Optimal detection of intersections between convex polyhedra

Luis Barba*† Stefan Langerman*‡

Abstract

For a polyhedron P in $\mathbb{R}^d$, denote by $|P|$ its combinatorial complexity, i.e., the number of faces of all dimensions of the polyhedra. In this paper, we revisit the classic problem of preprocessing polyhedra independently so that given two preprocessed polyhedra P and Q in $\mathbb{R}^d$, each translated and rotated, their intersection can be tested rapidly.

For $d = 3$ we show how to perform such a test in $O(\log |P| + \log |Q|)$ time after linear preprocessing time and space. This running time is the best possible and improves upon the last best known query time of $O(\log |P| \log |Q|)$ by Dobkin and Kirkpatrick (1990).

We then generalize our method to any constant dimension d, achieving the same optimal $O(\log |P| + \log |Q|)$ query time using a representation of size $O(|P|^{\lfloor d/2 \rfloor + \varepsilon})$ for any $\varepsilon > 0$ arbitrarily small. This answers an even older question posed by Dobkin and Kirkpatrick 30 years ago.

In addition, we provide an alternative $O(\log |P| + \log |Q|)$ algorithm to test the intersection of two convex polygons P and Q in the plane.

1 Introduction

Constructing or detecting the intersection between geometric objects is probably one of the first and most important applications of computational geometry. It was one of the main questions addressed in Shamos' seminal paper that lay the grounds of computational geometry [21], the first application of the plane sweep technique [22], and is still the topic of several volumes being published today.

It is hard to overstate the importance of finding efficient algorithms for intersection testing or collision detection as this class of problems has countless applications in motion planning, robotics, computer graphics, Computer-Aided Design, VLSI design and more. For information on collision detection refer to surveys [14, 16] and to Chapter 38 of the Handbook of Computational Geometry [13].

The first problem to be addressed is to compute the intersection of two convex objects. In this paper we focus on convex polygons and convex polyhedra (or simply polyhedra). Let P and Q be two polyhedra to be tested for intersection. Let $|P|$ and $|Q|$ denote the combinatorial complexities of P and Q, respectively, i.e., the number of faces of all dimensions of the polygon or polyhedra (vertices are 0-dimensional faces while edges are 1-dimensional faces). Let $n = |P| + |Q|$ denote the total complexity.

In the plane, Shamos [21] presented an optimal $\Theta(n)$-time algorithm to construct the intersection of a pair of convex polygons. Another linear time algorithm was later presented by O'Rourke et al. [20]. In 3D space, Muller and Preparata [19] proposed an $O(n \log n)$ time algorithm to test whether two polyhedra in three-dimensional space intersect. Their algorithm has a second phase which computes the intersection of these polyhedra within the same running time using geometric dualization. Dobkin and Kirpatrick [8] introduced a hierarchical data structure to represent a polyhedra that allows them to test if two polyhedra intersect in linear time. In a subsequent

*Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium {lbarbafl,slanger}@ulb.ac.be
†School of Computer Science, Carleton University, Ottawa, Canada
‡Directeur de recherches du F.R.S.-FNRS.

paper, Chazelle [2] presented an optimal linear time algorithm to compute the intersection of two polyhedra in 3D-space.

A natural extension of this problem is to consider the effect of preprocessing on the complexity of intersection detection problems. In this case, significant improvements are possible in the query time. It is worth noting that each object should be preprocessed separately which allows us to work with large families of objects and to introduce new objects without triggering a reconstruction of the whole structure.

Chazelle and Dobkin [3, 4] were the first to formally define and study this class of problems and provided an algorithm running in $O(\log |P| + \log |Q|)$ time to test the intersection of two convex polygons P and Q in the plane. An alternate solution was given by Dobkin and Kirkpatrick [7] with the same running time. Edelsbrunner [10] then used that algorithm as a preprocessing phase to find the closest pair of points between two convex polygons, within the same running time. Dobkin and Souvaine [9] extended these algorithms to test the intersection of two convex planar regions with piecewise curved boundaries of bounded degree in logarithmic time. These separation algorithms rely on an involved case analysis to solve the problem. In Section 2, we show an alternate (and hopefully simpler) algorithm to determine if two convex polygons P and Q intersect in $O(\log |P| + \log |Q|)$ time.

In all these 2D algorithms, preprocessing is unnecessary if the polygon is represented by an array with the vertices of the polygon in sorted order along its boundary. In 3D-space (and in higher dimensions) however, the need for preprocessing is more evident as the traditional DCEL representation of the polyhedron is not sufficient to perform fast queries.

In this setting, Chazelle and Dobkin [4] presented a method to preprocess a 3D polyhedron and use this structure to test if two preprocessed polyhedra intersect in $O(\log^3 n)$ time. Dobkin and Kirkpatrick [7] unified and extended these results, showing how to detect if two independently preprocessed polyhedra intersect in $O(\log^2 n)$ time. Both methods represent a polyhedron P by storing parallel slices of P through each of its vertices, and thus require $O(|P|^2)$ time, although space usage could be reduced using persistent data structures.

In 1990, Dobkin and Kirkpatrick [6] proposed a fast query algorithm that uses the linear space hierarchical representation of a polyhedron P defined in their previous article [8]. Using this structure, they show how to determine in $O(\log |P| \log |Q|)$ time if the polyhedra P and Q intersect. They achieve this by maintaining the closest pair between subsets of the polyhedra P and Q as the algorithms walks down the hierarchical representation. Unfortunately, the paper seems to have omitted an important case which would cause a naive implementation to take time $\Omega(|P| + |Q|)$ rather than the claimed bound. However, by combining different results from the same article, it seems the $O(\log |P| \log |Q|)$ bound could be salvaged [15]. In Section 4, we detail the specific problem with the algorithm, and in Section 4 we show a simple modification of the data structure that overcomes this issue and restores all bounds claimed in that article.

Whether the intersection of two preprocessed polyhedra P and Q can be tested in $O(\log |P| + \log |Q|)$ time is an open question that was implicit in the paper of Chazelle and Dobkin [3] in STOC'80, and explicitly posed in 1983 by Dobkin and Kirkpatrick [7]. More recently, the open problem was listed again in 2004 by David Mount in Chapter 38 of the Handbook of Computational Geometry [13]. Together with this question in 3D-space, Dobkin and Kirkpatrick [7] asked if it is possible to extend these result to higher dimensions, i.e., to independently preprocess two polyhedra in $\mathbb{R}^d$ such that their intersection could be tested in $O(\log n)$ time.

These running times are best possible as, even in the plane, testing if a point intersects a regular m-gon M has a lower bound of $\Omega(\log m)$ in the algebraic decision tree model.

In this paper, we match this lower bound by showing how to independently preprocess polyhedra P and Q in any bounded dimension such that their intersection can be tested in $O(\log n)$ time[1]. In Section 4, we show how to preprocess a polyhedron $P \in \mathbb{R}^3$ in linear time to obtain a linear space representation. In Section 5 we provide an algorithm that, given any translation and rotation of two preprocessed polyhedra P and Q in $\mathbb{R}^3$, tests if they intersect in $O(\log |P| + \log |Q|)$ time. In

[1] In this paper, all algorithms are in the real RAM model of computation.

Section 6 we generalize our results to any constant dimension d and show a representation that allows to test if two polyhedra P and Q in $\mathbb{R}^d$ (rotated and translated) intersect in $O(\log |P| + \log |Q|)$ time. The space required by the representation of a polyhedron P is then $O(|P|^{\lfloor d/2 \rfloor + \varepsilon})$ for any small $\varepsilon > 0$. This increase in the space requirements for $d \geq 4$ is not unexpected as the problem studied here is at least as hard as performing halfspace emptiness queries for a set of m points in $\mathbb{R}^d$. For this problem, the best known log-query data structures use roughly $O(m^{\lfloor d/2 \rfloor})$ space [17], and super-linear space lower bounds are known for $d \geq 5$ [12].

2 Algorithm in the plane

Let P and Q be two convex polygons in the plane with n and m vertices, respectively. We assume that a convex polygon is given as an array with the sequence of its vertices sorted in clockwise order along its boundary. Let $V(P)$ and $E(P)$ be the set of vertices and edges of P, respectively. Let ∂P denote the boundary of P. Analogous definitions apply for Q. As a warm-up, we describe an algorithm to determine if P and Q intersect whose running time is $O(\log n + \log m)$. Even though algorithms with these running time already exists in the literature, they require an involved case analysis whereas our approach avoids them and is arguably easier to implement. Moreover, it provides some intuition for the higher-dimension algorithms presented in subsequent sections.

For each edge $e \in E(Q)$, its *supporting halfplane* is the halfplane containing Q supported by the line extending e. Given a subset of edges $F \subseteq E(Q)$, the *edge hull* of F is the intersection of the supporting halfplanes of each of the edges in F. Throughout the algorithm, we consider a triangle $\mathcal{T}_P$ being the convex hull of three vertices of P and a triangle (possibly unbounded) $\mathcal{T}_Q$ defined as the edge hull of three edges of Q; see Figure 1 for an illustration. Notice that $\mathcal{T}_P \subseteq P$ while $Q \subseteq \mathcal{T}_Q$.

Intuitively, in each round the algorithm compares $\mathcal{T}_P$ and $\mathcal{T}_Q$ for intersection and, depending on the output, prunes a fraction either of the vertices of P or of the edges of Q. Then, the triangles $\mathcal{T}_P$ and $\mathcal{T}_Q$ are redefined should there be a subsequent round of the algorithm.

Let $V^*(P)$ and $E^*(Q)$ respectively be the sets of vertices and edges of P and Q remaining after the pruning steps performed so far by the algorithm. Initially, $V^*(P) = V(P)$ while $E^*(Q) = E(Q)$. After each pruning step, we maintain the *correctness invariant* which states that an intersection between P and Q can be computed with the remaining vertices and edges after the pruning. That is, P and Q intersect if and only if $\textsc{ch}(V^*(P))$ intersects an edge of $E^*(Q)$, where $\textsc{ch}(V^*(P))$ denotes the convex hull of $V^*(P)$.

