial(\bigcap \overline{Front}(C))$. □

Therefore, it suffices to solve, for each cell C, the ray-shooting problem only amidst the faces f of $\mathcal{A}$ that lie on the boundary of $\bigcap \overline{Front}(C)$ and are good for $\overline{Front}(C)$. See Figure 6(a) for an illustration.

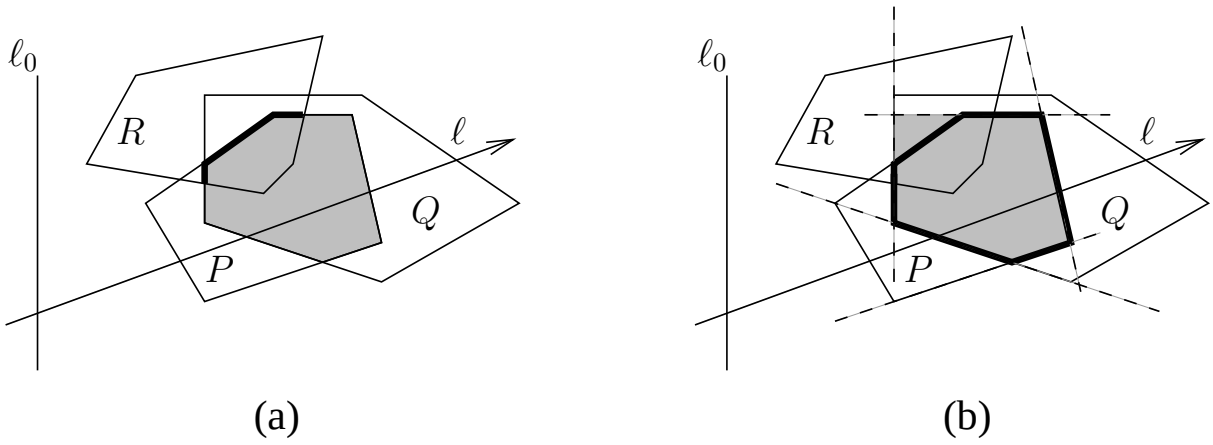

Figure 6: (a) For the cell C which contains ℓ, $\bigcap \overline{Front}(C) = P \cap Q$ (shaded). The bold facets f are not good for $\overline{Front}(C)$, because f is contained in a polyhedron R which is not in $\overline{Front}(C)$. Since R is not in $\mathcal{P}_C$, f is not stabbed by lines in C. (b) The polyhedron $G(\overline{Front}(C))$ (shaded) is bounded by affine extensions of the facets of $\bigcap \overline{Front}(C)$ which are good for $\overline{Front}(C)$ (whose boundary is drawn bold).

Let f be a facet of $\mathcal{A}$, and put $\mathcal{P}_f$ to be the set of polyhedra containing f. Equivalently, $\mathcal{P}_f$ is the unique set of polyhedra $\mathcal{P}'$ such that f is good for $\mathcal{P}'$. Assume $\mathcal{P}_f$ is not empty. Let h_f be the plane containing f, and let h_f^+ be the halfspace bounded by h_f and containing $\bigcap \mathcal{P}_f$. For a subset $\mathcal{P}'$ of $\mathcal{P}$, 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 $\mathcal{P}'$ lies on $\partial G(\mathcal{P}')$. In particular, we have the following property. (See Figure 6 (b) for an illustration.)

Lemma 3.6. *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 the intersection of ℓ^+ with $\bigcap \overline{Front}(C)$ and the intersection of ℓ^+ with $G(\overline{Front}(C))$ are identical.*

Proof. We show that the first intersection point of ℓ^+ with $\bigcap \overline{Front}(C)$ and $G(\overline{Front}(C))$ is the same. By Lemma 3.5, ℓ^+ first hits $\bigcap \overline{Front}(C)$ at a point q contained in a facet f of $\mathcal{A}$ which is good for $\overline{Front}(C)$. As just noted, f is contained in the boundary of $G(\overline{Front}(C))$, which is easily seen to imply that ℓ^+ first hits $G(\overline{Front}(C))$ at q. A symmetric argument shows that the last intersection point of ℓ^+ with $\bigcap \overline{Front}(C)$ and $G(\overline{Front}(C))$ is the same. □

For the correctness of the data structure of this section, it suffices that the first intersection point of ℓ^+ with $\bigcap \overline{Front}(C)$ and $G(\overline{Front}(C))$ is the same, but the data structure in Section 4 requires Lemma 3.6 in its more general form.

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 [12], 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 $\mathcal{P}_f$ of polyhedra, the total storage and preprocessing required for all these auxiliary ray-shooting data-structures is $O(nk^2)$ and $O(nk^2 \log n)$, respectively.

