ELSEVIER
Computational Geometry 23 (2002) 195–207
Computational Geometry
Theory and Applications
www.elsevier.com/locate/comgeo

Reporting intersecting pairs of convex polytopes in two and three dimensions

Pankaj K. Agarwal [a,*], Mark de Berg [b], Sariel Har-Peled [c], Mark H. Overmars [b], Micha Sharir [d,e,1], Jan Vahrenhold [f,2]

[a] *Department of Computer Science, Duke University, Durham, NC 27708-0129, USA*
[b] *Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands*
[c] *Department of Computer Science, DCL 2111, University of Illinois, 1304 West Springfield Ave., Urbana, IL 61801, USA*
[d] *School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel*
[e] *Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA*
[f] *Westfälische Wilhelms-Universität Münster, Institut für Informatik, 48149 Münster, Germany*

Received 6 August 2001; accepted 10 January 2002

Communicated by S. Suri

Abstract

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of m convex polytopes in $\mathbb{R}^d$, for $d = 2, 3$, with a total of n vertices. We present output-sensitive algorithms for reporting all k pairs of indices (i, j) such that P_i intersects P_j. For the planar case we describe a simple algorithm with running time $O(n^{4/3}\log^{2+\varepsilon} n + k)$, for any constant $\varepsilon > 0$, and an improved randomized algorithm with expected running time $O((n\log m + k)\alpha(n)\log n)$ (which is faster for small values of k). For $d = 3$, we present an $O(n^{8/5+\varepsilon} + k)$-time algorithm, for any $\varepsilon > 0$. Our algorithms can be modified to count the number of intersecting pairs in $O(n^{4/3}\log^{2+\varepsilon} n)$ time for the planar case, and in $O(n^{8/5+\varepsilon})$ time for the three-dimensional case. © 2002 Elsevier Science B.V. All rights reserved.

* Corresponding author. P.A. was supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA-9870724, EIA-997287, and CCR-9732787, and by a grant from the U.S.–Israeli Binational Science Foundation.

E-mail addresses: pankaj@cs.duke.edu (P.K. Agarwal), markdb@cs.ruu.nl (M. de Berg), sariel@cs.uiuc.edu (S. Har-Peled), markov@cs.ruu.nl (M.H. Overmars), sharir@cs.tau.ac.il (M. Sharir), jan@math.uni-muenster.de (J. Vahrenhold).

1 M.S. was supported by NSF Grant CCR-97-32101, by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University, and by a grant from the U.S.–Israeli Binational Science Foundation.

2 Part of this work was done while visiting Duke University.

PII: S0925-7721(02)00049-4

1. Introduction

Computing intersections in a set of geometric objects is a fundamental problem in computational geometry. A basic version of this problem is when the objects are line segments in the plane. Indeed, computing the intersecting pairs in a set of n line segments was one of the first problems studied in computational geometry: Already in 1979, Bentley and Ottmann [8] described an algorithm for this problem with $O((n+k)\log n)$ running time, where k is the number of intersecting pairs of segments. Since then much research has been done on this problem, culminating in optimal—that is, with $O(n\log n+k)$ running time—deterministic algorithms by Chazelle and Edelsbrunner [13] and Balaban [6], and simpler randomized algorithms by Clarkson and Shor [15] and Mulmuley [20].

Another well-studied variant of the problem is the red–blue intersection problem. Here one is given a set of red segments and a set of blue segments, and the goal is to report all bichromatic intersections. If there are no monochromatic intersections, then the problem can be solved in $O(n\log n+k)$ time by applying an optimal standard line–segment intersection algorithm; when the red segments and the blue segments both form simply connected subdivisions, then the problem can even be solved in $O(n+k)$ time [16]. The situation becomes considerably more complicated when there are monochromatic intersections. Applying a standard line–segment intersection algorithm will not lead to an output-sensitive algorithm because it may report a quadratic number of monochromatic intersections even when there are no bichromatic intersections. Somehow one has to avoid processing all the monochromatic intersections. Agarwal and Sharir [4] showed that one can detect whether the two sets intersect in $O(n^{4/3+\varepsilon})$ time.[3] Later Agarwal [1] and Chazelle [10] gave $O(n^{4/3}\log^{O(1)} n+k)$-time algorithm to report all k red–blue intersections. Basch et al. [7] presented a deterministic $O((n+k)\alpha(n)\log^3 n)$ algorithm for the case where the set of red segments is connected and the set of blue segments is connected. This algorithm also works for the case of Jordan arcs, each pair of which intersect at most t times; its running time then becomes $O(\lambda_{t+2}(n+k)\log^3(n))$, where $\lambda_s(n)$—the maximum length of an (n,s) Davenport–Schinzel sequence—is an almost linear function of n for any fixed s [22]. The bound for the case of segments was later improved to $O((n+k)\log^2 n\log\log n)$ by Brodal and Jacob [9]. Har-Peled and Sharir [18] improved the general case of Jordan arcs by giving a randomized algorithm with $O(\lambda_{t+2}(n+k)\log n)$ running time.

We are interested in the case in which the input consists of convex polygons in the plane. We want to compute all intersecting pairs of polygons. More formally, we are given a set $\mathcal{P}=\{P_1,\ldots,P_m\}$ of m convex polygons in $\mathbb{R}^2$ with a total of n vertices, and we want to report all k pairs of indices i,j such that P_i intersects P_j. (The polygons are considered to be 2-dimensional regions, so two polygons intersect also in the case that one of them is fully contained inside the other.) If each polygon P_i has constant complexity, then the number of intersections between pairs of edges will not exceed the total number of intersecting pairs of polygons by more than a constant factor, and one can solve the problem in $O(n\log n+k)$ time, by a straightforward modification of the algorithms mentioned above for reporting segment intersections. If the given polygons do not have constant complexity, then the problem becomes considerably harder because the intersection of a pair of the given polygons can have many vertices. Regarding each input polygon as a collection of segments will thus not lead to an output-sensitive algorithm in this case.

---

[3] The meaning of a bound like this is that for any $\varepsilon>0$ there exists a constant $c=c(\varepsilon)$ that depends on ε, so that the bound holds with c as the constant of proportionality.

Gupta et al. [17] nevertheless developed an output-sensitive algorithm for this case that runs in time $O(n^{4/3+\varepsilon} + k)$. The algorithm first computes a trapezoidal decomposition for each polygon. Then it computes, using a multi-level partition tree, those pairs of intersecting trapezoids such that the leftmost intersection point of the trapezoids is also the leftmost intersection point of the corresponding polygons. This way it is ensured that each intersecting pair of polygons is reported exactly once.