For a given polygonal chain, its *vertex-median* is a vertex whose removal splits this chain into two pieces that differ by at most one vertex. In the same way, the *edge-median* of this chain is the edge whose removal splits the chain into two parts that differ by at most one edge.

The 2D algorithm

To begin with, define $\mathcal{T}_P$ as the convex hull of three vertices whose removal splits the boundary of P into three chains, each with at most $\lceil (n-3)/3 \rceil$ vertices. In a similar way, define $\mathcal{T}_Q$ as the edge hull of three edges of Q that split its boundary into three polygonal chains each with at most $\lceil (m-3)/3 \rceil$ edges; see Figure 1.

A line *separates* two convex polygons if they lie in opposite closed halfplanes supported by this line. After each round of the algorithm, we maintain one of the two following invariants: The *separation invariant* states that we have a line ℓ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ such that ℓ is tangent to $\mathcal{T}_P$ at a vertex v. The *intersection invariant* states that we have a point in the intersection between $\mathcal{T}_P$ and $\mathcal{T}_Q$. Note that at least one of among separation and the intersection invariant must hold, and they only hold at the same time when $\mathcal{T}_P$ is tangent to $\mathcal{T}_Q$. The algorithm performs two different tasks depending on which of the two invariants holds (if both hold, we choose a task arbitrarily).

Separation invariant.

If the separation invariant holds, then there is a line ℓ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ such that ℓ is tangent to $\mathcal{T}_P$ at a vertex v. Let ℓ^- be the closed halfplane supported by ℓ that contains $\mathcal{T}_P$ and let ℓ^+ be its complement.

Consider the two neighbors n_v and n'_v of v along the boundary of P. Because P is a convex polygon, if both n_v and n'_v lie in ℓ^-, then we are done as ℓ separates P from $\mathcal{T}_Q \supseteq Q$. Otherwise, by the convexity of P, either n_v or n'_v lies in ℓ^+ but not both. Assume without loss of generality that $n_v \in \ell^+$ and notice that the removal of the vertices of $\mathcal{T}_P$ split ∂P into three polygonal chains. In this case, we know that only one of these chains, say c_v, intersects ℓ^+. Moreover, we know that v is an endpoint of c_v and we denote its other endpoint by u.

Because Q is contained in ℓ^+, only the vertices in c_v can define an intersection with Q. Therefore, we prune $V^*(P)$ by removing every vertex of P that does not lie on c_v and maintain the correctness invariant. We redefine $\mathcal{T}_P$ as the convex hull of v, u and the vertex-median of c_v. With the new $\mathcal{T}_P$, we can test in $O(1)$ time if $\mathcal{T}_P$ and $\mathcal{T}_Q$ intersect. If they do not, then we can compute a new line that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$ and preserve the separation invariant. Otherwise, if $\mathcal{T}_P$ and $\mathcal{T}_Q$ intersect, then we establish the intersection invariant and proceed to the next round of the algorithm.

Intersection invariant.

If the intersection invariant holds, then $\mathcal{T}_P \cap \mathcal{T}_Q \neq \emptyset$. In this case, let e_1, e_2 and e_3 be the three edges whose edge hull defines $\mathcal{T}_Q$. Notice that if $\mathcal{T}_P \subseteq P$ intersects $\text{CH}(e_1, e_2, e_3) \subseteq Q$, then P and Q intersect and the algorithm finishes. Otherwise, there are three disjoint connected components in $\mathcal{T}_Q \setminus \text{CH}(e_1, e_2, e_3)$ and $\mathcal{T}_P$ intersects exactly one of them; see Figure 1. Assume without loss of generality that $\mathcal{T}_P$ intersects the component bounded by the lines extending e_1 and e_2 and let x be a point on the boundary of $\mathcal{T}_Q$ in this intersection. Let C be the polygonal chain that connects e_1 with e_2 along ∂Q such that C passes through e_3. We claim that to test if P and Q intersect, we need only to consider the edges on $\partial Q \setminus C$. To prove this claim, notice that if P intersects C at a point y, then the edge xy is contained in Q. Because x and y lie in two disjoint connected components of $\mathcal{T}_Q \setminus \text{CH}(e_1, e_2, e_3)$, the edge xy also intersects ∂Q at another point lying on $\partial Q \setminus C$. Therefore, an intersection between P and Q will still be identified even if we ignore every edge on C. That is, P and Q intersect if and only if P and $\partial Q \setminus C$ intersect. Thus, we can prune $E^*(Q)$ by removing every edge along C while preserving the correctness invariant. After the pruning step, we redefine $\mathcal{T}_Q$ as the edge hull of e_1, e_2 and the edge-median of the remaining edges of $E(Q)$ after the pruning.

If $\mathcal{T}_P$ intersects $\mathcal{T}_Q$ after being redefined, then the intersection invariant is preserved an we proceed to the next round of the algorithm. Otherwise, if $\mathcal{T}_P$ does not intersect $\mathcal{T}_Q$, then we we can compute in $O(1)$ time a line ℓ tangent to $\mathcal{T}_P$ that separates $\mathcal{T}_P$ from $\mathcal{T}_Q$. That is, the separation invariant is reestablished should there be a subsequent round of the algorithm.

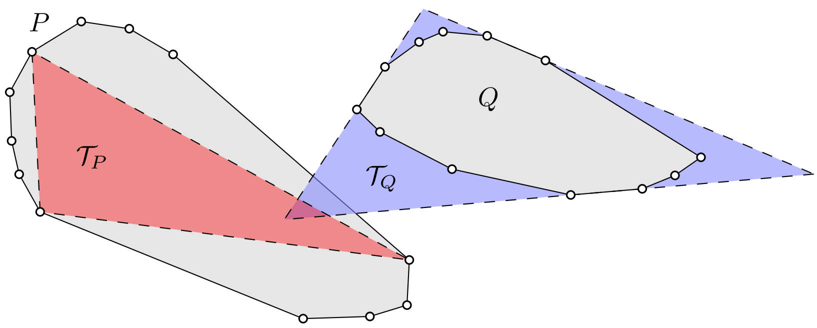

Figure 1: Two convex polygons P and Q and the triangles $\mathcal{T}_P$ and $\mathcal{T}_Q$ such that $\mathcal{T}_Q \subseteq P$ and $Q \subseteq \mathcal{T}_Q$. Moreover, $\mathcal{T}_Q \setminus Q$ consists of three connected components.

Theorem 2.1. *Let P and Q be two convex polygons with n and m vertices, respectively. The 2D-algorithm determines if P and Q intersect in $O(\log n + \log m)$ time.*

Proof. Each time we redefine $\mathcal{T}_P$, we take three vertices that split the remaining vertices of $V^*(P)$ into two chains of roughly equal length along ∂P. Therefore, after each round where the separation invariant holds, we prune a constant fraction of the vertices of $V^*(P)$. That is, the separation invariant step of the algorithm can be performed at most $O(\log n)$ times.

Each time $\mathcal{T}_Q$ is redefined, we take three edges that split the remaining edges along the boundary of Q into equal pieces. Thus, we prune a constant fraction of the edges of $E^*(Q)$ after each round where the intersection invariant holds. Hence, this can be done at most $O(\log m)$ times before being left with only three edges of Q. Furthermore, the correctness invariant is maintained after each of the pruning steps.

Thus, if the algorithm does not find a separating line or an intersection point, then after $O(\log n + \log m)$ steps, $\mathcal{T}_P$ consists of the only three vertices left in $V^*(P)$ while $\mathcal{T}_Q$ consist of the only three edges remaining from $E^*(Q)$. If e_1, e_2 and e_3 are the edges whose edge hull defines $\mathcal{T}_Q$, then by the correctness invariant we know that P and Q intersect if and only if $\mathcal{T}_P$ intersects either e_1, e_2 or e_3. Consequently, we can test them for intersection in $O(1)$ time and determine if P and Q intersect. ☐

3 The polar transformation

Let $\mathbb{0}$ be the *origin* of $\mathbb{R}^d$, i.e., the point with d coordinates equal to zero. Throughout this paper, a *hyperplane* h is a $(d-1)$-dimensional affine space in $\mathbb{R}^d$ such that for some $z \in \mathbb{R}^d$, $h = \{x \in \mathbb{R}^d : \langle z, x \rangle = 1\}$, where $\langle *, * \rangle$ represents the interior product of Euclidean spaces. Therefore, in this paper a hyperplane does not contain the origin. A *halfspace* is the closure of either of the two parts into which a hyperplane divides $\mathbb{R}^d$, i.e., a halfspace contains the hyperplane defining its boundary.

Given a point $x \in \mathbb{R}^d$, we define its *polar* to be the hyperplane $\rho(x) = \{y \in \mathbb{R}^d : \langle x, y \rangle = 1\}$. Given a hyperplane h in $\mathbb{R}^d$, we define its *polar* $\rho(h)$ as the point $z \in \mathbb{R}^d$ such that $h = \{y \in \mathbb{R}^d : \langle z, y \rangle = 1\}$. Let $\rho_{\mathbb{0}}(x) = \{y \in \mathbb{R}^d : \langle x, y \rangle \leq 1\}$ and $\rho_\infty(x) = \{y \in \mathbb{R}^d : \langle x, y \rangle \geq 1\}$ be the two halfspaces supported by $\rho(x)$, where $\mathbb{0} \in \rho_{\mathbb{0}}(x)$ while $\mathbb{0} \notin \rho_\infty(x)$. In the same way, $h_{\mathbb{0}}$ and h_∞ denote the halfspaces supported by h such that $\mathbb{0} \in h_{\mathbb{0}}$ while $\mathbb{0} \notin h_\infty$.

Note that the polar of a point $x \in \mathbb{R}^d$ is a hyperplane whose polar is equal to x, i.e., the polar operation is self-inverse (for more information on this transformation see Section 2.3 of [23]). Given a set of points (or hyperplanes), its *polar set* is the set containing the polar of each of its elements. The following result is illustrated in Figure 2(a).