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 $\mathcal{C}(\mathcal{Q})$), and supports segment intersection detection queries in $O(\log^2 n)$ time.

Ray shooting via parametric searching. We next apply the parametric searching technique to turn the segment intersection detection procedure to one that actually performs ray shooting. Let $o_1 \in \ell_0$ be the origin of the positive query ray ℓ^+. We first locate o_1 in the arrangement $\mathcal{A}$ and return o_1 as an answer to the query if o_1 belongs to a (non-artificial) facet on the boundary of some polyhedron of $\mathcal{P}$. Otherwise, we find the cell C of $\mathcal{C}(\mathcal{Q})$ containing the line ℓ which supports ℓ^+. If $\mathcal{P}_C^+ = \emptyset$, ℓ^+ does not intersect any polyhedra boundary. Otherwise, we shoot along ℓ to find the first point q at which ℓ^+ intersects $G(\overline{Front}(C))$, in $O(\log n)$ time. We note that if $o_1 \neq q$ then the actual answer to the ray-shooting query must lie in the segment o_1q of ℓ^+. So far, no parametric searching is involved.

We now perform parametrically the last part of the segment intersection detection procedure, in which we test whether o_1 and o_2 (which is the unknown answer to the ray shooting query) are in the same cell of $\mathcal{A}$ (we note that if $q \neq o_1$ then the point o_2 necessarily precedes or coincides with q). We use the algorithm of Preparata and Tamassia generically with the (unknown) query point o_2.

The tests performed by the Preparata-Tamassia algorithm are of the following kinds: (i) Comparing the z-coordinate of o_2 with some critical z-value (at which the topology of the cross-section of $\mathcal{A}$ changes); (ii) Comparing the y-coordinate of o_2 with that of a vertex of the planar cross-section of $\mathcal{A}$ containing o_2 (this vertex lies on an edge e of $\mathcal{A}$); (iii) Testing which side of an edge of the cross-section contains o_2 (this edge is contained in some faces f of $\mathcal{A}$). Each of these tests is easy to implement generically: In case (i) we find the point $o' \in \ell^+$ which has the tested z-value; in case (ii) we find the point $o' \in \ell^+$ whose y and z-coordinates coincide with those of a point on e; and in case (iii) we find the point $o' \in \ell^+$ at which ℓ^+ intersects the plane containing f. In either of these cases we resolve the comparison by deciding whether or not o_2 precedes o' on ℓ^+, that is, whether or not o_1o' intersects any polyhedron boundary. Recall that the segment intersection detection routine can also detect the case where o' is the first intersection of ℓ^+ with a polyhedron boundary. If this is the case, we stop right away and output o'. One can easily check that if $q = o_1$ then the generic procedure will either decide that ℓ^+ does not intersect any polyhedra[10] or reach a comparison whose critical value o' is indeed the output to our ray-shooting query. Similarly, if $q \neq o_1$ then the generic procedure will also reach a comparison, whose critical value o', is the output to our ray-shooting query (in this case o' could be q).

This parametric part, which dominates the running time, takes $O(\log^4 n)$ time, since we answer each of the $O(\log^2 n)$ comparisons in the generic execution of this procedure in $O(\log^2 n)$ time.

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 subset $\mathcal{P}_f \subseteq \mathcal{P}$, 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 $O(nk^3 \log n)$ total time[11]. We represent each subset $\mathcal{P}_f$ by a bit-vector in $\{0,1\}^k$ that has 1 in position j if and only if the jth polyhedron is in $\mathcal{P}_f$. 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 bit-vectors $\mathcal{P}_f$, 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 $G(\mathcal{T})$. Clearly, given a bit-vector representing $\mathcal{T}$, the corresponding ray-shooting data structure amidst $\bigcap \mathcal{T}$ can be found in $O(k \log n)$ time.

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. As in Section 2, we assume without loss of generality, general position of the input set of polyhedra $\mathcal{P}$. We say that a pair of cells $C, C' \in \mathcal{C}(\mathcal{P})$ are neighbors if C' can be reached from a point in C by crossing a single two-dimensional surface σ_R, for some $R \in \mathcal{Q}$. Recall that $\mathcal{C}(\mathcal{Q})$ is a refinement of $\mathcal{C}(\mathcal{P})$, and fix a cell $\tilde{C} \in \mathcal{C}(\mathcal{P})$. We claim that the

[10] This happens when $\overline{Front}(C)$ is unbounded and contains ℓ^+.