We develop two new algorithms for this problem. The first algorithm is randomized and combines hereditary segment trees [14] with the above mentioned red–blue intersection algorithm of Har-Peled and Sharir [18]. Its expected running time is $O((n \log m + k)\alpha(n) \log n)$ and it is significantly faster than the algorithm of Gupta et al. for $k = o(n^{4/3})$. In addition, the algorithm also works for convex splinegons (that is, convex shapes whose boundary is composed of Jordan arcs) with only a minor increase in running time; this is not the case for the algorithm of Gupta et al. Our algorithm can be made deterministic at the expense of an additional polylogarithmic factor.

Our second algorithm has $O(n^{4/3} \log^{2+\varepsilon} n + k)$ running time, for any constant $\varepsilon > 0$, and is thus slightly faster than our first algorithm for $k = \Omega(n^{4/3})$. It is related to the algorithm of Gupta et al.—it uses partition trees and similar techniques to search for the rightmost intersection points of intersecting pairs of polygons—but it is conceptually simpler and it has a slightly better running time.

The main advantage of our approach over Gupta et al.'s is that it generalizes to the 3-dimensional version of the problem: Given a set $\mathcal{P} = \{P_1, \ldots, P_m\}$ of m convex polytopes in $\mathbb{R}^3$ with a total of n vertices, report all k pairs of indices (i, j) such that P_i intersects P_j. For this problem, no subquadratic algorithm was known except for the special case where the polytopes satisfy two special properties: the minimum aspect ratio of any bounding box of the polytopes is not too small (thus the polytopes must be fat), and the scale factor (the ratio of the sizes of the largest and the smallest polytope) is not too large. In particular, if these two values are constant, Suri et al. [23] gave an algorithm for arbitrary dimension d with running time $O(n \log^{d-1} n + k \log^{d-1} n)$. Their algorithm works by first computing all pairs of bounding boxes that intersect, and then checking for each such pair whether the polytopes themselves intersect. Under the conditions mentioned above, the number of intersecting bounding boxes is in the same order of magnitude as the number of intersecting polytopes (up to an additive linear term), which means their algorithm is efficient. In general, however, the number of intersecting bounding boxes can be quadratic even when there are no intersections at all among the polytopes.

We generalize our second 2-dimensional algorithm to 3-dimensional space, and obtain an algorithm with running time $O(n^{8/5+\varepsilon} + k)$, for any $\varepsilon > 0$. Such a generalization seems hard for the algorithm of Gupta et al., as the vertical decomposition of a convex polytope can have quadratic complexity. Note that our algorithm for the 3-dimensional case has the same running time as the best known algorithm for the much simpler problem of reporting all intersecting pairs in a set of triangles in $\mathbb{R}^3$ [3].

2. The planar case

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of m convex polygons in the plane, with a total of n vertices. For simplicity, we assume that none of the polygons has a vertical edge and that all the vertex coordinates are distinct; we can enforce this in $O(n \log n)$ time by applying a suitable rotation. We also assume that no two edges overlap (that is, intersect in more than one point). To this end, we shift such edges slightly in $O(n \log n + k)$ time in a preprocessing step; this can be done in such a way that the collection of intersecting pairs of polygons does not change.

For a polygon P_i, we define ℓ_i to be the leftmost point of P_i and r_i to be the rightmost point of P_i (since there are no vertical edges, ℓ_i and r_i are uniquely defined). They partition the boundary of P_i into two convex chains: the *upper chain*, denoted U_i, and the *lower chain*, denoted L_i. Note that the rightmost vertex of $P_i \cap P_j$ is r_i, r_j, an intersection point of U_i with L_j, or an intersection point of L_i with U_j.

We first describe an algorithm whose running time is near-linear in n and k, and then a worst-case optimal algorithm for the case of large k whose running time is $\mathrm{O}(n^{4/3}\log n + k)$.

2.1. A near-linear randomized algorithm

We present a randomized algorithm that reports, in $\mathrm{O}((n\log m + k)\alpha(n)\log n)$ expected time, all k intersecting pairs of polygons in $\mathcal{P}$. For each polygon P_i, we define s_i to be the segment connecting ℓ_i to r_i; we call s_i the *spine* of P_i. Let $\mathcal{SP}$ denote the set of all the spines.

Our algorithm starts by constructing a *hereditary* segment tree T on (the x-projections of) the spines of $\mathcal{SP}$ [14]. Each node v of T is associated with a vertical strip W_v and with a subset $\mathcal{SP}(v)$ of spines. A spine s_i intersecting W_v is *short* at v if at least one of its endpoints lies in the interior of W_v, otherwise it is *long*. The set $\mathcal{SP}(v)$ is the subset of spines that intersect W_v and are short at the parent of v. If v is the root, then $\mathcal{SP}(v) = \mathcal{SP}$. Let $\mathcal{P}(v) = \{P_i \mid s_i \in \mathcal{SP}(v)\}$. A polygon is short (respectively, long) at v if its spine is short (respectively, long) at v. As shown in [14], $\sum_v |\mathcal{P}(v)| = \mathrm{O}(m\log m)$.

We assume that $\mathcal{SP}(v)$ and $\mathcal{P}(v)$ are clipped to within W_v. At each node v of the tree, we will report all pairs (i, j) with the following property:

$$\text{the rightmost intersection point of } P_i \text{ and } P_j \text{ lies inside } W_v \text{ and } P_i \text{ is long at } v. \qquad (\star)$$

The following lemma is straightforward from the structure of hereditary segment trees.

Lemma 2.1. *For every pair of intersecting polygons P_i and P_j, there is exactly one node v of T at which property $(\star)$ holds.*

Let k_v be the number of pairs that satisfy property $(\star)$ at a node v. Then $\sum_v k_v = k$. Our procedure will ensure that a pair (i, j) is reported only once, at the node where $(\star)$ is satisfied, but it will spend roughly $\mathrm{O}(\log n)$ time for each intersecting pair.