Lemma 3.1. *Let x and h be a point and a hyperplane in $\mathbb{R}^d$, respectively. Then, $x \in h_{\mathbb{0}}$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x)$. Also, $x \in h_\infty$ if and only if $\rho(h) \in \rho_\infty(x)$. Moreover, $x \in h$ if and only if $\rho(h) \in \rho(x)$.*

Proof. Recall that $h_{\mathbb{0}} = \{y \in \mathbb{R}^d : \langle y, \rho(h) \rangle \leq 1\}$. Then, $x \in h_{\mathbb{0}}$ if and only if $\langle x, \rho(h) \rangle \leq 1$. Furthermore, $\langle x, \rho(h) \rangle \leq 1$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x) = \{y \in \mathbb{R}^d : \langle y, x \rangle \leq 1\}$. That is, $x \in h_{\mathbb{0}}$ if and only if $\rho(h) \in \rho_{\mathbb{0}}(x)$. Analogous proofs hold for the other statements. ☐

A polyhedron is a convex region in the d-dimensional space being the non-empty intersection of a finite set of halfspaces. Given a set of hyperplanes S in $\mathbb{R}^d$, let $\text{PH}_\infty[S] = \cap_{h \in S} h_\infty$ and $\text{PH}_{\mathbb{0}}[S] = \cap_{h \in S} h_{\mathbb{0}}$ be two polyhedra defined by S. Let $P \subset \mathbb{R}^d$ be a polyhedron. Let $V(P)$ denote the set of vertices of P and let $S(P)$ be the set of hyperplanes that extend the $(d-1)$-dimensional faces of P. Therefore, if P is bounded, then it can be seen as the convex hull of $V(P)$, denoted by $\text{CH}(V(P))$. Moreover, if P contains the origin, then P can be also seen as $\text{PH}_{\mathbb{0}}[S(P)]$.

To *polarize* P, let $\mathbb{S}(P)$ be the polar set of $V(P)$, i.e., the set of hyperplanes being the polars of the vertices of P. Therefore, we can think of $\text{PH}_{\mathbb{0}}[\mathbb{S}(P)]$ and $\text{PH}_\infty[\mathbb{S}(P)]$ as the possible polarizations

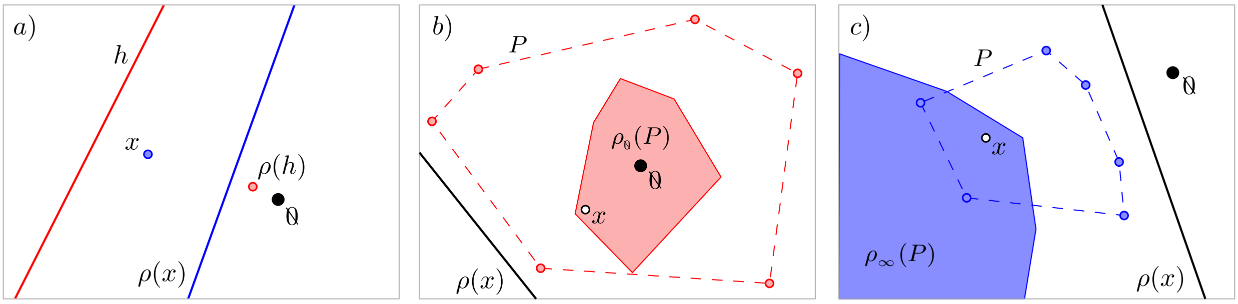

Figure 2: *a)* The situation described in Lemma 3.1. *b)* A polygon P containing the origin and its polarization $\rho_{\mathbb{0}}(P)$. The first statement of Lemma 3.3 is depicted. *c)* A polygon P that does not contains the origin and its polarization $\rho_\infty(P)$. The second statement of Lemma 3.3 is also depicted.

of P. For ease of notation, we let $\rho_{\mathbb{0}}(P)$ and $\rho_\infty(P)$ denote the polyhedra $\text{PH}_{\mathbb{0}}[\mathbb{S}(P)]$ and $\text{PH}_\infty[\mathbb{S}(P)]$, respectively. Note that P contains the origin if and only if $\rho_\infty(P) = \emptyset$ and $\rho_{\mathbb{0}}(P)$ is bounded.

Lemma 3.2. *(Clause (v) of Theorem 2.11 of [23]) Let P be a polyhedron in $\mathbb{R}^d$ such that $\mathbb{0} \in P$. Then, $\rho_{\mathbb{0}}(\rho_{\mathbb{0}}(P)) = P$.*

As a consequence of Lemma 3.1 we obtain the following result depicted in Figures 2(*b*) and 2(*c*).

Lemma 3.3. *Let P be a polyhedron in $\mathbb{R}^d$ and let $x \in \mathbb{R}^d$. Then, $x \in \rho_{\mathbb{0}}(P)$ if and only if $P \subseteq \rho_{\mathbb{0}}(x)$. Moreover, $x \in \rho_\infty(P)$ if and only if $P \subseteq \rho_\infty(x)$.*

Proof. Let x be a point in $\rho_{\mathbb{0}}(P)$. Notice that for every hyperplane $s \in \mathbb{S}(P)$, $x \in s_{\mathbb{0}}$. Therefore, by Lemma 3.1 we know that the vertex $\rho(s) \in V(P)$ lies in $\rho_{\mathbb{0}}(x)$. Consequently, every vertex of P lies in $\rho_{\mathbb{0}}(x)$, i.e., $P \subseteq \rho_{\mathbb{0}}(x)$.

On the other direction, let v be a vertex of P, i.e., $\rho(v) \in \mathbb{S}(P)$. If $v \in \rho_{\mathbb{0}}(x)$, then by Lemma 3.1 $x \in \rho_{\mathbb{0}}(v)$. Therefore, for every $\rho(v) \in \mathbb{S}(P)$, we know that $x \in \rho_{\mathbb{0}}(v)$, i.e., $x \in \rho_{\mathbb{0}}(P)$.

The same proof holds for the second statement by replacing all instances of $\mathbb{0}$ by ∞. □

In the case that $\mathbb{0} \in P$, $\rho_\infty(P)$ is empty and the second conclusion of the previous lemma holds trivially. Thus, even though the previous result is always true, it is non-trivial only when $\mathbb{0} \notin P$.

Lemma 3.4. *Let P be a polyhedron in $\mathbb{R}^d$. If $x \in P$, then $\rho_{\mathbb{0}}(P) \subseteq \rho_{\mathbb{0}}(x)$ while $\rho_\infty(P) \subseteq \rho_\infty(x)$.*

Proof. Assume for a contradiction that there is a point $y \in \rho_{\mathbb{0}}(P)$ such that $y \notin \rho_{\mathbb{0}}(x)$. Therefore, by Lemma 3.1 we know that $x \notin \rho_{\mathbb{0}}(y)$. Moreover, because $y \in \rho_{\mathbb{0}}(P)$, Lemma 3.3 implies that $P \subseteq \rho_{\mathbb{0}}(y)$—a contradiction with the fact that $x \in P$ and $x \notin \rho_{\mathbb{0}}(y)$. An analogous proof holds to show that $\rho_\infty(P) \subseteq \rho_\infty(x)$. □

Note that the converse of Lemma 3.4 is not necessarily true.

Lemma 3.5. *Let P be a polyhedron in $\mathbb{R}^d$ and let γ be a hyperplane. If γ is either tangent to $\rho_{\mathbb{0}}(P)$ or to $\rho_\infty(P)$, then $\rho(\gamma)$ is a point lying on the boundary of P.*

Proof. Let γ be a hyperplane tangent to $\rho_{\mathbb{0}}(P)$ at a vertex v. Because $v \in \gamma$, Lemma 3.1 implies that $\rho(\gamma) \in \rho(v)$. We claim that $\rho(\gamma) \in P$. Assume for a contradiction that $\rho(\gamma) \notin P$. Since $v \in \rho_{\mathbb{0}}(P)$, we know that $P \subseteq \rho_{\mathbb{0}}(v)$ by Lemma 3.3. Therefore, because $\rho(\gamma) \in \rho(v)$ and from the assumption that $\rho(\gamma) \notin P$, we can slightly perturb $\rho(v)$ to obtain a hyperplane h such that $P \subseteq h_{\mathbb{0}}$ while $\rho(\gamma)$ lies in the interior of h_∞. Thus, since $\rho(\gamma) \in h_\infty$ while $\rho(\gamma) \notin h$, Lemma 3.1 implies that $\rho(h)$ lies in the interior of γ_∞. Moreover, because $P \subseteq h_{\mathbb{0}}$ we know by Lemma 3.3 that $\rho(h) \in \rho_{\mathbb{0}}(P)$. Therefore, there is a point of $\rho_{\mathbb{0}}(P)$, say $\rho(h)$, that lies in the interior of γ_∞—a contradiction with the fact that γ is tangent to $\rho_{\mathbb{0}}(P)$. Therefore, $\rho(\gamma) \in P$. Moreover, because $\rho(\gamma) \in \rho(v)$ and from the fact that $P \subseteq \rho_{\mathbb{0}}(v)$, $\rho(\gamma)$ cannot lie in the interior of P, i.e, $\rho(\gamma)$ lies on the boundary of P. An analogous proof holds for the case when γ is tangent to $\rho_\infty(P)$. □

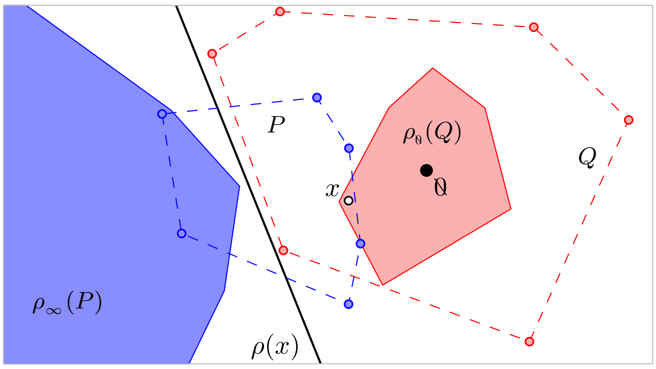

Figure 3: The statement of Theorem 3.7 where a point x lies in the intersection of P and $\rho_{\mathbb{0}}(Q)$ if and only if $\rho(x)$ separates Q from $\rho_\infty(P)$.