[11] This can be improved, by a more careful traversal of $\mathcal{A}$, but it will not affect the overall running time.

adjacency graph of all three-dimensional cells in the restriction of $\mathcal{C}(\mathcal{Q})$ to $\tilde{C}$ is connected. Indeed, let R be a polyhedron of $\mathcal{Q}$, and let σ_R be the corresponding surface of lines tangent to R. Pick a point p which belongs to the intersection of σ_R with the relative interior of $\tilde{C}$. Since p is contained in the relative interior of $\tilde{C} \in \mathcal{C}(\mathcal{P})$, the line ℓ represented by p is not tangent to any $P \in \mathcal{P}$. Hence, there is a pair of polyhedra $P, Q \in \mathcal{P}$ such that $R = P \cap Q$, and ℓ is tangent to R at the common intersection of the boundaries of P and Q. Thus, restricted to the interior of $\tilde{C}$, the surfaces σ_R, for $R \in \mathcal{Q}$ are in general position, and the corresponding adjacency graph is connected. Moreover, the algorithm of Theorem 2.2 can be easily modified to construct such a graph, for each $\tilde{C} \in \mathcal{C}(\mathcal{P})$, in $O(nk^5 \log^2 n)$ total time (in addition to the time required to constructed the cell-location structure in $\mathcal{C}(\mathcal{Q})$).

Let C and C' be a pair of adjacent three-dimensional cells in $\mathcal{C}(\mathcal{Q})$ which are contained in $\tilde{C} \in \mathcal{C}(\mathcal{P})$ as above. That is, C and C' are connected across a common face of the surface σ_R, for $R \in \mathcal{Q}$. Hence, there is a unique pair of polyhedra P and Q such that $\{R\} = \{P \cap Q\} = \mathcal{Q}_C \triangle \mathcal{Q}_{C'}$. Therefore, (P, Q) is the only pair of polyhedra that are related under $\prec_C$ and are 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 $q \in \bigcap \overline{Front}(C)$; see Figure 7 (a). In other words, we have argued[12] that $|\overline{Front}(C) \triangle \overline{Front}(C')| \leq 1$.

Let $\tilde{C}$ be a cell of $\mathcal{C}(\mathcal{P})$. We pick a representative cell $C_0 \in \mathcal{C}(\mathcal{Q})$ contained in $\tilde{C}$, and compute $\overline{Front}(C_0)$ in a brute-force manner, by picking a line $\ell \in C$ and computing, for each $P \in \mathcal{P}_C$, the segment $P \cap \ell$, (using two ray-shooting queries at P). Then $\overline{Front}(C)$ consists of every P which is minimal in the relation $\prec_C$, which we compute using the segments $P \cap \ell$. We next search, in $O(k \log n)$ time, the table of equivalence classes, with the bit-vector corresponding to $\overline{Front}(C)$.

Next we trace the previously constructed adjacency structure of $\mathcal{C}(\mathcal{Q})$ within $\tilde{C}$ in breadth-first fashion, say, and find all the cells $C \in \mathcal{C}(\mathcal{Q})$ contained in $\tilde{C}$. Recall that the algorithm of Theorem 2.2, that we use, also computes, for each pair of adjacent cells C and C', a line ℓ and a polyhedron R. Here R is the polyhedron in $\mathcal{Q}$ such that C and C' are connected across a two-dimensional face of $\mathcal{C}(\mathcal{Q})$ contained in σ_R, and ℓ is a line on the common boundary of C and C'. For each newly encountered cell C', reached via a previous cell C, we thus have the corresponding pair $P, Q \in \mathcal{P}$ such that $\{R\} = \{P \cap Q\} = \mathcal{Q}_C \triangle \mathcal{Q}_{C'}$. We can now compute $\overline{Front}(C')$ from $\overline{Front}(C)$ as follows. We use the 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 $\overline{Front}(C)$ (and we know which action to take); otherwise $\overline{Front}(C) = \overline{Front}(C')$.

To facilitate the transition from $\overline{Front}(C)$ to $\overline{Front}(C')$, we extend a table on the equivalence classes $\mathcal{T}$ that is constructed at the preliminary stage, as follows. For each class $\mathcal{T}$ and each polyhedron P we store a pointer to $\mathcal{T}' = \mathcal{T} \triangle \{P\}$. Since there are $O(nk^2)$ classes, each represented as a k-long bit-vector, this takes a total of $O(nk^4 \log k)$ time, and $O(nk^3)$ storage, well within the overall performance bounds of the algorithm. This completes the preprocessing stage.

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 $\mathcal{C}(\mathcal{Q})$. Each such transition takes $O(\log n)$ time. We similarly handle lower-dimensional cells of $\mathcal{C}(\mathcal{Q})$ and $\mathcal{C}(\mathcal{P})$. In summary, we have:

[12] This is not the case when we cross between different cells of $\mathcal{C}(\mathcal{P})$; then the change in $\overline{Front}(C)$ may be quite substantial.