Fix a node v. Let $\mathcal{P}_L \subseteq \mathcal{P}(v)$ denote the subset of long polygons at v, and let $\mathcal{P}_S \subseteq \mathcal{P}(v)$ denote the subset of short polygons at v. Denote the set of spines of $\mathcal{P}_L$ by $\mathcal{SP}_L$, the set of their upper chains by $\mathcal{U}_L$, and the set of their lower chains by $\mathcal{L}_L$. The sets $\mathcal{SP}_S$, $\mathcal{U}_S$, and $\mathcal{L}_S$ are defined analogously for $\mathcal{P}_S$. Again, all these objects are clipped to within W_v. Let n_v denote the total number of edges in (the clipped) $\mathcal{P}_L$ and $\mathcal{P}_S$. As above, the structure of hereditary segment trees implies that $\sum_v n_v = \mathrm{O}(n\log m)$. Finally, we define R_S to be the set of right endpoints of the spines in $\mathcal{SP}(v)$ that lie in the interior of W_v. Note that every point in R_S is the right endpoint of an (unclipped) original spine in $\mathcal{SP}$. Let μ_v be the number of intersection points between $\mathcal{SP}_L$ and $\mathcal{SP}(v) \cup \partial\mathcal{P}(v)$—ignoring, of course, "intersections" between a spine and itself—plus the number of intersection points between the upper (respectively lower) chains of $\mathcal{P}_L$ and the lower (respectively upper) chains of $\mathcal{P}(v)$, where $\partial\mathcal{P}(v) = \{\partial P \mid P \in \mathcal{P}(v)\}$.

Lemma 2.2. $\sum_{v\in T} \mu_v = \mathrm{O}(k)$.

Proof. Let σ be an intersection point of a spine $s_i \in \mathcal{SP}_L(v)$ and another spine $s_j \in \mathcal{SP}(v)$; σ is one of the intersection points counted by μ_v. We claim that v is the only node at which σ is counted by μ_v. It is obvious that σ cannot be counted by a node w other than an ancestor or a descendent of v, as the vertical strip W_w associated with w has to contain the intersection point σ. Since neither s_i nor s_j belongs to $\mathcal{SP}_L(w)$ for any ancestor w of v, σ will not by counted by μ_w. On the other hand, s_i does not belong to $\mathcal{SP}(u)$ for any descendent u of v, so σ will not be counted by any descendent of v either. Hence v is the only node at which σ is counted. A similar claim holds for an intersection point of $\mathcal{SP}_L(v)$ and $\partial\mathcal{P}(v)$ or of upper (respectively lower) chains of $\mathcal{P}_L(v)$ and lower (respectively upper) chains of $\mathcal{P}(v)$. Since there are $O(k)$ intersection points between two spines, between a spine and a polygonal chain, and between upper and lower polygonal chains, the lemma follows. □

Since all the vertex coordinates are distinct, there exists at most one spine in $\mathcal{SP}(v)$ whose right endpoint r_i lies on the right boundary of W_v. We can easily compute in $O(n_v)$ time all polygons of $\mathcal{P}(v)$ that contain r_i. We now describe how we report all the other pairs that satisfy (⋆) at v.

We construct, in $O(n_v \log n_v + \mu_v)$ time, the arrangement $\mathcal{A} = \mathcal{A}(\mathcal{SP}_L)$ of the spines of the long polygons [13]. We also add the vertical lines bounding W_v to $\mathcal{A}$. Each face f of $\mathcal{A}$ is a convex polygon, so we can compute the intersections between a line and ∂f in $O(\log n_v)$ time. We preprocess $\mathcal{A}$, in $O((n_v + \mu_v)\log n_v)$ time, for planar point-location queries [21]. For each edge e of $\mathcal{P}(v)$, we locate its left endpoint in $\mathcal{A}$ and then trace it through $\mathcal{A}$, spending $O(\log n_v)$ time at each face of $\mathcal{A}$ that e intersects.

For each face $f \in \mathcal{A}$, we report the pairs (i, j) that satisfy (⋆) at v and for which the rightmost point of $P_i \cap P_j$ lies inside f. This is accomplished in the following three stages.

(a) Report all pairs (i, j) such that $P_i \in \mathcal{P}_L$ contains the right endpoint $r_j \in R_S$ and $r_j \in f$.
(b) Report all pairs (i, j) such that the lower chain of $P_i \in \mathcal{P}_L$ intersects the upper chain of $P_j \in \mathcal{P}(v)$ and the rightmost point of their intersection lies inside f.
(c) Report all pairs (i, j) such that the upper chain of $P_i \in \mathcal{P}_L$ intersects the lower chain of $P_j \in \mathcal{P}(v)$ and the rightmost point of their intersection lies inside f.

It is easily verified that stages (a)–(c) indeed report all the desired intersections. Since (b) and (c) are symmetric, we omit the description of (c).

Containments of rightmost points. Let $R(f) \subset R_S$ be the subset of right endpoints that lie inside f. Using the point-location structure we have constructed for $\mathcal{A}$, the sets $R(f)$ can be found in $O(n_v \log n_v)$ time in total. We wish to report all pairs (i, j) such that $r_j \in R(f)$ lies inside $P_i \in \mathcal{P}_L$. Let $\mathcal{P}(f) \subseteq \mathcal{P}_L$ denote the set of long polygons that contain f in their interior (i.e., for a polygon $P \in \mathcal{P}(f)$, we have $f \subseteq P$), and let $\mathcal{Q}(f) \subseteq \mathcal{P}_L$ denote the set of polygons whose boundaries intersect f. Let n_f denote the number of vertices of the polygons in $\mathcal{Q}(f)$ that lie inside f, and let n'_f denote the number of edges in $\mathcal{Q}(f)$ that intersect f but their endpoints do not lie inside f. Then

$$\sum_f n_f \leqslant n_v \quad \text{and} \quad \sum_f n'_f \leqslant \mu_v. \tag{2}$$

Obviously, $|\mathcal{Q}(f)| \leqslant n_f + n'_f$. Since we have already traced the edges of $\mathcal{P}_L(v)$ through $\mathcal{A}$, we have $\mathcal{Q}(f)$ at our disposal. However, we do not store $\mathcal{P}(f)$ explicitly for each face f because the resulting storage could be quite large.

We first describe how to deal with the polygons in $\mathcal{P}(f)$. Note that every point in $R(f)$ lies inside every polygon in $\mathcal{P}(f)$, so we can report every pair in $\mathcal{P}(f) \times R(f)$. We do this for all faces f in a single plane sweep, as described below. In fact, the algorithm will report a superset of these pairs: a pair P_i, r_j with $r_j \in f$ will be reported if and only if the intersection of f with the vertical line through r_j is contained in the intersection of P_i with that line. This clearly holds for all polygons in $\mathcal{P}(f)$, but it may also hold for some of the polygons in $\mathcal{Q}(f)$. Hence, when we are dealing with the polygons from $\mathcal{Q}(f)$ we have to make sure that we do not report any pair for the second time. This is easy to do, as we can afford to spend $O(\log n_v)$ time to check a pair.