Lemma 3.6. *Let P and Q be two polyhedra. If $P \subseteq Q$, then $\rho_{\mathbb{0}}(Q) \subseteq \rho_{\mathbb{0}}(P)$ and $\rho_\infty(Q) \subseteq \rho_\infty(P)$.*

Proof. Let $x \in \rho_{\mathbb{0}}(Q)$. Then, Lemma 3.3 implies that $Q \subseteq \rho_{\mathbb{0}}(x)$. Because we assumed that $P \subseteq Q$, $P \subseteq \rho_{\mathbb{0}}(x)$. Therefore, we infer from Lemma 3.3 that $x \in \rho_{\mathbb{0}}(P)$. That is, $\rho_{\mathbb{0}}(Q) \subseteq \rho_{\mathbb{0}}(P)$. An analogous proof holds to show that $\rho_\infty(Q) \subseteq \rho_\infty(P)$. □

A hyperplane π *separates* two geometric objects in $\mathbb{R}^d$ if they are contained in opposite halfspaces supported by π, note that both objects can contain points lying on π. We obtain the main result of this section illustrated in Figure 3.

Theorem 3.7. *Let P and Q be two polyhedra. The polyhedra P and $\rho_{\mathbb{0}}(Q)$ intersect if and only if there is a hyperplane that separates $\rho_\infty(P)$ from Q. Also, (1) if $x \in P \cap \rho_{\mathbb{0}}(Q)$, then $\rho(x)$ separates $\rho_\infty(P)$ from Q, and (2) if γ is a hyperplane that separates $\rho_\infty(P)$ from Q such that γ is tangent to $\rho_\infty(P)$, then $\rho(\gamma) \in P \cap \rho_{\mathbb{0}}(Q)$. Moreover, the symmetric statements of (1) and (2) hold if we replace all instances of P (*resp. ∞*) by Q (*resp. $\mathbb{0}$*) and vice versa.*

Proof. Let x be a point in $P \cap \rho_{\mathbb{0}}(Q)$. Because $x \in P$, by Lemma 3.4 we know that $\rho_\infty(P) \subseteq \rho_\infty(x)$. Moreover, since $x \in \rho_{\mathbb{0}}(Q)$, by Lemma 3.3, $Q \subseteq \rho_{\mathbb{0}}(x)$. Therefore, $\rho(x)$ is a hyperplane that separates $\rho_\infty(P)$ from Q.

In the other direction, let γ' be a hyperplane that separates $\rho_\infty(P)$ from Q. Then, there is a hyperplane γ parallel to γ' that separates $\rho_\infty(P)$ from Q such that γ is tangent to $\rho_\infty(P)$. Therefore, $\rho(\gamma)$ is a point on the boundary of P by Lemma 3.5. Because $\rho(\gamma) \in P$, Lemma 3.4 implies that $\rho_{\mathbb{0}}(P) \subseteq \gamma_{\mathbb{0}}$ while $\rho_\infty(P) \subseteq \gamma_\infty$. Because γ separates $\rho_\infty(P)$ from Q and from the fact that $\rho_\infty(P) \subseteq \gamma_\infty$, we conclude that $Q \subseteq \gamma_{\mathbb{0}}$. Consequently, by Lemma 3.3 $\rho(\gamma) \in \rho_{\mathbb{0}}(Q)$. That is, $\rho(\gamma)$ is a point in the intersection of P and $\rho_{\mathbb{0}}(Q)$. The symmetric statements have analogous proofs. □

Notice that if $\mathbb{0} \in P$, then P and $\rho_{\mathbb{0}}(Q)$ trivially intersect. Moreover, $\rho_\infty(P) = \emptyset$ implying that every hyperplane trivially separates $\rho_\infty(P)$ from Q. Therefore, while being always true, this result is non-trivial only when $\mathbb{0} \notin P$.

4 Polyhedra in 3D space

In this section, we focus on polyhedra in $\mathbb{R}^3$. Therefore, we can consider the 1-skeleton of a polyhedron being the planar graph connecting its vertices through the edges of the polyhedron.

Given a polyhedron P, a sequence $P_1, P_2, \ldots, P_k$ is a DK-hierarchy of P if the following properties hold [8].

*A*1. $P_1 = P$ and P_k a tetrahedron.

*A*2. $P_{i+1} \subseteq P_i$, for $1 \leq i \leq k$.

A3. $V(P_{i+1}) \subseteq V(P_i)$, for $1 \leq i \leq k$.

A4. The vertices of $V(P_i) \setminus V(P_{i+1})$ form an independent set in P_i, for $1 \leq i < k$.

A5. The *height* of the hierarchy $k = O(\log n)$, $\sum_{i=1}^{k} V(P_i) = O(n)$.

Given a polyhedron P on n vertices, a set $I \subseteq V(P)$ is a *P-independent set* if (1) $|I| \geq n/10$, (2) I forms an independent set in the 1-skeleton of P and (3) the degree of every vertex in I is $O(1)$.

Dobkin and Kirkpatrick [8] showed how to construct a DK-hierarchy. This construction was later improved by Biedl and Wilkinson [1]. Formally, they start by defining $P_1 = P$. Then, given a polyhedron P_i, they show how to compute a P_i-independent set I and define P_{i+1} as the convex hull of the set $V(P_i) \setminus I$.

Using this data structure, they claimed to have an algorithm that computes the distance between two preprocessed polyhedra in $O(\log^2 n)$ time [6]. Unfortunately as we show below, a slight oversight in their paper could cause a straightforward implementation of their algorithm to be much slower than this claimed bound.

In our algorithm, as well as in the algorithm presented by Dobkin and Kirpatrick [6], we are given a plane tangent to P_i at a vertex v and want to find a vertex of P_{i-1} lying on the other side of this plane (if it exists). Although they showed that at most one vertex of P_{i-1} can lie on the other side of this plane and that it has to be adjacent to v, they do not explain how to find such a vertex. An exhaustive walk through the neighbors of v in P_{i-1} would only be fast enough for their algorithm if v is always of constant degree. Unfortunately this is not always the case as shown in the following example.

Start with a tetrahedron P_k and select a vertex q of P_k. To construct the polyhedron P_{i-1} from P_i, we refine it by adding a vertex slightly above each face adjacent to q. In this way, the degree of the new vertices is exactly three. After k steps, we reach a polyhedron $P_1 = P$. In this way, the sequence $P = P_1, P_2, \ldots, P_k$ defines a DK-hierarchy of P. Moreover, when going from P_i to P_{i-1}, a new neighbor of q is added for each of its adjacent faces in P_i. Thus, the degree of q doubles when going from P_i to P_{i-1} and hence, the degree of q in P_1 is linear. Note that this situation can occur at a deeper level of the hierarchy, even if every vertex of P has degree three.

This issue seems to have been overlooked in the paper [6]. A naive implementation of their algorithm could then cause a query to take $\Omega(n)$ time instead of the $O(\log^2 n)$ bound claimed. We solve this problem by bounding the degree of each vertex in every polyhedron of the DK-hierarchy.

Bounded hierarchies

Let c be a fixed constant. We say that a polyhedron is *c-bounded* if at most c faces of this polyhedron can meet at a vertex, i.e., the degree of each vertex in its 1-skeleton is bounded by c.

Given a polyhedron P with n vertices, we describe a method to modify the structure of Dobkin and Kirkpatrick to construct a DK-hierarchy where every polyhedron other than P is c-bounded. As a starting point, we can assume that the faces of P are in general position (i.e., no four planes of $S(P)$ go through a single point) by using Simulation of Simplicity [11]. This implies that every vertex of P has degree three. To avoid having vertices of large degree in the hierarchy, we introduce the following operation. Given a vertex $v \in V(P)$ of degree $k > 3$, consider a plane π that separates v from every other vertex of P. Let $e_1, e_2, \ldots, e_k$ be the edges of P incident to v. For each $1 \leq i \leq k$, let v_i be the intersections of e_i with π. *Split* the edge e_i at v_i to obtain a new polyhedron with k more vertices and k new edges; for an illustration see Figure 4 (a) and (b).

To construct a c-bounded DK-hierarchy (or simply BDK-hierarchy), we start by letting $P_1 = P$. Given a polyhedron P_i in this BDK-hierarchy, let I be a P_i-independent set. Compute the convex hull of $V(P_i) \setminus I$, two cases arise: **Case 1.** If $\text{CH}(V(P_i) \setminus I)$ has no vertex of degree larger than c, then let $P_{i+1} = \text{CH}(V(P_i) \setminus I)$. **Case 2.** Otherwise, let W be the set of vertices of P_i with degree larger than c. For each vertex of W, split its adjacent edges as described above and let P_{i+1} be the obtained polyhedron. Notice that P_{i+1} is a polyhedron with the same number of faces than P_i. Moreover, because each edge of P_i may be split for each of its endpoints, P_{i+1} has at most three

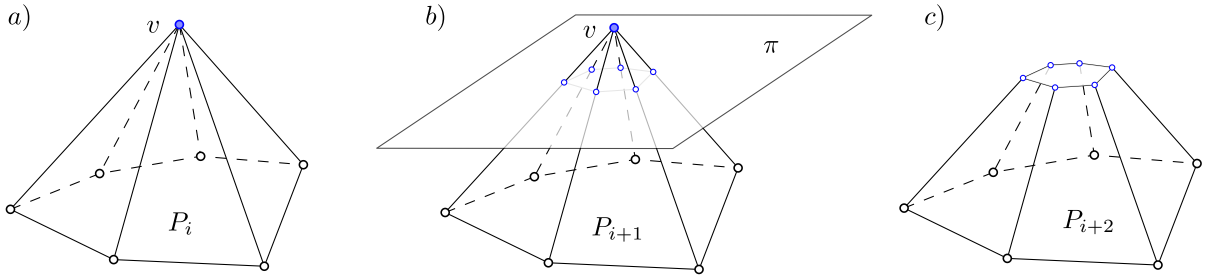

Figure 4: A polyhedron P and a vertex v of large degree. A plane π that separates v from $V(P) \setminus \{v\}$ is used to split the edges adjacent to v. New vertices are added to split these edges. Finally, the removal of v from the polyhedron leaves every one of its neighbors with degree three while adding a new face.

times the number the edges of P_i. Therefore $|V(P_{i+1})| \leq (2/3)|E(P_{i+1}) \leq 2|E(P_i)| \leq 6|V(P_i)|$ by Euler's formula.

