


Subquadratic Algorithms for Some 3Sum-Hard Geometric Problems in the Algebraic Decision Tree Model

Boris Aronov ✉ 

Tandon School of Engineering, New York University, Brooklyn, NY, USA

Mark de Berg ✉ 

Eindhoven University of Technology, The Netherlands

Jean Cardinal ✉ 

Université libre de Bruxelles (ULB), Brussels, Belgium

Esther Ezra ✉ 

School of Computer Science, Bar Ilan University, Ramat Gan, Israel

John Iacono ✉ 

Université libre de Bruxelles (ULB), Brussels, Belgium

Tandon School of Engineering, New York University, Brooklyn, NY, USA

Micha Sharir ✉ 

School of Computer Science, Tel Aviv University, Israel

Abstract

We present subquadratic algorithms in the algebraic decision-tree model for several 3SUM-hard geometric problems, all of which can be reduced to the following question: Given two sets A, B , each consisting of n pairwise disjoint segments in the plane, and a set C of n triangles in the plane, we want to count, for each triangle $\Delta \in C$, the number of intersection points between the segments of A and those of B that lie in Δ . The problems considered in this paper have been studied by Chan (2020), who gave algorithms that solve them, in the standard real-RAM model, in $O((n^2/\log^2 n) \log^{O(1)} \log n)$ time. We present solutions in the algebraic decision-tree model whose cost is $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$.

Our approach is based on a primal-dual range searching mechanism, which exploits the multi-level polynomial partitioning machinery recently developed by Agarwal, Aronov, Ezra, and Zahl (2020).

A key step in the procedure is a variant of point location in arrangements, say of lines in the plane, which is based solely on the *order type* of the lines, a “handicap” that turns out to be beneficial for speeding up our algorithm.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, Algebraic decision-tree model, Polynomial partitioning, Primal-dual range searching, Order types, Point location, Hierarchical partitions

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.3

Related Version *Full Version:* <https://arxiv.org/abs/2109.07587>

Funding *Boris Aronov:* Partially supported by NSF Grants CCF-15-40656 and CCF-20-08551, and by Grant 2014/170 from the US-Israel Binational Science Foundation.

Mark de Berg: Partially supported by the Dutch Research Council (NWO) through Gravitation Grant NETWORKS (project no. 024.002.003).

Jean Cardinal: Partially supported by the F.R.S.-FNRS (Fonds National de la Recherche Scientifique) under CDR Grant J.0146.18.

Esther Ezra: Partially supported by NSF CAREER under Grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation.

John Iacono: Partially supported by Fonds National de la Recherche Scientifique (FNRS) under Grant no. MISU F.6001.1.



© Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir; licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 3; pp. 3:1–3:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Micha Sharir: Partially supported by ISF Grant 260/18, by Grant 1367/2016 from the German-Israeli Science Foundation (GIF), and by Blavatnik Research Fund in Computer Science at Tel Aviv University.

Acknowledgements We thank Zuzana Patáková for helpful discussions on multilevel polynomial partitioning.

1 Introduction

Let A and B be two sets, each consisting of n pairwise disjoint line segments in the plane, and let C be a set of n triangles in the plane. We study the problem of counting, for each triangle $\Delta \in C$, the number of intersection points between the segments of A and those of B that lie inside Δ . We refer to this problem as *within-triangle intersection-counting*. This is one of four 3SUM-hard problems (among many others) studied by Chan [11], all of which can be reduced to the problem just mentioned.¹ The other three problems are:²

- (i) *Intersection of three polygons*. Given three simple n -gons A, B, C in the plane, determine whether $A \cap B \cap C$ is nonempty.
- (ii) *Coverage by three polygons*. Given three simple n -gons A, B, C in the plane, determine whether $A \cup B \cup C$ covers a given triangle Δ_0 .
- (iii) *Segment concurrency*. Given sets A, B, C , each consisting of n pairwise disjoint segments in the plane,³ determine whether $A \times B \times C$ contains a concurrent triple.

Chan [11] presents slightly subquadratic algorithms for all four problems, whose running time in the standard real-RAM model (also referred to as the *uniform* model) is $O((n^2/\log^2 n) \log^{O(1)} n)$. He has observed that, as already mentioned, all these problems can be reduced in near-linear time to the within-triangle intersection-counting problem, so it suffices to present an efficient subquadratic solution for that problem.