We now come to the plane sweep. The sweep is from left to right. While we sweep, we maintain the following information. First of all, we have a balanced binary search tree $\mathcal{T}_1$ storing all the long spines in the order in which they intersect the sweep line λ. We also maintain a segment tree $\mathcal{T}_2$. Its elementary intervals are of the form $[i : i+1]$, for $1 \leqslant i < n_v$. The segment tree stores the intersections of the polygons in $\mathcal{P}_L$ with λ in the following way. Number the faces of $\mathcal{A}$ that are intersected by λ from bottom to top as $f_0, \ldots, f_r$, for some $r \leqslant n_v$. Suppose that the intersection of λ with the lower (upper) boundary of a given polygon P lies in f_l (f_u). If $l < u$, we store for P the interval $[l : u]$ in $\mathcal{T}_2$. Otherwise $l = u$, and P is currently not stored in $\mathcal{T}_2$. Note that these structures can be easily built in $O(n_v \log n_v)$ time when the sweep starts (and λ is at the left boundary of the strip W_v). There are three types of events during the sweep.

The first type is a vertex of $\mathcal{A}$. At such an event we update $\mathcal{T}_1$ in $O(\log n_v)$ time. The second type of event is an intersection of the boundary of a polygon $P \in \mathcal{P}_L$ with an edge of $\mathcal{A}$. When this happens, the interval we stored for P in $\mathcal{T}_2$ has to be changed. This can be done in $O(\log n_v)$ time. The third type of event is when we reach a point $r_j \in R_S$. We then search with r_j in $\mathcal{T}_1$ to see in which face f it lies. With that information we can search in $\mathcal{T}_2$ and report all polygons P such that $f \cap \lambda \subset P \cap \lambda$. In total, the sweep takes time $O((n_v + \mu_v + k_v)\log n_v)$.

Next, for every point $r_j \in R(f)$, we report the polygons in $\mathcal{Q}(f)$ that contain r_j. We build a *union tree* Ψ on the polygons in $\mathcal{Q}(f)$, which is a minimum-height binary tree whose leaves store the polygons of $\mathcal{Q}(f)$. Each node ξ of Ψ is associated with the subset $\mathcal{Q}_\xi \subseteq \mathcal{Q}(f)$ of polygons that are stored at the leaves of the subtree rooted at ξ. Let ν_ξ be the total number of vertices of the polygons in $\mathcal{Q}_\xi$ that lie in the interior of f, and let ν'_ξ be the number of edges of the polygons in $\mathcal{Q}_\xi$ that intersect f but both of whose endpoints do not lie inside f; we have

$$\sum_\xi \nu_\xi = O(n_f \log n_v) \quad \text{and} \quad \sum_\xi \nu'_\xi = O(n'_f \log n_v).$$

For any polygon $P \in \mathcal{Q}(f)$ either its upper or its lower boundary intersects f, but not both, because the spine of P does not intersect the interior of f. We partition Q_ξ into two subsets Q^+_ξ, Q^-_ξ; a polygon $P \in Q_\xi$ belongs to Q^+_ξ (respectively Q^-_ξ) if the upper (respectively lower) chain of P intersects the interior of f. We construct the lower envelope $\mathcal{L}_\xi$ of the lower chains of the intersection polygons $\{P \cap f \mid P \in Q^-_\xi\}$ and the upper envelope $\mathcal{U}_\xi$ of the upper chains of the intersection polygons $\{P \cap f \mid P \in Q^+_\xi\}$. We store only those portions of the envelopes which lie in the interior of f. These portions have $O((\nu_\xi + \nu'_\xi)\alpha(n_v))$ breakpoints, where a breakpoint is a vertex of a polygon, an intersection point of two lower (upper) chains, or an intersection point of ∂f and an edge of a polygon. If we have already computed the lower and upper envelopes of the children of ξ, then $\mathcal{L}_\xi, \mathcal{U}_\xi$ can be computed in an additional $O((\nu_\xi + \nu'_\xi)\alpha(n_v))$ time. We store the sequences of breakpoints of $\mathcal{L}_\xi$ and of $\mathcal{U}_\xi$ in an array, sorted from left to right. For each breakpoint, we store the segment that appears on the envelope immediately to its left if the envelope lies in the interior of f to the left of the breakpoint; otherwise we mark that the envelope appears on ∂f to the left of the breakpoint. We also apply fractional cascading [11] so that if we know the breakpoint of $\mathcal{L}_\xi$ (respectively $\mathcal{U}_\xi$) that lies immediately to the right of a given x-coordinate x_0, we can compute, in O(1) time, the corresponding breakpoints at the children of ξ. The total time spent in preprocessing Ψ is $O((n_f + n'_f)\alpha(n_v)\log n_v)$.

For each point $r_j \in R(f)$, we find all polygons in $\mathcal{Q}(f)$ containing r_j by traversing Ψ in a top-down manner. Suppose we are at a node ξ of Ψ. Since f is not crossed by any spine, r_j lies in a polygon $P \in Q_\xi^+$ (respectively $P \in Q_\xi^-$) if and only if r_j lies below (respectively above) the upper (respectively lower) chain of $P \cap f$. We thus find the breakpoints of $\mathcal{L}_\xi, \mathcal{U}_\xi$ that lie immediately to the right of r_j. We can then decide in O(1) time whether r_j lies inside any polygon of Q_ξ, by determining whether it lies below $\mathcal{U}_\xi$ or above $\mathcal{L}_\xi$. If r_j does not lie in any polygon of $\mathcal{Q}_\xi$, we stop. If ξ is a leaf and r_j lies inside the only polygon, say P_i, in $\mathcal{Q}_\xi$, then we return the pair (i, j). If ξ is not a leaf and r_j lies inside a polygon of $\mathcal{Q}_\xi$, we recursively visit the children of ξ. Suppose r_j lies inside k_j polygons of $\mathcal{Q}(f)$, then the query procedure visits $O(1 + k_j \log n_v)$ nodes of Ψ. It spends $O(\log n_v)$ time at the root and O(1) at any other node, so the time spent in processing r_j is $O((1 + k_j)\log n_v)$. Hence, the algorithm spends

$$O\left(\left((n_f + n'_f)\alpha(n_v) + \sum_{r_j \in R(f)} (1 + k_j)\right)\log n_v\right)$$

time at the face f. Summing over all the faces of $\mathcal{A}$ and using (2), we obtain that the total time spent in reporting the pairs that satisfy condition (a), over all faces f of $\mathcal{A}$, is $O((n_v + \mu_v + k_v)\alpha(n_v)\log n_v)$.