Because each vertex of W is adjacent only to new vertices added during the split of its adjacent edges, the vertices in W form an independent set in the 1-skeleton of P_{i+1}. In this case, we let P_{i+2} be the convex hull of $V(P_{i+1}) \setminus W$. Therefore, (1) every vertex of P_{i+2} has degree three, and (2) the vertices in $V(P_{i+1}) \setminus V(P_{i+2})$ form an independent set in P_{i+1}; see Figure 4(c). Note that P_{i+1} and P_{i+2} have new vertices added during the splits. However, we know that $|V(P_{i+2})| \leq |V(P_{i+1})| \leq 6|V(P_i)|$. Furthermore, we also know that $P_{i+2} \subseteq P_{i+1} \subseteq P_i$.

We claim that by choosing c carefully, we can guarantee that the depth of the BDK-hierarchy is $O(\log n)$. To prove this claim, notice that after a pruning step, the degree of a vertex can increase at most by the total degree of its neighbors that have been eliminated. Let v be a vertex with the largest degree in P_i. Note that its neighbors can also have at most degree $\delta(v)$, where $\delta(v)$ denotes the number of neighbors of v in P_i. Therefore, after removing a P_i-independent set, the degree of v can be at most $\delta(v)^2$ in P_{i+1}. That is, the maximum degree of P_i can be at most squared when going from P_i to P_{i+1}.

Therefore, if we assume Case 2 has just been applied and that every vertex vertex of P_i has degree three, then after r rounds of Case 1, the maximum degree of any vertex is at most 3^{2^r}. Therefore, the degree of any of its vertices can go above c only after $\log_2(\log_3 c)$ rounds, i.e., we go through Case 1 at least $\log_2(\log_3 c)$ times before running into Case 2.

Since we removed at least 1/10-th of the vertices after each iteration of Case 1 [1], after $\log_2(\log_3 c)$ rounds the size of the current polyhedron is at most $(9/10)^{\log_2(\log_3 c)}|P_i|$. At this point, we run into Case 2 and add extra vertices to the polyhedron. However, by choosing c sufficiently large, we guarantee that the number of remaining vertices is at most $6 \cdot (9/10)^{\log_2(\log_3 c)}|P_i| < \alpha|P_i|$ for some constant $0 < \alpha < 1$. That is, after $\log_2(\log_3 c)$ rounds the size of the polyhedron decreases by constant factor implying a logarithmic depth. We obtain the following result.

Lemma 4.1. *Given a polyhedron P, the previous algorithm constructs a BDK-hierarchy $P_1, P_2, \ldots, P_k$ with following properties.*

- $B1$. *$P_1 = P$ and P_k is a tetrahedron.*
- $B2$. *$P_{i+1} \subseteq P_i$, for $1 \leq i \leq k$.*
- $B3$. *The polyhedron P_i is c-bounded, for $1 \leq i \leq k$.*
- $B4$. *The vertices of $V(P_i) \setminus V(P_{i+1})$ form an independent set in P_i, for $1 \leq i < k$.*
- $B5$. *The* height *of the hierarchy $k = O(\log n)$, $\sum_{i=1}^{k} V(P_i) = O(n)$.*

By bounding the degree of each vertex on every vertex of the BDK-hierarchy by a constant, we offer a solution to the issue in the algorithm presented in [6].

The following property of a DK-hierarchy of P was proved in [6] and is easily extended to BDK-hierarchies because its proof does not use property $A3$. Note that all properties of DK and BDK hierarchies are identical except for $B3 \neq A3$.

Lemma 4.2. *Let $P_1, \ldots, P_k$ be a BDK-hierarchy of a polyhedron P and let H be a plane defining two halfspaces H^+ and H^-. For any $1 \leq i \leq k$ such that P_{i+1} is contained in H^+, either $P_i \subseteq H^+$ or there exists a unique vertex $v \in V(P_i)$ such that $v \in H^- \setminus H$.*

5 Detecting intersections in 3D

In this section, we show how to independently preprocess polyhedra in 3D-space so that their intersection can be tested in logarithmic time.

Preprocessing

Let P be a polyhedron in $\mathbb{R}^3$. Assume without loss of generality that the origin lies in the interior of P. Otherwise, modify the coordinate system. To preprocess P, we first compute the polyhedron $\rho_{\mathbb{0}}(P)$ being the polarization of P. Then, we independently compute two BDK-hierarchies as described in Section 4, one for P and one for $\rho_{\mathbb{0}}(P)$. Recall that in the construction of BDK-hierarchies, we assume that the faces of the polyhedra being processed are in general position using Simulation of Simplicity [11]. Assuming that both P and $\rho_{\mathbb{0}}(P)$ have vertices in general position at the same time is not possible. However, this is not a problem as only one of the two BDK-hierarchies will ever be used in a single query. Therefore, we can independently use Simulation of Simplicity [11] on each of them.

Preliminaries of the algorithm

Let P and R be two independently preprocessed polyhedra with combinatorial complexities n and m, respectively. Throughout this algorithm, we fix the coordinate system used in the preprocessing of R, i.e., $\mathbb{0} \in R$. For ease of notation, let $Q = \rho_{\mathbb{0}}(R)$. Because $\mathbb{0} \in R$, Lemma 3.2 implies that $R = \rho_{\mathbb{0}}(Q)$.

The algorithm described in this section tests if P and $R = \rho_{\mathbb{0}}(Q)$ intersect. Therefore, we can assume that P and $\rho_{\mathbb{0}}(Q)$ lie in a *primal space* while $\rho_\infty(P)$ and Q lie in a *polar space*. That is, we look at the primal and polar spaces independently and switch between them whenever necessary. To test the intersection of P and $\rho_{\mathbb{0}}(Q)$ in the primal space, we use the BDK-hierarchies of P and Q stored in the preprocessing step. In an intersection query, we are given arbitrary translations and rotations for P and $\rho_{\mathbb{0}}(Q)$ and we want to decide if they intersect. Note that this is equivalent to answering the query when only a translation and rotation of P is given and $\rho_{\mathbb{0}}(Q)$ remains unchanged. This is important as we fixed the position of the origin inside $R = \rho_{\mathbb{0}}(Q)$. The idea of the algorithm is to proceed by rounds and in each of them, move down in one of the two hierarchies while maintaining some invariants. In the end, when reaching the bottom of the hierarchy, we determine if P and $\rho_{\mathbb{0}}(Q)$ are separated or not.

Let k and l be the depths of the hierarchies of P and Q, respectively. We use indices $1 \leq i \leq k$ and $1 \leq j \leq l$ to indicate our position in the hierarchies of P and Q. The idea is to decrement at least one of them in each round of the algorithm.

To maintain constant time operations, instead of considering a full polyhedron P_i in the BDK-hierarchy of P, we consider constant complexity polyhedra $P_i^* \subseteq P_i$ and $Q_j^* \subseteq Q_j$. Intuitively, both P_i^* and Q_j^* are constant size polyhedra that respectively represent the portions of P_i and Q_j that need to be considered to test for an intersection.

We also maintain a special point p^* in the primal space which is a vertex of both P_i^* and P_i, and a plane φ whose properties will be determined later. In the polar space, we keep a point q^* being a vertex of both Q_j^* and Q_j and a plane γ.

For ease of notation, given a polyhedron T and a vertex $v \in V(T)$, let $T \setminus v$ denote the convex hull of $V(T) \setminus \{v\}$. The *star invariant* consists of two parts, one in the primal and another in the polar space. In the primal space, this invariant states that if $i < k$, then (1) the plane φ separates $P_i \setminus p^*$ from $\rho_{\mathbb{0}}(Q_j)$ and (2) $\rho(\varphi) \in Q_j$. In the polar space, the star invariant states if $j < l$, then

(1) the plane γ separates $Q_j \setminus q^*$ from $\rho_\infty(P_i)$ and (2) $\rho(\gamma) \in P_i$. Whenever the star invariant is established, we store references to φ and γ, and to the vertices p^* and q^*.

Other invariants are also considered throughout the algorithm. The *separation invariant* states that we have a plane π that separates P_i from $\rho_\emptyset(Q_j^*)$ such that π is tangent to P_i at one of its vertices. The *inverse separation invariant* states that there is a plane μ that separates $\rho_\infty(P_i^*)$ from Q_j such that μ is tangent to Q_j at one of its vertices.

Before stepping into the algorithm, we need a couple of definitions. Given a polyhedron T and a vertex $v \in V(T)$, let $N_v(T)$ be a polyhedron defined as the convex hull of v and its neighbors in T. Let $\kappa_v(T)$ be the convex hull of the set of rays apexed at v shooting from v to each of its neighbors in T. That is, $\kappa_v(T)$ is a convex cone with apex v that contains T and has complexity $O(\delta(v))$, where $\delta(v)$ denotes the number of neighbors of v in T. We say that $\kappa_v(T)$ *separates* T from another polyhedra if the latter does not intersect the interior of $\kappa_v(T)$.