Theorem 3.7. *Let $\mathcal{P}$ be a set of k possibly intersecting convex polyhedra with a total of n facets, and 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 per query. The structure uses $O(nk^5 \log^2 n)$ storage and preprocessing time.*

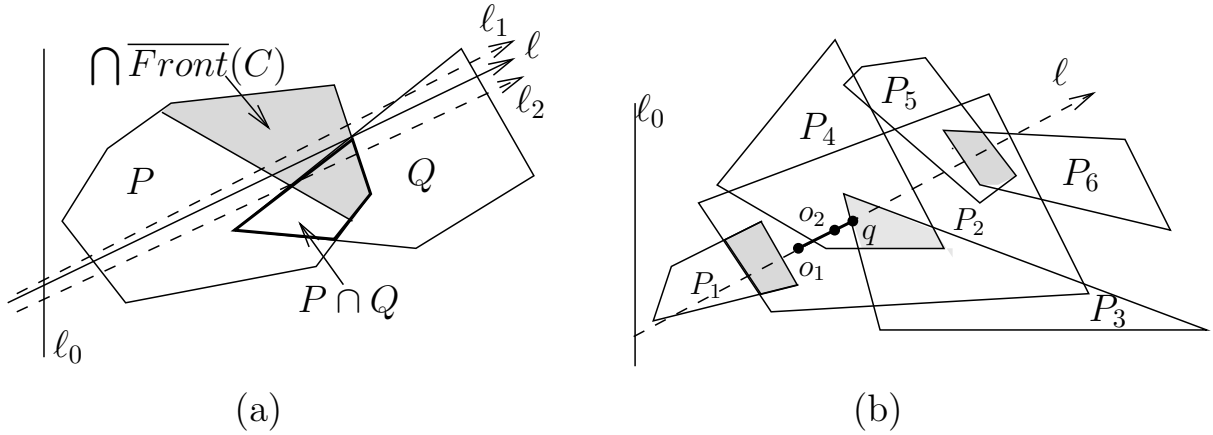

Figure 7: (a) Updating $\overline{Front}(C)$. The lines ℓ_1 and ℓ_2 belong to C and C', respectively. The line ℓ is on the common boundary of C and C'. (b) Shooting rays contained in lines through ℓ_0. For the cell C containing ℓ, we have $\Xi_1^C = \{P_1, P_2\}$, $\Xi_2^C = \{P_2, P_3, P_4\}$, and $\Xi_3^C = \{P_2, P_5, P_6\}$ (the corresponding polyhedra $\bigcap \Xi_1^C$, $\bigcap \Xi_2^C$, $\bigcap \Xi_3^C$ are shaded).

4 Extensions

In this section we consider several extensions of the preceding algorithm. First, we remove the assumption that the query rays originate on a fixed line ℓ_0. Now they are only required to be contained in lines intersecting ℓ_0. The next extension is to ray-shooting with rays orthogonal to a fixed line. Finally, we briefly describe a simple solution for vertical ray-shooting amidst k possibly intersecting convex polyhedra.

4.1 Ray-shooting involving rays contained in lines intersecting ℓ_0.

We use the notations of Section 3. Fix a cell C of $\mathcal{C}(\mathcal{Q})$, and let $\Xi_1^C, \ldots, \Xi_{t_C}^C$ be the maximal antichains under $\prec_C$. Observe that for any pair of indices $1 \leq i < j \leq t_C$, lines represented in C do not intersect the polyhedron $\left(\bigcap \Xi_i^C\right) \cap \left(\bigcap \Xi_j^C\right)$. To see this, consider the set of polyhedra $\Xi = \Xi_i^C \cup \Xi_j^C$. By the maximality of the antichains Ξ_i^C, Ξ_j^C, it follows that Ξ is not an antichain under $\prec_C$. Thus Ξ contains two polyhedra Q_1, Q_2 satisfying $Q_1 \prec_C Q_2$, which implies that $Q_1 \cap Q_2$ is not intersected by any line in C. Since $\bigcap \Xi$ is contained in $Q_1 \cap Q_2$, the claim follows.

We can thus define the $\prec_C$ relation also between the polyhedra of the form $\bigcap \Xi_i^C$, over the maximal antichains Ξ_i^C. The preceding argument implies that this is a *total* order, which we assume to be $\bigcap \Xi_1^C \prec_C \bigcap \Xi_2^C \cdots \prec_C \bigcap \Xi_{t_C}^C$.

Observe that each polyhedron $P \in \mathcal{P}_C$ belongs to maximal antichains in a contiguous interval of this list. Otherwise, there is a triple of indices $1 \leq i < j < h \leq t_C$ and a polyhedron $P \in P_C$, such that $P \in \Xi_i^C$, $P \in \Xi_h^C$, but P is not in Ξ_j^C. By the maximality condition, $\bigcap \Xi_i^C \subseteq P, \bigcap \Xi_h^C \subseteq P$, but $\bigcap \Xi_j^C \not\subseteq P$. Since all lines in C intersect $\bigcap \Xi_i^C$, $\bigcap \Xi_j^C$, and $\bigcap \Xi_h^C$ in the same order, this contradicts the convexity of P. It then follows that $t_C = O(k)$.