Intersections between long lower chains and upper chains. For a face f of $\mathcal{A}$, let $L(f)$ denote the set of maximal connected portions of the chains in $\mathcal{L}_L$ that lie inside f, let $U(f)$ denote the set of maximal connected portions of upper chains of (short and long) polygons in $\mathcal{P}(v)$ that lie inside f, and let $\mathcal{SP}(f)$ denote the set of portions of short spines inside f. Since we have traced the edges of $\mathcal{P}(v)$ through $\mathcal{A}$, the sets $L(f)$ and $U(f)$ are already available for all faces f. We will report all pairs (i, j) that satisfy $(\star)$ and whose rightmost intersection points lie inside f.

The endpoints of all chains in $L(f)$ lie on ∂f because they are portions of long chains. Let A_f be the set of edges that constitute $L(f)$ and ∂f; set $a_f = |A_f|$. The union of A_f is connected. If both endpoints of a chain $\gamma \in U(f)$ lie in the interior of f, then γ is the entire upper chain of a short polygon P_j. In this case, we add a vertical segment σ_j from the right endpoint r_j of P_j downwards until it meets ∂f. Let B_f denote the union of the set of edges that constitute $U(f)$ and ∂f, and the set of vertical segments that we have just added; set $b_f = |B_f|$. By construction, the union of B_f is also connected because all the upper chains in $U(f)$ are connected to ∂f after introducing the vertical segments. Since the unions of A_f and of B_f are both connected, we can use the randomized algorithm of Har-Peled and Sharir [18] to compute all I_f intersection points between the segments of A_f and of B_f that lie in the interior of f, in $O((a_f + b_f + I_f)\alpha(n_v)\log n_v)$ expected time.

We report a pair (i, j) if there exists an edge e_i of P_i in A_f and an edge e_j of P_j in B_f such that the intersection point of e_i and e_j (one of the I_f intersections that we have computed) is the rightmost vertex of $P_i \cap P_j$. The total expected running time spent in reporting the pairs that satisfy property (b) is $\sum_f O((a_f + b_f + I_f)\alpha(n_v)\log n_v)$. Each endpoint of a segment of A_f or of B_f is a vertex of $\mathcal{P}(v)$, an intersection point of a long spine and an edge of $\mathcal{P}(v)$, or the lower endpoint of a vertical segment σ_j. Therefore, $\sum_f (a_f + b_f) = O(n_v + \mu_v)$. The expected running time is thus $O((n_v + \mu_v + \sum_f I_f)\alpha(n_v)\log n_v)$.

We call an intersection point of $e \in A_f$ and $e' \in B_f$ *real* if e is an edge of a lower chain in $L(f)$ and e' is an edge of an upper chain in $U(f)$; otherwise we call the intersection point *virtual*. Each real intersection point is an intersection point of $\mathcal{L}_L$ and the upper chains of $\mathcal{P}(v)$, so the total number of real intersection points, summed over all faces of $\mathcal{A}$, is $O(\mu_v)$. Since ∂f does not intersect the relative interior of any segment in $U(f)$ or $L(f)$, a virtual intersection point is an intersection point $e \cap e'$, where e is an edge of the lower chain of a long polygon P_i and e' is the vertical segment σ_j emanating from the right endpoint r_j of (the upper chain of) a short polygon P_j. We can ignore intersections on ∂f because they correspond to degenerate intersections between A_f and B_f, and, in any case, their number is only $O(\mu_v)$. Since P_i is a long polygon, its spine s_i is in $\mathcal{SP}_L$. Therefore, s_i lies above the interior of the face f and thus above r_j. The intersection of e and σ_j implies that r_j is inside P_i. We charge the intersection point $e \cap e'$ to the pair (i, j). Each pair (i, j) is charged by at most one virtual intersection point and the pair (i, j) is reported at v, therefore the total number of virtual intersection points, summed over all faces of $\mathcal{A}$, is at most k_v. Hence, $\sum_f I_f = O(k_v + \mu_v)$, and the total expected time spent in executing stage (b) is $O((n_v + k_v + \mu_v)\alpha(n_v)\log n_v)$.

We have thus described procedures for reporting all intersecting pairs that satisfy properties (a)–(c) at a node v of T. The total expected time we spend at v is $O((n_v + k_v + \mu_v)\alpha(n_v)\log n_v)$. Since $\sum_v n_v = O(n \log m)$, $\sum_v k_v = k$, and $\sum_v \mu_v = O(k)$ (Lemma 2.2), we obtain our first main result.

Theorem 2.3. *Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be m convex polygons in the plane with a total of n vertices. All k pairs of indices (i, j) such that P_i intersects P_j can be reported in $O((n \log m + k)\alpha(n)\log n)$ expected time.*

Remark 2.4.

(i) To get a worst-case time bound instead of an expected time bound, we can replace the algorithm of Har-Peled and Sharir [18] used in the second part of the algorithm by an algorithm of Basch et al. [7]. This will increase the time bound by a polylogarithmic factor.

(ii) The algorithm also works for splinegons, whose boundaries are composed of Jordan arcs instead of straight edges, provided the splinegons are still convex. If t is the maximum number of intersection points between any pair of Jordan arcs that form the boundaries of the splinegons, the running time of the algorithm is $O((\lambda_{t+2}(n)\log m + \lambda_{t+2}(k))\log n)$.

2.2. An alternative deterministic algorithm

Let P_i and P_j be two intersecting polygons in $\mathcal{P}$. Recall that the rightmost vertex of $P_i \cap P_j$ is r_i, r_j, an intersection point of the upper chain of P_i with the lower chain of P_j, or an intersection point of the lower chain of P_i with the upper chain of P_j. Using this observation, we can report the intersecting pairs of the given polygons as follows.