The algorithm

To begin the algorithm, let $i = k$ and $j = l$, i.e., we start with $P_i^* = P_i$ and $Q_j^* = Q_j$ being both tetrahedra. Notice that for the base case, $i = k$ and $j = l$, we can determine in $O(1)$ time if P_i and $\rho_\emptyset(Q_j) = \rho_\emptyset(Q_j^*)$ intersect. If they do not, then we can compute a plane separating them and establish the separation invariant. Otherwise, if P_i and $\rho_\emptyset(Q_j)$ intersect, then by Theorem 3.7 we know that $\rho_\infty(P_i) = \rho_\infty(P_i^*)$ do not intersect Q_j. Thus, in constant time we can compute a plane tangent to Q_j in the polar space that separates $\rho_\infty(P_i) = \rho_\infty(P_i^*)$ from Q_j. That is, we can establish the inverse separation invariant. Thus, at the beginning of the algorithm the star invariant holds trivially, and either the separation invariant or the inverse separation invariant holds (maybe both if P_i and $\rho_\emptyset(Q_j)$ are tangent).

After each round of the algorithm, we advance in at least one of the hierarchies of P and Q while maintaining the star invariant. Moreover, we maintain at least one among the separation and the inverse separation invariants. Depending on which invariant is maintained, we step into the primal or the polar space as follows (if both invariants hold, we choose arbitrarily).

A walk in the primal space.

We step into this case if the separation invariant holds. That is, P_i is separated from $\rho_\emptyset(Q_j^*)$ by a plane π tangent to P_i at a vertex v.

We know by Lemma 4.2 that there is at most one vertex p in P_{i-1} that lies in $\pi_\emptyset \setminus \pi$. Moreover, this vertex must be a neighbor of v in P_{i-1}. Because P_{i-1} is c-bounded, we scan the $O(1)$ neighbors of v and test if any of them lies in $\pi_\emptyset \setminus \pi$. Two cases arise:

Case 1. If P_{i-1} is contained in π_∞, then π still separates P_{i-1} from $\rho_\emptyset(Q_j^*)$ while being tangent to the same vertex v of P_{i-1}. Therefore, we have moved down one level in the hierarchy of P while maintaining the separation invariant.

To maintain the star invariant, let $P_{i-1}^* = N_v(P_{i-1})$ and let $p^* = v \in V(P_{i-1}^*) \cap V(P_{i-1})$. Because P_{i-1} is c-bounded, we know that P_{i-1}^* has constant size. Since $\rho_\emptyset(Q_j^*)$ has constant size, we can compute the plane φ parallel to π and tangent to $\rho_\emptyset(Q_j^*)$ in $O(1)$ time, i.e., φ also separates P_{i-1} from $\rho_\emptyset(Q_j^*)$. Because $\rho_\emptyset(Q_j^*) \supseteq \rho_\emptyset(Q_j)$ by Lemma 3.6 and from the fact that $P_{i-1} \setminus p^* \subset P_{i-1}$, we conclude that (1) φ separates $P_{i-1} \setminus p^*$ from $\rho_\emptyset(Q_j)$. Moreover, because $\rho(\varphi) \in Q_j^*$ by Lemma 3.5 and from the fact that $Q_j^* \subseteq Q_j$, we conclude that (2) $\rho(\varphi) \in Q_j$. Thus, the star invariant is maintained in the primal space.

In the polar space, if $j < l$, then since $\rho_\infty(P_{i-1}) \subseteq \rho_\infty(P_i)$ by Lemma 3.6, (1) the plane γ that separates $Q_j \setminus q^*$ from $\rho_\infty(P_i)$ also separates $Q_j \setminus q^*$ from $\rho_\infty(P_{i-1})$. Moreover, because $P_i \subseteq P_{i-1}$ and from the fact that $\rho(\gamma) \in P_i$, we conclude that (2) $\rho(\gamma) \in P_{i-1}$. Thus, the star invariant is also maintained in the polar space and we proceed with a new round of the algorithm in the primal space.

Case 2. If P_{i-1} crosses π, then by Lemma 4.2 there is a unique vertex p of P_{i-1} that lies in $\pi_\emptyset \setminus \pi$. To maintain the star invariant, let $P^*_{i-1} = N_p(P_{i-1})$ and let $p^* = p$. Then, proceed as in to the first case. In this way, we maintain the star invariant in both the primal and the polar space.

Recall that $\kappa_{p^*}(P_{i-1})$ is the cone being the convex hull of the set of rays shooting from p^* to each of its neighbors in P_{i-1}. Since P_{i-1} is c-bounded, p^* has at most c neighbors in P_{i-1}. Thus, both $\kappa_{p^*}(P_{i-1})$ and $\rho_\emptyset(Q^*_j)$ have constant complexity and we can test if they intersect in constant time. Two cases arise:

Case 2.1. If $\kappa_{p^*}(P_{i-1})$ and $\rho_\emptyset(Q^*_j)$ do not intersect, then as $P_{i-1} \subseteq \kappa_{p^*}(P_{i-1})$, we can compute in constant time a plane π' tangent to $\kappa_{p^*}(P_{i-1})$ at p^* that separates $P_{i-1} \subseteq \kappa_{p^*}(P_{i-1})$ from $\rho_\emptyset(Q^*_j)$. That is, we reestablish the separation invariant and proceed with a new round in the primal space.

Case 2.2. Otherwise, if $\kappa_{p^*}(P_{i-1})$ and $\rho_\emptyset(Q^*_j)$ intersect, then because $P_{i-1} \setminus p^* \subseteq \pi_\infty$ and $\rho_\emptyset(Q^*_j) \subseteq \pi_\emptyset$, we know that this intersection happens at a point of P^*_{i-1}, i.e., P^*_{i-1} intersects $\rho_\emptyset(Q^*_j)$. Therefore, by Theorem 3.7 there is a plane μ' that separates $\rho_\infty(P^*_{i-1})$ from Q^*_j in the polar space. In this case, we would like to establish the inverse separation invariant which states that $\rho_\infty(P^*_{i-1})$ is separated from Q_j. Note that if $j = l$, then $Q_j = Q^*_j$ and the inverse separation invariant is established. Therefore, assume that $j < l$ and recall that $q^* \in V(Q^*_j) \cap V(Q_j)$.

By the star invariant and from the assumption that $j < l$, the plane γ separates $Q_j \setminus q^*$ from $\rho_\infty(P_{i-1})$, i.e., $Q_j \setminus q^* \subseteq \gamma_\emptyset$. In this case, we refine P^*_{i-1} by adding the vertex $\rho(\gamma)$ to it, i.e., we let $P^*_{i-1} = \textsc{ch}(N_p(P_{i-1}) \cup \{\rho(\gamma)\})$. Note that this refinement preserves the star invariant as p^* is still a vertex of the refined P^*_{i-1}. Moreover, because $\rho(\gamma) \in P_{i-1}$ by the star invariant, we know that $P^*_{i-1} \subseteq P_{i-1}$.

Because $\rho(\gamma) \in P^*_{i-1}$, Lemma 3.4 implies that $\rho_\infty(P^*_{i-1}) \subseteq \gamma_\infty$. Since $Q_j \setminus q^* \subseteq \gamma_\emptyset$, γ separates $\rho_\infty(P^*_{i-1})$ from $Q_j \setminus q^*$. Because μ' separates $\rho_\infty(P^*_{i-1})$ from $Q^*_j \supseteq N_{q^*}(Q_j)$, we conclude that there is a plane that separates $\rho_\infty(P^*_{i-1})$ from Q_j and it only remains to compute it in $O(1)$ time.

In fact, because $Q_j \setminus q^* \subseteq \gamma_\emptyset$, all neighbors of q^* in Q_j lie in $\gamma_\emptyset$ and hence, the cone $\kappa_{q^*}(Q_j)$ does not intersect $\rho_\infty(P^*_{i-1})$. Since $\kappa_{q^*}(Q_j)$ and $\rho_\infty(P^*_{i-1})$ have constant complexity, we can compute a plane μ tangent to $\kappa_{q^*}(Q_j)$ at q^* such that μ separates $\kappa_{q^*}(Q_j)$ from $\rho_\infty(P^*_{i-1})$. Because $Q_j \subseteq \kappa_{q^*}(Q_j)$, μ separates Q_j from $\rho_\infty(P^*_{i-1})$ while being tangent to Q_j at q^*. That is, we establish the inverse separation invariant. In this case, we step into the polar space and try to move down in the hierarchy of Q in the next round of the algorithm.

A walk in the polar space.

We step into this case if the inverse separation invariant holds. That is, we have a plane tangent to Q_j at one of its vertices that separates $\rho_\infty(P^*_i)$ from Q_j. In this case, we perform an analogous procedure to that described for the case when the separation invariant holds. However, all instances of P_i (*resp.* P) are replaced by Q_j (*resp.* Q) and vice versa, and all instances of $\rho_\infty(*)$ are replaced by $\rho_\emptyset(*)$ and vice versa. Moreover, all instances of the separation and the inverse separation invariant are also swapped. At the end of this procedure, we decrease the value of j and establish either the separation or the inverse separation invariant. Moreover, the star invariant is also preserved should there be a subsequent round of the algorithm.

Analysis of the algorithm

After going back and forth between the primal and the polar space, we reach the bottom of the hierarchy of either P or Q. Thus, we might reach a situation in which we analyze P_1 and $\rho_\emptyset(Q^*_j)$ in the primal space for some $1 \leq j \leq l$. In this case, if the separation invariant holds, then we have computed a plane π that separates P_1 from $\rho_\emptyset(Q^*_j) \supseteq \rho_\emptyset(Q)$. Because $P = P_1$, we conclude that π separates P from $R = \rho_\emptyset(Q)$.