For each cell C in $\mathcal{C}(\mathcal{Q})$ and index $1 \leq i \leq t_C$, we maintain a data structure which supports ray-shooting queries at the polyhedron $\bigcap \Xi_i^C$. In addition, we construct and store a data structure which supports point location in the spatial arrangement $\mathcal{A} = \mathcal{A}(\mathcal{P})$ of the polyhedra of $\mathcal{P}$; see Section 3.

Ray Shooting. As above, we use parametric search, and therefore given a segment $s = o_1o_2$, contained in a line passing through ℓ_0, we need to determine whether s intersects the boundary of any polyhedron in $\mathcal{P}$. Extending the analysis in Lemma 3.4, we have:

Lemma 4.1. *Let ℓ be a line through ℓ_0, let ρ be a positive ray contained in ℓ which emanates from some point $o_1 \in \ell$ not contained in the boundary of any polyhedron of $\mathcal{P}$. Let C be the cell of $\mathcal{C}(\mathcal{Q})$ that contains ℓ, and let q be the first intersection of ρ with the boundary of some polyhedron $\bigcap \Xi_i^C$, for $1 \leq i \leq t_C$.*[13] *Then, for all points o_2 in the (closed) segment o_1q of ρ, the segment $s = o_1o_2$ intersects the boundary of some polyhedron of $\mathcal{P}$ if and only if o_1 and o_2 belong to distinct cells of $\mathcal{A}$.*

For an illustration, see Figure 7(b). Shooting with negative rays is analogous so we discuss only positive rays.

Proof. Clearly, if o_1 and o_2 belong to distinct cells of $\mathcal{A}$ then o_1o_2 intersects the boundary of a polyhedron of $\mathcal{P}$. For the converse implication, let $P \in \mathcal{P}$ be the first polyhedron that s intersects (i.e., the polyhedron whose boundary intersects s at a point closest to o_1). If one of the endpoints o_1, o_2 lies inside P then the other endpoint must lie outside P, so o_1 and o_2 belong to distinct cells of $\mathcal{A}$. (We exclude the case where both o_1 and o_2 lie on the boundary of the same polyhedron of $\mathcal{P}$.) Thus, assume that both o_1 and o_2 lie outside P, so they must lie on different sides of the interval $\ell \cap P$. By the discussion preceding the lemma, P participates in one or several anti-chains Ξ_i^C. Pick any such anti-chain Ξ_i^C. By Lemma 3.2, ℓ intersects $\bigcap \Xi_i^C$, and since $\bigcap \Xi_i^C \subseteq P$, it follows that o_1 and o_2 lie on different sides of the interval $\ell \cap \bigcap \Xi_i^C$, and thus s intersects $\bigcap \Xi_i^C$. But this contradicts the choice of q, and the fact that o_2 belongs to the segment o_1q. □

We now describe our query algorithm which tests, for a query segment $s = o_1o_2$ contained in a line passing through ℓ_0, whether s intersects a (non-artificial) facet on the boundary of some polyhedron of $\mathcal{P}$. First, we query the point location structure in $\mathcal{A}$, with the point o_1, to test whether o_1 belongs to the boundary of some polyhedron of $\mathcal{P}$ and report intersection, if the answer is positive. We next 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 run a binary search over the sorted list $\bigcap \Xi_1^C \prec_C \bigcap \Xi_2^C \prec_C \cdots \prec_C \bigcap \Xi_{t_C}^C$ with o_1. This takes $O(\log t_C) = O(\log k)$ steps, in each step of the search we test, for some $1 \leq i \leq t_C$, on which side of the polyhedron $\bigcap \Xi_i^C$ the point o_1 lies, or whether it is contained in $\bigcap \Xi_i^C$. Each of these steps is a ray shooting query at a convex polyhedron, which can be accomplished with logarithmic query time and linear storage. If o_1 lies to the right of all polyhedra $\bigcap \Xi_i^C$, for $1 \leq i \leq t_C$, we set q equal to infinity. Otherwise, we perform a ray-shooting query at the polyhedron $\bigcap \Xi_i^C$ which either contains o_1 or lies immediately to the right of o_1 along ℓ, to find q, the first intersection of ρ with the boundary of a polyhedron $\bigcap \Xi_i^C$, for some $1 \leq i \leq t_C$. If o_2 appears along ρ after q, we report intersection and stop. Otherwise, we query the data structure for point location in $\mathcal{A}$, with the point o_2, to test whether o_1 and o_2 belong to different cells of $\mathcal{A}$, and report intersection if and only if the answer is positive. The correctness of this procedure follows from Lemma 4.1.