Let $V = \{r_i \mid 1 \leqslant i \leqslant m\}$. We first report all intersecting pairs of polygons for which the rightmost vertex of the intersection polygon is the rightmost vertex of one of the two polygons. A vertex r_i is the rightmost vertex of $P_i \cap P_j$ if and only if $r_i \in P_j$. For each P_i, we therefore report $P_i \cap V$. Using the range-searching data structure of Matoušek [19], for given parameters $m \leqslant s \leqslant m^2$ and $\varepsilon > 0$, we preprocess V, in time $O(m^{1+\varepsilon} + s \log^{\varepsilon} m)$, into a data structure of size $O(s)$, so that all μ_i points of $P_i \cap V$

can be reported in time $O(|P_i|(m/\sqrt{s})\log^3 m + \mu_i)$. Since $\sum_{i=1}^{m}|P_i| = n$, the total time spent in this step is

$$O\big(m^{1+\varepsilon} + s\log^{\varepsilon} m + (mn/\sqrt{s})\log^3 m + \mu\big),$$

where $\mu = \sum_{i=1}^{m}|P_i \cap V| \leqslant k$. Choosing $s = \max\{m^{2/3}n^{2/3}\log^2 m, m^2\}$, the running time becomes $O(m^{2/3}n^{2/3}\log^{2+\varepsilon} m + n\log^3 m + \mu)$.

Next, we report the pairs (i, j) such that the rightmost vertex of $P_i \cap P_j$ is an intersection point of an edge of P_i with an edge of P_j. Let U be the set of segments in the upper chains of the polygons in $\mathcal{P}$, and let L be the set of segments in the lower chains of these polygons. We compute all ν intersecting pairs of segments between U and L. This can be accomplished in $O(n^{4/3}\log^{2/3} n + \nu)$ time [1,10]. Suppose that an edge e of the upper chain of P_i and an edge e' of the lower chain of P_j intersect. We check in $O(1)$ time whether $e \cap e'$ is the rightmost vertex of $P_i \cap P_j$, and, if so, report the pair (i, j). Since an upper chain intersects a lower chain in at most two points, the number of intersections between U and L is at most $2k$, where k is the number of intersecting pairs of polygons in $\mathcal{P}$.

Hence, we obtain the following result.

Theorem 2.5. *Let $\mathcal{P}$ be a set of m convex polygons in the plane with a total of n vertices. All k pairs of indices (i, j) such that P_i intersects P_j can be reported in $O(n^{4/3}\log^{2+\varepsilon} n + k)$ time, for any $\varepsilon > 0$.*

Remark 2.6. As in Agarwal and Sharir [5], we can use a more sophisticated data structure to improve the running time of the algorithm to $O(m^{2/3}n^{2/3}\log^{c} n + n^{1+\varepsilon} + k)$, for an appropriate constant c and for any $\varepsilon > 0$.

The data structure by Matoušek in [19] can count the number of points lying inside a k-gon in time $O(k(m^{2/3}/n^{1/3}) \cdot \log^2 m)$ time using $O(m^{2/3}n^{2/3}\log^{2+\varepsilon} m + m^{1+\varepsilon})$ preprocessing. Moreover, a minor variant of the algorithm of Chazelle [10] can count, in $O(n^{4/3}\log n)$ time, the number of intersection points between L and U that correspond to the rightmost intersection points of the corresponding polygons. Hence, we obtain the following.

Theorem 2.7. *Let $\mathcal{P}$ be a set of convex polygons in the plane with a total of n vertices. The number of pairs of indices (i, j) such that P_i intersects P_j can be counted in $O(n^{4/3}\log^{2+\varepsilon} n)$ time, for any $\varepsilon > 0$.*

3. The three-dimensional case

Let $\mathcal{P} = \{P_1, \ldots, P_m\}$ be a set of m convex polytopes in $\mathbb{R}^3$ with a total of n vertices. We present an algorithm, with running time $O(n^{8/5+\varepsilon} + k)$, for any $\varepsilon > 0$, which reports all k pairs of indices (i, j) such that P_i intersects P_j. Our approach is similar to the algorithm described in Section 2.2. We compute the *bottom* vertex, i.e., the vertex with the minimum z-coordinate, of each nonempty intersection polytope $P_{ij} = P_i \cap P_j$, and report the corresponding pairs (i, j). The bottom vertex of an intersection polytope P_{ij} is the bottom vertex of P_i, the bottom vertex of P_j, the intersection point of an edge of P_i and a face of P_j, or the intersection point of a face of P_i and an edge of P_j. In the two latter cases, the intersection has to satisfy a few additional properties, which we describe and exploit below.

Let b_i be the bottom vertex of P_i, and let $V = \{b_i \mid 1 \leqslant i \leqslant m\}$. We first report all pairs (i, j) such that the bottom vertex of P_{ij} is the bottom vertex of P_i or of P_j. A vertex $b_i \in V$ is the bottom vertex of P_{ij} if

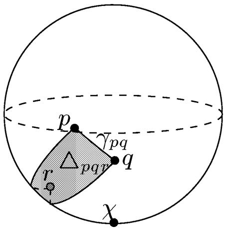

Fig. 1. An arc γ_{pq} and a spherical triangle $\triangle pqr$.

and only $b_i \in P_j$. Therefore, for each $P_j \in \mathcal{P}$, we need to compute and report $P_j \cap V$. As in Section 2.2, we can accomplish this in time $O(m^{3/4}n^{3/4}\log^c n + \mu)$, for some constant c, where $\mu = \sum_{i=1}^{m} |P_j \cap V| \leqslant k$, using the range-searching algorithm of Matoušek [19].

Next, we report all pairs (i, j) such that the bottom vertex of (the nonempty) P_{ij} is an edge-face intersection. Let E and F denote the sets of edges and of faces, respectively, of the polytopes in $\mathcal{P}$. Using the partition-tree data structure of Agarwal and Matoušek [3], we can compute, in $O(n^{8/5+\varepsilon})$ time, for any $\varepsilon > 0$, a family of pairs $\mathcal{F} = \{(E_1, F_1), \ldots, (E_r, F_r)\}$, such that

(i) $E_\alpha \subseteq E$ and $F_\alpha \subseteq F$, for all $1 \leqslant \alpha \leqslant r$;
(ii) every edge in E_α crosses every face of F_α, for all $1 \leqslant \alpha \leqslant r$;
(iii) for every crossing edge-face pair $(e, f) \in E \times F$, there is an α so that $e \in E_\alpha$ and $f \in F_\alpha$; and
(iv) $\sum_{\alpha=1}^{u}(|E_\alpha| + |F_\alpha|) = O(n^{8/5+\varepsilon})$.

We will describe an algorithm that, for a given pair (E_α, F_α), computes, in time $O((|E_\alpha| + |F_\alpha|)\log^2 n + \nu_i)$, all ν_i pairs $(e, f) \in E_\alpha \times F_\alpha$ such that $e \cap f$ is the bottom vertex of the corresponding intersection polytope. Repeating this procedure for all pairs of $\mathcal{F}$, we report, in time $O(n^{8/5+\varepsilon} + \nu)$ (for a slightly larger, but still arbitrarily small $\varepsilon > 0$), all ν pairs (i, j) such that the bottom vertex of P_{ij} is the intersection of an edge-face pair.