We may also reach a situation in which we test Q_1 and $\rho_\infty(P^*_i)$ in the polar space for some $1 \leq i \leq k$. In this case, if the inverse separation invariant holds, then we have a plane μ that separates Q_1 from $\rho_\infty(P_i^*)$. Since $\rho_\infty(P_i^*)$ has constant complexity, we can assume that μ is tangent to $\rho_\infty(P_i^*)$ as we can compute a plane parallel to μ with this property. Because $Q = Q_1$, we conclude that μ is a plane that separates Q from $\rho_\infty(P_i^*)$ such that μ is tangent to $\rho_\infty(P_i^*)$. Therefore, Theorem 3.7 implies that $\rho(\mu)$ is a point in the intersection of $P_i^* \subseteq P$ and $\rho_{\mathbb{0}}(Q)$, i.e., P and $R = \rho_{\mathbb{0}}(Q)$ intersect.

In any other situation the algorithm can continue until one of the two previously mentioned cases arises and the algorithm finishes. Because we advance in each round in either the BDK-hierarchy of P or the BDK-hierarchy of Q, after $O(\log n + \log m)$ rounds the algorithm finishes. Because each round is performed in $O(1)$ time, we obtain the following result.

Theorem 5.1. *Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^3$ with combinatorial complexities n and m, respectively. For any given translations and rotations of P and R, we can determine if P and R intersect in $O(\log n + \log m)$ time.*

6 Detecting intersections in higher dimensions

In this section, we extend our algorithm to any constant dimension d at the expense of increasing the space to $O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$. To do that, we replace the BDK-hierarchy and introduce a new hierarchy produced by recursively taking ε-nets of the faces of the polyhedron. Our objective is to obtain a new hierarchy with logarithmic depth with properties similar to those described in Lemma 4.2. For the latter, we use the following definition.

Given a polyhedron P, the intersection of $(d+1)$ halfspaces is a *shell-simplex* of P if it contains P and each of these $(d+1)$ halfspaces is supported by a $(d-1)$-dimensional face of P.

Lemma 6.1. *Let P be a polyhedron in $\mathbb{R}^d$ with k vertices. We can compute a set $\Sigma(P)$ of at most $O(k^{\lfloor d/2 \rfloor})$ shell-simplices of P such that given a hyperplane π tangent to P, there is a shell-simplex $\sigma \in \Sigma(P)$ such that π is also tangent to σ.*

Proof. Without loss of generality assume that $\mathbb{0} \in P$. Note that $\rho_{\mathbb{0}}(P)$ has exactly k $(d-1)$-dimensional faces. Using Lemma 3.8 of [5] we infer that there exists a triangulation T of $\rho_{\mathbb{0}}(P)$ such that the combinatorial complexity of T is $O(k^{\lfloor d/2 \rfloor})$. That is, T decomposes $\rho_{\mathbb{0}}(P)$ into interior disjoint d-dimensional simplices.

Let s be a simplex of T. For each $v \in V(s)$, notice that since $v \in \rho_{\mathbb{0}}(P)$, $P \subseteq \rho_{\mathbb{0}}(v)$ by Lemma 3.4. Therefore, $P \subseteq \cap_{v \in V(s)} \rho_{\mathbb{0}}(v) = \rho_{\mathbb{0}}(s)$, i.e., $\sigma_s = \rho_{\mathbb{0}}(s)$ is a shell-simplex of P obtained from polarizing s. Finally, let $\Sigma(P) = \{\sigma_s : s \in T\}$ and notice that $|\Sigma(P)| = O(k^{\lfloor d/2 \rfloor})$.

Because $\mathbb{0} \in P$, Lemma 3.2 implies that $P = \rho_{\mathbb{0}}(\rho_{\mathbb{0}}(P))$. Let π be a hyperplane tangent to $P = \rho_{\mathbb{0}}(\rho_{\mathbb{0}}(P))$ and note that its polar is a point $\rho(\pi)$ lying on the boundary of $\rho_{\mathbb{0}}(P)$ by Lemma 3.5. Hence, $\rho(\pi)$ lies on the boundary of a simplex s of T. Thus, by Lemma 3.4 we know that $\sigma_s \subseteq \pi_{\mathbb{0}}$. Because $\rho(\pi)$ lies on the boundary of s, π is tangent to σ_s yielding our result. ☐

Hierarchical trees

Let P be a polyhedron with combinatorial complexity n. We can assume that the vertices of P are in general position (i.e., no $d+1$ vertices lie on the same hyperplane) using Simulation of Simplicity [11].

Let $F(P)$ be the set of all faces of P. Consider the family G such that a set $g \in G$ is the complement of the intersection of $d+1$ halfspaces. Let $F_g = \{f \in F(P) : f \cap g \neq \emptyset\}$ be the set of faces of P induced by g. Let $G_{F(P)} = \{F_g : g \in G\}$ be the family of subsets of $F(P)$ induced by G.

To compute the hierarchy of P, let $0 < \varepsilon < 1$ and consider the range space defined by $F(P)$ and $G_{F(P)}$. Since the VC-dimension of this range space is finite, we can compute an ε-net N of $(F(P), G_{F(P)})$ of size $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) = O(1)$ [18]. Because the vertices of P are in general position, each face of P has at most $d+1$ vertices. Therefore, $\textsc{ch}(N)$ has $O(|N|)$ vertices, i.e., $\textsc{ch}(N)$ has constant complexity. By Lemma 6.1 and since $|N| = O(1)$, we can compute the set $\Sigma(\textsc{ch}(N))$ having $O(|N|^{\lfloor d/2 \rfloor})$ shell-simplices of $\textsc{ch}(N)$ in constant time.

Given a shell-simplex $\sigma \in \Sigma(\textsc{ch}(N))$, let $\bar{\sigma} \in G$ be the complement of σ. Because $\textsc{ch}(N) \subseteq \sigma$, $\bar{\sigma}$ intersects no face of N. Recall that $F_{\bar{\sigma}} = \{f \in F(P) : f \cap \bar{\sigma} \neq \emptyset\}$. Therefore, since N is an ε-net of $(F(P), G_{F(P)})$, we conclude that $F_{\bar{\sigma}}$ contains at most $\varepsilon|F(P)|$ faces of P.

We construct the *hierarchical tree* of a polyhedron P recursively. In each recursive step, we consider a subset F of the faces of P as input. As a starting point, let $F = F(P)$. The recursive construction considers two cases: (1) If F consists of a constant number of faces, then its tree consists of a unique node storing a reference to $\textsc{ch}(F)$. (2) Otherwise, compute the ε-net N of F as described above and store $\textsc{ch}(N)$ together with $\Sigma(\textsc{ch}(N))$ at the root node. Then, for each shell-simplex $\sigma \in \Sigma(\textsc{ch}(N))$ construct recursively the tree for $F_{\bar{\sigma}}$ and attach it to the root node. Because the size of the ε-net is independent of the size of the polyhedron, we obtain a hierarchical structure being a tree rooted at $\textsc{ch}(N)$ with maximum degree $O(|N|^{\lfloor d/2 \rfloor})$.

Lemma 6.2. *Given a polyhedron P in $\mathbb{R}^d$ with combinatorial complexity n and any $\delta > 0$, we can compute a hierarchical tree for P with $O(\log n)$ depth in $O(n^{\lfloor d/2 \rfloor + \delta})$ time using $O(n^{\lfloor d/2 \rfloor + \delta})$ space.*

Proof. Because we reduce the number of faces of the original polyhedron by a factor of ε on each branching of the hierarchical tree, the depth of this tree is $O(\log n)$.

The space $S(n)$ of this hierarchical tree of P can be described by the following recurrence $S(n) = O(|N|^{\lfloor d/2 \rfloor})S(\varepsilon n) + O(1)$. Recall that $|N| = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. Moreover, if we let $r = 1/\varepsilon$, we can solve this recurrence using the master theorem and obtain that $S(n) = O(n^{\frac{\lfloor d/2 \rfloor \log(Cr \log r)}{\log r}})$ for some constant $C > 0$. Therefore, by choosing $r = 1/\varepsilon$ sufficiently large, we obtain that the total space is $S(n) = O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$ arbitrarily small. To analyze the time needed to construct this hierarchical tree, recall that an ε-net can be computed in linear time [18] which leads to the following recurrence $T(n) = O(|N|^{\lfloor d/2 \rfloor})S(\varepsilon n) + O(n)$. Using the same arguments as with the space we solve this recurrence and obtain that the total time is $T(n) = O(n^{\lfloor d/2 \rfloor + \delta})$ for any $\delta > 0$ arbitrarily small. ☐

Testing intersection in higher dimensions

Using hierarchical trees, we extend the ideas used for the 3D-algorithm presented in Section 5 to higher dimensions. We start by describing the preprocessing of a polyhedron.

Preprocessing

Let P be a polyhedron $\mathbb{R}^d$ with combinatorial complexity n. Assume without loss of generality that the origin lies in the interior of P. Otherwise, modify the coordinate system. To preprocess P, we first compute the polyhedron $\rho_{\mathbb{0}}(P)$ being the polarization of P. Then, we compute two hierarchical trees as described in the previous section, one for P and another for $\rho_{\mathbb{0}}(P)$. Similarly to the 3D case, because only one of the two hierarchical trees will ever be used in a single intersection query, we can independently use Simulation of Simplicity [11] in the construction of each of the trees. Because $|F(\rho_{\mathbb{0}}(P))| = |F(P)| = n$ by Corollary 2.14 of [23], the total size of these hierarchical trees is $O(n^{\lfloor d/2 \rfloor + \delta})$.

Preliminaries of the algorithm

Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^d$ with combinatorial complexity n and m, respectively. Throughout this algorithm, we fix the coordinate system used in the preprocessing of R, i.e., we assume that $\mathbb{0} \in R$. For ease of notation, let $Q = \rho_{\mathbb{0}}(R)$. Because $\mathbb{0} \in R$, Lemma 3.2 implies that $R = \rho_{\mathbb{0}}(Q)$. Assume that P and $\rho_{\mathbb{0}}(Q)$ lie in a *primal space* while $\rho_{\infty}(P)$ and Q lie in a *polar space*. As in the 3D-algorithm, we look at the primal and polar spaces independently and switch between them whenever necessary.