[13] If ρ does not intersect the boundary of any polyhedron $\bigcap \Xi_j^C$, for $1 \leq j \leq t_C$, we say q is at infinity. (This happens, for instance, if ρ intersects the boundary of some polyhedron of $\Xi_{t_C}^C$ but does not intersect $\cap \Xi_{t_C}^C$.) In such cases, the segment o_1q is equal to ρ.

Query Time and Storage Complexity. Clearly, the query time for segment intersection detection is $O(\log n \log k + \log^2 n) = O(\log^2 n)$ so a ray-shooting query takes $O(\log^4 n)$ time, using parametric search, in a manner similar to that described in Section 3. That is, we first compute q explicitly, without any parametric search, since it is independent of o_2 (the only generic part of the input). This takes $O(\log^2 n)$ time, as described above. Then we search for the actual answer o_2 to the ray-shooting query, within the segment $o_1 q$ of ρ, by parametrically locating o_2 in the arrangement $\mathcal{A}$, as above. By the argument preceding Theorem 3.7, this requires $O(\log^4 n)$ time.

Consider next the storage complexity. As noted above, the point location data structure for $\mathcal{C}(\mathcal{Q})$ requires $O(nk^5 \log^2 n)$ storage (and preprocessing). To bound the storage needed for ray shooting structures amidst polyhedra $\bigcap \Xi_j^C$ we need the following lemma, which is an easy generalization of Lemma 3.5.

Lemma 4.2. *Let $\mathcal{T}$ be a maximal antichain under $\prec_C$. If $\ell \in C$ intersects $f \in \mathcal{A}$, such that f is on $\partial(\bigcap \mathcal{T})$, then f is good for $\mathcal{T}$.*

Proof. We have to prove that $\mathcal{P}_f = \mathcal{T}$. Since f is on $\partial(\bigcap \mathcal{T})$, it is clear that every polyhedron in $\mathcal{T}$ contains f. Conversely, let P be a polyhedron that contains f. We claim that P is unrelated to all the polyhedra in $\mathcal{T}$ under $\prec_C$. Indeed, if there is $Q \in \mathcal{T}$ such that $P \prec_C Q$ (or vice versa), then $P \cap Q$ is not stabbed by any line of C. However, f is intersected by $\ell \in C$ and f belongs to $P \cap Q$, a contradiction. Since $\mathcal{T}$ is a maximal antichain, P belongs to $\mathcal{T}$. □

To efficiently store the ray-shooting structures for the polyhedra $\bigcap \Xi_1^C, \bigcap \Xi_2^C, \ldots \bigcap \Xi_{t_C}^C$, over all cells $C \in \mathcal{C}(\mathcal{Q})$, we apply the approach of Section 3. That is, for each face f of $\mathcal{A}$, we compute $\mathcal{P}_f \subseteq \mathcal{P}$. Then we partition the collection of faces into corresponding equivalence classes, and, for each equivalence class $\mathcal{T}$ we construct a ray-shooting structure amidst the polyhedron $G(\mathcal{T})$. For each $\mathcal{T} \subseteq \mathcal{P}$ and each cell $C \in \mathcal{C}(\mathcal{Q})$ such that $\Xi_i^C = \mathcal{T}$, for some $1 \leq i \leq t_C$, we store with C a pointer to the ray-shooting structure at $G(\mathcal{T})$. By Lemma 4.2, ray shooting at $G(\mathcal{T})$ is equivalent to ray shooting at $\bigcap \Xi_j^C$, for rays on lines belonging to the cell C. Thus, the correctness analysis of Section 3 readily extends to the case at hand. As is easily checked, all the auxiliary ray-shooting structures require $O(nk^2)$ overall storage (and $O(nk^2 \log n)$ preprocessing).

Preprocessing. To complete the description of the preprocessing algorithm we need to show how to compute for each cell $C \in \mathcal{C}(\mathcal{Q})$ and each $1 \leq i \leq t_C$, the corresponding list of pointers to the ray-shooting data structures at $G(\Xi_1^C)$, $G(\Xi_2^C)$, …, $G(\Xi_{t_C}^C)$.