Consider a pair (E_α, F_α) from the family $\mathcal{F}$. For each edge $e \in E_\alpha$ (respectively, each face $f \in F_\alpha$), let $P_e \in \mathcal{P}$ (respectively, $P_f \in \mathcal{P}$) be the polytope containing e (respectively, f). Let $\mathbb{S}^2$ be the unit sphere of directions in $\mathbb{R}^3$, and let $\chi = (0, 0, -1)$ be the south pole of $\mathbb{S}^2$. For two points $p, q \in \mathbb{S}^2$ that are not antipodal, let $\gamma_{pq} \subset \mathbb{S}^2$ be the shorter arc of the great circle passing through p and q. For three points $p, q, r \in \mathbb{S}^2$, no two of which are antipodal, let $\triangle pqr$ be the smaller spherical triangle formed by the arcs γ_{pq}, γ_{qr}, and γ_{pr}. See Fig. 1.

Let $\mathbf{n}_f$ denote the outward unit normal of a face f. For an edge e, let γ_e be the great circular arc on $\mathbb{S}^2$ representing all outward normals to the planes supporting P_e at e. The endpoints ξ and η of γ_e are the outward normals of the faces of P_e incident upon e, and $\gamma_e = \gamma_{\xi\eta}$. For an edge $e \in E_\alpha$ and a face $f \in F_\alpha$, let $\tau_{ef} = \triangle\xi\eta\mathbf{n}_f$ be the spherical triangle formed by γ_e, $\gamma_{\xi\mathbf{n}_f}$, and $\gamma_{\eta\mathbf{n}_f}$; τ_{ef} is the set of outward normals supporting $P_e \cap P_f$ at the vertex $e \cap f$. The following lemma is straightforward but crucial to our analysis.

Lemma 3.1. *For a pair $(e, f) \in E_\alpha \times F_\alpha$, the intersection point $e \cap f$ is the bottom vertex of $P_e \cap P_f$ if and only if $\chi \in \tau_{ef}$.*

In order to find the edge-face pairs with the above property, we define a spherical triangle $\triangle_e$ for each edge $e \in E_\alpha$ as follows. Let p and q be the antipodal points of the endpoints of γ_e, and let $\overline{\gamma}_e$ be the antipodal arc of γ_e, i.e., the set of points that are antipodal to the points on γ_e. We define $\triangle_e$ to be the spherical triangle $\triangle pq\chi$, which is bounded by the arcs $\overline{\gamma}_e, \gamma_{p\chi}$, and $\gamma_{q\chi}$. We also define W_e to be the spherical wedge that contains the arc $\overline{\gamma}_e$ and is formed by the meridians passing through p and q. Finally, let H_e be the hemisphere containing $\triangle_e$ and bounded by the great circle containing γ_e and $\overline{\gamma}_e$ (this circle is the set of normals to the planes passing through the edge e). Then $\triangle_e = H_e \cap W_e$.

It can be easily checked that $\chi \in \tau_{ef}$ if and only if $\mathbf{n}_f \in \triangle_e$, which implies the following lemma.

Lemma 3.2. *For a given pair $(e, f) \in E_\alpha \times F_\alpha$, the intersection point $e \cap f$ is the bottom vertex of $P_e \cap P_f$ if and only if $\mathbf{n}_f \in \triangle_e$.*

Let $\Delta = \{\triangle_e \mid e \in E_\alpha\}$ and $N = \{\mathbf{n}_f \mid f \in F_\alpha\}$. For each $\triangle_e \in \Delta$, we wish to report $\triangle_e \cap N$. Recall that $\triangle_e = W_e \cap H_e$. We thus preprocess N into a two-level data structure—the first level reports, for any query $\triangle_e$, all points of $W_e \cap N$ as the union of $O(\log|F_\alpha|)$ canonical subsets, and the second level reports all points of the canonical subsets that lie inside H_e. More precisely, we proceed as follows. We sort the points in N by their longitudes and construct a minimum-height binary tree T on the sorted point set (we omit the easy details concerning the handling of the circularity of this order). Each node u of T is associated with the subset $N_u \subseteq N$ of points that are stored at the leaves of the subtree rooted at u. We preprocess N_u for hemisphere reporting queries, where each query reports all points of N_u lying inside a query hemisphere $H \subset \mathbb{S}^2$. By using a halfplane range-reporting structure [12], we can preprocess N_u, in $O(|N_u|\log|N_u|)$ time, into a data structure of size $O(|N_u|)$, so that a hemisphere query can be answered in $O(\log|N_u| + t)$ time, where t is the output size. We attach this structure at u as its secondary structure. The total time spent in preprocessing N is $O(|F_\alpha|\log^2|F_\alpha|)$. For an edge $e \in E_\alpha$, we report $\triangle_e \cap N$ as follows. By searching with the longitudes of the endpoints of $\overline{\gamma}_e$, we first find, in $O(\log|F_\alpha|)$ time, a set U_e of $O(\log|F_\alpha|)$ nodes of T, so that $\bigcup_{u \in U_e} N_u = W_e \cap N$. For each node $u \in U_e$, we report all t_u points of $N_u \cap \triangle_e$ in $O(\log|F_\alpha| + t_u)$ time, by searching with H_e in the secondary structure attached to u. Therefore the total time spent in reporting all t_e points of $\triangle_e \cap N$ is $O(\log^2|F_\alpha| + t_e)$. Hence, the overall time spent in reporting all ν pairs of $E_\alpha \times F_\alpha$ such that $e \cap f$ is the bottom vertex of $P_e \cap P_f$ is $O((|E_\alpha| + |F_\alpha|)\log^2|F_\alpha| + \nu)$.

Summing up all the bounds, and replacing ε by a slightly larger (but still arbitrarily small) constant, we obtain the following.