To test the intersection of P and $R = \rho_{\mathbb{0}}(Q)$, we use the hierarchical trees of P and Q computed during the preprocessing step. The idea is to walk down these trees using paths going from the root to a leaf while maintaining some invariants.

Throughout the algorithm, we prune the faces of P and keep only those that can define an intersection. Formally, we consider a set $F^*(P) \subseteq F(P)$ such that P intersects $\rho_\emptyset(Q)$ if and only if a face of $F^*(P)$ intersects $\rho_\emptyset(Q)$. In the same way, we prune $F(Q)$ and maintain a set $F^*(Q) \subseteq F(Q)$ such that Q intersects $\rho_\infty(P)$ if and only if a face of $F^*(Q)$ intersects $\rho_\infty(P)$. If these properties hold, we say that the *correctness invariant* is maintained.

At the beginning of the algorithm let $F^*(P) = F(P)$ and $F^*(Q) = F(Q)$. In each round of the algorithm we discard a constant fraction of the vertices of either $F^*(P)$ or $F^*(Q)$ while maintaining the correctness invariant. Note that these sets are not explicitly maintained.

Throughout, we consider constant size polyhedra $P_N \subseteq P$ and $Q_N \subseteq Q$ being the convex hull of ε-nets of $F^*(P)$ and $F^*(Q)$, respectively. The algorithm tests if P_N and $\rho_\emptyset(Q_N)$ intersect to determine either the separation or the inverse separation invariant, both analogous to those used by the 3D-algorithm. Formally, the *separation invariant* states that we have a hyperplane π that separates P_N from $\rho_\emptyset(Q_N)$ such that π is tangent to P_N at one of its vertices. The *inverse separation invariant* states that there is a hyperplane μ that separates $\rho_\infty(P_N)$ from Q_N such that μ is tangent to Q_N at one of its vertices. By Theorem 3.7 at least one of the invariants must hold.

At the beginning of the algorithm, we let $P_N \subseteq P$ and $Q_N \subseteq Q$ be the convex hulls of the ε-nets computed for $F(P)$ and $F(Q)$ at the root of their respective hierarchical trees. Because they have constant complexity, we can test if the separation or the inverse separation invariant holds. Depending on which invariant is established, we step into the primal or the polar space as follows (if both invariants hold, we choose arbitrarily).

Separation invariant.

If the separation invariant holds, then we have a hyperplane π tangent to P_N at a vertex v such that π separates P_N from $\rho_\emptyset(Q_N)$. Therefore, by Lemma 6.1 there is a simplex $\sigma \in \Sigma(P_N)$ such that π is also tangent to σ at v. Because we stored $\Sigma(P_N)$ in the hierarchical tree, we go through the $O(1)$ shell-simplices of $\Sigma(P_N)$ to find σ. Recall that $F_{\bar{\sigma}}$ is the set of faces of $F^*(P)$ that intersect the complement of σ. Thus, every face of P intersecting the halfspace $\pi_\emptyset$ belongs to $F_{\bar{\sigma}}$.

Because π separates P_N from $\rho_\emptyset(Q) \subseteq \rho_\emptyset(Q_N) \subseteq \pi_\emptyset$, the only faces of $F^*(P)$ that could define an intersection with $\rho_\emptyset(Q)$ are those in $F_{\bar{\sigma}}$, i.e., a face of $F^*(P)$ intersects $\rho_\emptyset(Q)$ if and only if a face of $F_{\bar{\sigma}}$ intersects $\rho_\emptyset(Q)$. Because the correctness invariant held prior to this step, we conclude that a face of $F_{\bar{\sigma}}$ intersects $\rho_\emptyset(Q)$ if and only if P intersects $\rho_\emptyset(Q)$.

Recall that we have recursively constructed a tree for $F_{\bar{\sigma}}$ which hangs from the node storing P_N. In particular, in the root of this tree we have stored the convex hull of an ε-net of $F_{\bar{\sigma}}$. Therefore, after finding σ in $O(1)$ time, we move down one level to the root of the tree of $F_{\bar{\sigma}}$. Then, we redefine P_N to be the convex hull of the ε-net of $F_{\bar{\sigma}}$ stored in this node. Moreover, we let $F^*(P) = F_{\bar{\sigma}}$ which preserves the correctness invariant. Then, we test if the new P_N and $\rho_\emptyset(Q_N)$ intersect to determine if either the separation or inverse separation invariant holds. In this way, we moved down one level in the hierarchical tree of P and proceed with a new round of the algorithm.

Inverse separation invariant.

If the inverse separation invariant holds, then we have a hyperplane that separates $\rho_\infty(P_N)$ from Q_N. Applying an analogous procedure to the one described for the separation invariant, we redefine Q_N and move down one level in the hierarchical tree of Q while maintaining the correctness invariant. Then, we test if $\rho_\infty(P_N)$ intersects the new Q_N to determine if either the separation or inverse separation invariants holds and proceed with the algorithm.

After $O(\log n + \log m)$ rounds, the algorithm reaches the bottom of the hierarchical tree of either P or Q. If we reach the bottom of the hierarchical tree of P and the separation invariant holds, then because $\rho_\emptyset(Q_N) \supseteq \rho_\emptyset(Q)$ by Lemma 3.6, we have a hyperplane that separates $P_N = \text{CH}(F^*(P))$ from $\rho_\emptyset(Q_N)$. That is, no face of $F^*(P)$ intersects $\rho_\emptyset(Q_N)$. Because P and $\rho_\emptyset(Q)$ intersect if and only if a face of $F^*(P)$ intersects $\rho_\emptyset(Q)$ by the correctness invariant, we conclude that P and $R = \rho_\emptyset(Q)$ do not intersect.

Analogously, if we reach the bottom of the hierarchical tree of Q and the inverse separation invariant holds, then we have a hyperplane that separates $Q_N = \text{CH}(F^*(Q))$ from $\rho_\infty(P_N) \supseteq \rho_\infty(P)$. That is, no face of $F^*(Q)$ intersects $\rho_\infty(P)$. Thus, by the correctness invariant, we conclude that Q and $\rho_\infty(P)$ do not intersect. Therefore, Theorem 3.7 implies that P and $R = \rho_\emptyset(Q)$ intersect.

In any other situation the algorithm can continue until one of the two previously mentioned cases arises and the algorithm finishes. Recall that the hierarchical trees of P and Q have logarithmic depth by Lemma 6.2. Because in each round we move down in the hierarchical tree of either P or Q, after $O(\log n + \log m)$ rounds the algorithm finishes. Moreover, since each round can be performed in $O(1)$ time, we obtain the following result.

Theorem 6.3. *Let P and R be two independently preprocessed polyhedra in $\mathbb{R}^d$ with combinatorial complexities n and m, respectively. For any given translations and rotations of P and R, we can determine if P and R intersect in $O(\log n + \log m)$ time.*

Note that this algorithm does not construct a hyperplane that separates P and $\rho_\emptyset(Q)$ or a common point, but only determines if such a separating plane or intersection point exists. In fact, if P is disjoint from $\rho_\emptyset(Q)$, then we can take the $O(\log n)$ hyperplanes found by the algorithm, each of them separating some portion of P from $\rho_\emptyset(Q)$. Because all these hyperplanes support a halfspace that contains $\rho_\emptyset(Q)$, their intersection defines a polyhedron S that contains $\rho_\emptyset(Q)$ and excludes P. Therefore, we have a certificate of size $O(\log n)$ that guarantees that P and $\rho_\emptyset(Q)$ are separated.

Similarly, if Q is disjoint from $\rho_\infty(P)$, then we can find a polyhedron of size $O(\log m)$ whose boundary separates Q from $\rho_\infty(P)$. In this case, we have a certificate that guarantees that Q and $\rho_\infty(P)$ are disjoint which by Theorem 3.7 implies that P and $\rho_\emptyset(Q)$ intersect.

Acknowledgments.

We thank David Kirkpatrick for his observations regarding the oversight in [6]. We thank an anonymous referee for useful comments in a previous version of this paper.

References

[1] T. Biedl and D. F. Wilkinson. Bounded-degree independent sets in planar graphs. *Theory of Computing Systems*, 38(3):253–278, 2005.

[2] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal on Computing*, 21:586–591, 1992.

[3] B. Chazelle and D. P. Dobkin. Detection is easier than computation (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 146–153, 1980.

[4] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the ACM*, 34(1):1–27, Jan. 1987.

[5] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[6] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra—a unified approach. *Automata, Languages and Programming*, pages 400–413, 1990.

[7] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[8] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6(3):381–392, 1985.

[9] D. P. Dobkin and D. L. Souvaine. Detecting the intersection of convex objects in the plane. *Computer aided geometric design*, 8(3):181–199, 1991.

[10] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *Journal of Algorithms*, 6(2):213–224, 1985.

[11] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990.

[12] J. Erickson. Space-time tradeoffs for emptiness queries. *SIAM Journal on Computing*, 29(6):1968–1996, 2000.

[13] J. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition.* CRC Press LLC, 2004.

[14] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection: a survey. *Computers & Graphics*, 25(2):269–285, 2001.

[15] D. Kirkpatrick. Personal communication.

[16] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, volume 1, pages 602–608, 1998.

[17] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete & Computational Geometry*, 10(1):215–232, 1993.

[18] J. Matoušek. Construction of epsilon nets. In *Proceedings of the 5th Annual Symposium on Computational Geometry*, pages 1–10, New York, 1989. ACM.

[19] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[20] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor. A new linear algorithm for intersecting convex polygons. *Computer Graphics and Image Processing*, 19(4):384 – 391, 1982.

[21] M. I. Shamos. Geometric complexity. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, pages 224–233. ACM, 1975.

[22] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pages 208–215. IEEE, 1976.

[23] G. M. Ziegler. *Lectures on polytopes*, volume 152. Springer, 1995.