We use an algorithm similar to the one used in Section 3. Here is a brief description of the required modifications. Recall that $\mathcal{C}(\mathcal{Q})$ is a refinement of $\mathcal{C}(\mathcal{P})$. For each cell $\tilde{C} \in \mathcal{C}(\mathcal{P})$, the cells C of $\mathcal{C}(\mathcal{Q})$ contained in it, form a connected cluster. We pick any representative cell $C \in \mathcal{C}(\mathcal{Q})$ such that $C \subseteq \tilde{C}$, and construct the list $\Xi_1^C, \Xi_2^C, \ldots, \Xi_{t_C}^C$ in a brute-force manner, using $O(k)$ ray-shooting queries at the polyhedra of $\mathcal{P}$. This takes $O(k^2 \log k)$ time per cell of $\mathcal{C}(\mathcal{P})$, for a total of $O(nk^4 \log n)$ time. For each set Ξ_i^C, we locate the ray-shooting structure at $G(\Xi_i^C)$, in $O(k \log n)$ time, by searching the table of equivalence classes used by algorithm of Theorem 3.7, with a k-bit vector. This takes a total of $O(k^2 \log n)$ time per each representative cell $\tilde{C}$. Thus, we spend a total of $O(nk^4 \log n)$ time (over the $O(nk^2)$ representative cells $C \in \mathcal{C}(\mathcal{Q})$).

It suffices to describe, given two adjacent cells C and C' in $\mathcal{C}(\mathcal{Q})$ with the same $\mathcal{P}_C$, how to obtain the list of ray-shooting structures at $G(\Xi_1^{C'})$, $G(\Xi_2^{C'})$, …, $G(\Xi_{t_C}^{C'})$, from the corresponding list of C. We can assume, as above, that the symmetric difference between $\mathcal{Q}_C$ and $\mathcal{Q}_{C'}$ consists of a single intersection polyhedron $P \cap Q$, where $P, Q \in \mathcal{P}$.

Suppose first that $P \cap Q \in \mathcal{Q}_{C'} \setminus \mathcal{Q}_C$ and $P \prec_C Q$. Then there is an adjacent pair Ξ_i^C, Ξ_{i+1}^C of maximal antichains, such that $P \in \Xi_i^C$, and $Q \in \Xi_{i+1}^C$. The index i can be found by binary search. We then have to insert between Ξ_i^C and Ξ_{i+1}^C a maximal antichain of the form $\Xi = (\Xi_i^C \cap \Xi_{i+1}^C) \cup \{P, Q\}$. This antichain is constructed in a brute-force manner in time $O(k \log n)$,

using $O(k)$ ray-shooting queries against the polyhedra of $\mathcal{P}$. Then we find the ray-shooting structure against $G(\Xi)$ in $O(k \log n)$ time, by searching the above table of equivalence classes with a k-bit vector. In addition, one or both of Ξ_i^C, Ξ_{i+1}^C may become non-maximal under inclusion and should be removed. This can also be easily checked without increasing the running time of the algorithm. When $P \cap Q \in \mathcal{Q}_C \setminus \mathcal{Q}_{C'}$, we act symmetrically. Thus, the construction takes a total of $O(nk^6 \log n)$ time.

The lists of pointers to the ray-shooting structures at $G(\Xi_1^C)$, $G(\Xi_2^C)$, ..., $G(\Xi_{t_C}^C)$ are updated in a persistent manner, without increasing the total storage required by the data-structure. The following theorem summarizes the result which we obtained.

Theorem 4.3. *Let $\mathcal{P}$ be a set of k possibly intersecting convex polyhedra in $\mathbb{R}^3$ with a total of n facets. Then we can construct, in $O(nk^6 \log n)$ time, a data structure of size $O(nk^5 \log^2 n)$, that supports ray-shooting queries involving rays which are contained in lines intersecting ℓ_0, in $O(\log^4 n)$ time per ray.*

4.2 Ray-shooting with rays orthogonal to a fixed line.

Without loss of generality, assume the query rays are orthogonal to the z-axis. This problem can be considered as a special case of ray-shooting involving rays which are contained in lines intersecting ℓ_0. Indeed, we can place ℓ_0 at infinity, such that any plane which is parallel to the xy-plane contains ℓ_0. The algorithm of Theorem 3.7 can be symbolically modified to handle this degenerate placement of ℓ_0; for similar analysis see [8]. We thus obtain.

Theorem 4.4. *Let $\mathcal{P}$ be a set of k convex polyhedra in $\mathbb{R}^3$ having a total of n facets. Then we can construct, in $O(nk^6 \log n)$ time, a data structure of size $O(nk^5 \log^2 n)$, which supports ray-shooting queries involving rays orthogonal to the z-axis, in $O(\log^4 n)$ time per query.*

4.3 Vertical Ray-Shooting.

We describe an efficient data structure for vertical ray-shooting, in a collection $\mathcal{P}$ of k possibly intersecting polyhedra with a total of n facets.