Theorem 3.3. *Given a set $\mathcal{P}$ of m convex polytopes in $\mathbb{R}^3$ with a total of n vertices, we can report all k pairs of indices (i, j) such that P_i and P_j intersect, in time $O(n^{8/5+\varepsilon} + k)$, for any constant $\varepsilon > 0$.*

By replacing the halfplane range-reporting structure at each node $u \in T$ with a halfplane range-counting structure, we can count all intersecting pairs of polytopes in $\mathcal{P}$. Using Matoušek's partition-tree data structure once again, we can preprocess N_u in time $O((|E_\alpha|^{2/3}|N_u|^{2/3} + |N_u|)\log n)$ so that a halfplane range-counting query can be answered in $O((|N_u|^{2/3}/|E_\alpha|^{1/3})\log n)$ time. The total time spent in preprocessing N, summed over all nodes of T, is $O((|E_\alpha|^{2/3}|F_\alpha|^{2/3} + |F_\alpha|)\log^2 n)$, and for an edge $e \in E_\alpha$, $|\triangle_e \cap N|$ can be computed in time $O((|F_\alpha|^{2/3}/|E_\alpha|^{1/3})\log n)$ by traversing T as above. Putting every thing together, the time spent in counting the number of pairs in $E_\alpha \times F_\alpha$ so that $e \cap f$ is the bottom vertex of $P_e \cap P_f$ is $O((|E_\alpha|^{2/3}|F_\alpha|^{2/3} + |F_\alpha|)\log^2 n)$. As shown in [2], the properties of multi-level partition trees imply that $\sum_{i=1}^{u} |E_\alpha|^{2/3}|F_\alpha|^{2/3} = O(n^{8/5+\varepsilon})$. Hence, the total running time of the algorithm, summed over all pairs in $\mathcal{F}$, is $O(n^{8/5+\varepsilon'})$, for any $\varepsilon' > \varepsilon$. We thus conclude the following.

Theorem 3.4. *Given a set $\mathcal{P}$ of m convex polytopes in $\mathbb{R}^3$ with a total of n vertices, we can count all pairs of indices (i, j) such that P_i and P_j intersect, in time $O(n^{8/5+\varepsilon})$, for any constant $\varepsilon > 0$.*

4. Conclusions

In this paper, we presented output-sensitive algorithms for reporting all intersecting pairs among a set of convex polygons in the plane, and among a set of convex polytopes in three dimensions. For the planar case, we presented the first near-linear-time algorithm for this problem; its expected running time is $O((n \log m + k)\alpha(n) \log n)$. We also proposed a deterministic algorithm with running time $O(n^{4/3} \log^{2+\varepsilon} n + k)$. Our algorithm for the 3-dimensional case runs in $O(n^{8/5+\varepsilon} + k)$ time.

One can also consider the bichromatic version of the problem. Here one is given two sets of polytopes—the blue polytopes and the red polytopes, say—and the goal is to report all bichromatic pairs of intersecting polytopes. The challenge is to avoid spending time on the monochromatic intersections. It seems hard to generalize our near-linear-time algorithm for the planar case to the bichromatic problem, but the generalization of the deterministic algorithms for the planar and the three-dimensional case is straightforward.

An open question is whether there exists an $O(m^{2-\varepsilon} + k)$-time algorithm, where $\varepsilon > 0$ is a constant, for reporting all k pairs of intersecting polytopes in a set $\mathcal{P}$ of m convex polytopes in $\mathbb{R}^4$.

References

[1] P.K. Agarwal, Partitioning arrangements of lines. II. Applications, Discrete Comput. Geom. 5 (1990) 533–573.
[2] P.K. Agarwal, J. Erickson, Geometric range searching and its relatives, in: B. Chazelle, J.E. Goodman, R. Pollack (Eds.), Advances in Discrete and Computational Geometry, American Mathematical Society, Providence, RI, 1999, pp. 1–56.
[3] P.K. Agarwal, J. Matoušek, On range searching with semialgebraic sets, Discrete Comput. Geom. 11 (1994) 393–418.
[4] P.K. Agarwal, M. Sharir, Red-blue intersection detection algorithms, with applications to motion planning and collision detection, SIAM J. Comput. 19 (1990) 297–321.
[5] P.K. Agarwal, M. Sharir, Ray shooting amidst convex polygons in 2D, J. Algorithms 21 (1996) 508–519.
[6] I. Balaban, An optimal algorithm for finding segment intersections, in: Proc. 11th Annu. ACM Sympos. Comput. Geom., 1995, pp. 211–219.
[7] J. Basch, L.J. Guibas, G. Ramkumar, Reporting red–blue intersections between two sets of connected line segments, in: J. Diaz, M. Serna (Eds.), Algorithms—ESA'96, Springer, Berlin, 1996, pp. 302–319.
[8] J.L. Bentley, T.A. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. C-28 (1979) 643–647.
[9] G. Brodal, R. Jacob, Dynamic planar convex hull with optimal query time and $O(\log n \log\log n)$ update time, in: Proc. 7th Scand. Workshop Algorithm Theory, 2000, pp. 57–70.
[10] B. Chazelle, Cutting hyperplanes for divide-and-conquer, Discrete Comput. Geom. 9 (1993) 145–158.
[11] B. Chazelle, L. Guibas, Fractional cascading: I. A data structuring technique, Algorithmica 1 (1986) 133–162.
[12] B. Chazelle, L. Guibas, D.T. Lee, The power of geometric duality, BIT 25 (1985) 76–90.
[13] B. Chazelle, H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, J. ACM 39 (1992) 1–54.
[14] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, Algorithmica 11 (1994) 116–132.

[15] K.L. Clarkson, P. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387–421.
[16] U. Finke, K. Hinrichs, Overlaying simply connected planar subdivisions in linear time, in: Proc. 11th Annu. ACM Sympos. Comput. Geom., 1995, pp. 119–126.
[17] P. Gupta, R. Janardan, M. Smid, Efficient algorithms for counting and reporting pairwise intersections between convex polygons, Inform. Process. Lett. 69 (1999) 7–13.
[18] S. Har-Peled, M. Sharir, On-line point location in planar arrangements and its applications, Discrete Comput. Geom. 26 (2001) 19–40.
[19] J. Matoušek, Range searching with efficient hierarchical cuttings, Discrete Comput. Geom. 10 (1993) 157–182.
[20] K. Mulmuley, A fast planar partition algorithm. I, J. Symbolic Comput. 10 (1990) 253–280.
[21] N. Sarnak, R.E. Tarjan, Planar point location using persistent search trees, Comm. ACM 29 (1986) 669–679.
[22] M. Sharir, P.K. Agarwal, Davenport–Schinzel Sequences and Their Geometric Applications, Cambridge University Press, New York, 1995.
[23] S. Suri, P.M. Hubbard, J.F. Hughes, Analyzing bounding boxes for object intersection, ACM Trans. Graphics 18 (1999) 257–277.