For each polyhedron $P \in \mathcal{P}$, we raise unbounded vertical walls from the silhouette edges of P (namely, those edges e, for which the vertical plane containing e is tangent to P). These walls bound an infinite vertical prism which we denote by $P^\perp$. Let $\mathcal{A}^\perp = \mathcal{A}^\perp(\mathcal{P})$ be the resulting arrangement of $\mathcal{P} \bigcup \{P^\perp \mid P \in \mathcal{P}\}$. Clearly, $\mathcal{A}^\perp$ is xy-monotone and has complexity $O(nk^2)$. We construct a point location structure in $\mathcal{A}^\perp$, as described in [16], which requires $O(nk^2 \log^2 n)$ storage and preprocessing time, and answers queries in $O(\log^2 n)$ time.

For each cell a of $\mathcal{A}^\perp$, we project its top boundary onto the xy-plane, and preprocess each of the resulting planar maps for efficient point location. Altogether, these structures use $O(nk^2)$ storage and $O(nk^2 \log n)$ preprocessing time, and a query in any of them takes $O(\log n)$ time. To answer a vertical ray shooting query with a ray emanating upwards (say) from some point o, we locate o in $\mathcal{A}^\perp$, and then locate the xy-projection of o in the xy-projection of the top boundary of the cell containing o. This identifies the first polyhedron boundary hit by the query ray. The query time is $O(\log^2 n)$.

Theorem 4.5. *Let $\mathcal{P}$ be a set of k possibly intersecting convex polyhedra in $\mathbb{R}^3$ with a total of n facets. 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 vertical ray-shooting queries in $O(\log^2 n)$ time.*

5 Conclusion

We have considered several restricted instances of the ray-shooting problem amidst a set of k convex polyhedra in $\mathbb{R}^3$ with a total of n facets. We proposed data structures which require storage that is near-linear in n (and polynomial in k), and answer queries in polylogarithmic time. Our approach was based on decomposing the three-dimensional parametric space of the lines containing rays of the restricted class under consideration, and on using the point location structure of Preparata and Tamassia [16]. However, arbitrary lines in $\mathbb{R}^3$ have four degrees of freedom and are usually represented as points in $\mathbb{R}^4$ (or on the quadric Plücker surface in oriented projective 5-space). Moreover, the complexity of the decomposition of the four-dimensional space of lines in $\mathbb{R}^3$ into maximal connected regions, such that all lines in the same region stab the same subset of $\mathcal{P}$, is known to be $\Theta(n^2k^2)$; see [10]. Thus the approach of this paper does not seem directly applicable to the case of general ray-shooting, and leaves the problem of closing the gap between the bounds for $k = 1$ and for arbitrary $k \ll n$ as still open. In addition, we have not seriously attempted to reduce the dependence of the performance of our data structures on k. This too is an open problem for future research.

Finally, we have only considered the case when we want the query time to be polylogarithmic, and seek to optimize the storage and preprocessing costs. At the other end of the tradeoff, we want the storage to be nearly linear in n, and seek to optimize the query time. In the cases under consideration, the general techniques yield query time close to $O(n^{2/3})$. It would be interesting to refine this bound to make it depend also on k.

Acknowledgements We thank the anonymous referee for valuable suggestions that helped us to improve the presentation.

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] B. Aronov, M. Sharir and B. Tagansky, The union of convex polyhedra in three dimensions, *SIAM J. Comput.* 26 (1997), 1670–1688.

[3] P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* 22 (1993), 794–806.

[4] P. K. Agarwal and J. Matoušek, Range searching with semialgebraic sets, *Discrete Comput. Geom.* 11 (1994), 393–418.

[5] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions, *SIAM J. Comput.* 25 (1996), 100–116.

[6] B. Aronov, M. Pellegrini and M. Sharir, On the zone of a surface in a hyperplane arrangement, *Discrete Comput. Geom.* 9 (1993), 177–186.

[7] M. de Berg, *Ray Shooting, Depth Orders and Hidden Surface Removal*, volume 703 of Lect. Notes in Comput. Sci., Springer-Verlag, Berlin, 1993.

[8] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd Edition, Springer Verlag, Heidelberg, 2000.

[9] M. Bern, D. P. Dobkin, D. Eppstein, and R. Grossman, Visibility with a moving point of view, *Algorithmica* 11 (1994), 360–378.

[10] 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.

[11] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symb. Comput.* 7 (1989), 11–30.

[12] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach, *Proc. 17th Int. Colloq. Automata Languages and Programming* (1991), 400–413.

[13] J. Matoušek, *Lectures on Discrete Geometry*, Springer-Verlag New York, 2002.

[14] M. Pellegrini, Ray shooting on triangles in 3-space, *Algorithmica* 9 (1993), 471–494.

[15] 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.

[16] F. P. Preparata and R. Tamassia, Efficient point location in a convex spatial cell complex, *SIAM J. Comput.* 21 (1992), 267–280.