We study the within-triangle intersection-counting problem in the algebraic decision-tree model. In this model only sign tests of polynomial inequalities of constant degree that access explicitly (the endpoint coordinates of) the input segments or vertices of the input triangles count towards the running time. All other operations cost nothing in the model, but are not allowed to access the input segments explicitly. Although originally introduced for establishing lower bounds [7], the algebraic decision-tree model has become a standard model for upper bounds too, used in the study of many problems, including the 3SUM-problem itself [10, 17, 20, 23, 25] and various 3SUM-hard geometric problems [5, 6, 17]. One can interpret the decision-tree model as an attempt to isolate and minimize the cost of the part of the algorithm that explicitly accesses the real representation of the input objects, and ignore the cost of the other purely discrete steps. This has the potential of providing us with an insight about the problem complexity, which might eventually lead to an improved solution also in the uniform real-RAM model.

We show that the within-triangle intersection-counting problem and, hence, also problems (i)–(iii), can be solved in this model with $O(n^{60/31+\varepsilon})$ sign tests, for any $\varepsilon > 0$. Chan [11] also remarks (without providing details) that his algorithm can be implemented in $O(n^{2-\delta})$ time in the algebraic decision-tree model, for some $\delta > 0$ that he left unspecified. (With some care, as was communicated to us, one can obtain $\delta \approx 0.01$.) Our algorithm is rather

¹ Chan [11] refers to this problem as “triangle intersection-counting.”

² The fact that these problems are 3SUM-hard, and the connections between them, are stated in [11].

³ The segments of one set, say C , need not be pairwise disjoint. Although not explicitly stated, the technique in [11] for the uniform model can also handle this situation.

different from Chan's, and gives a concrete value for δ (any positive $\delta < 2/31$), as mentioned above. Our techniques appear to be of independent interest and to have the potential to apply to other problems, as we demonstrate in the full version [4, Section 4].

If the segments in A and B and the triangles in C were all full lines,⁴ then, in general, determining the existence of a concurrent triple of lines in $A \times B \times C$ (the so-called *concurrency testing* problem) is the dual version of the classical 3SUM-hard *collinearity testing* problem, in which we are given three sets of points in the plane, and wish to determine whether their Cartesian product contains a collinear triple. This problem has recently been studied in the algebraic decision-tree model by Aronov et al. [5], in a restricted version where two of the sets are assumed to lie on two constant-degree algebraic curves, where an algorithm with roughly $O(n^{28/15})$ comparisons has been presented.

The problems studied here can be regarded as other dual versions of collinearity testing, where restrictions of a different kind are imposed. As noted by Chan [11], the additional disjointness properties that are assumed here make the problem simpler than collinearity testing (albeit by no means simple), and its solution appears to have no bearing on the unconstrained collinearity problem itself. In the full version [4, Section 5] we comment on the substantial differences between this work and the work by Aronov et al. [5].

Our technique is based on hierarchical cuttings of the plane, as well as on tools and properties of segment-intersection range searching. We also use the so-called Fredman's trick in algebraic-geometric settings, in which the problem is mapped into a primal-dual range searching mechanism involving points and surfaces in \mathbb{R}^6 . This reduction exploits the very recent multi-level polynomial partitioning technique of Agarwal et al. [2] (or a similar technique of Matoušek and Patáková [27]). Our range-searching mechanism of points and algebraic surfaces in higher dimensions is a by-product of our analysis, which appears to be broadly applicable in other range-searching contexts, and we thus regard it as a technique of independent interest; see, for example, Proposition 2 and its proof.

Point location in arrangements. An additional key ingredient of our approach involves point location in an arrangement of lines in the plane (or an arrangement of curves, or of hyperplanes in higher dimensions). This is of course a well studied problem with several optimal solutions [29], but we adapt and use techniques that are handicapped by the requirement that each operation that examines the real parameters specifying the lines involves at most *three* input lines. In contrast, the persistent data structure of [29], for example, needs to sort the vertices of the arrangement from left to right, thus requiring comparisons of the x -coordinates of a pair of vertices, which are in general determined by the parameters of *four* input lines. The persistent data structure method has been used in [5, 11] for the study of other 3SUM-hard geometric problems. Here we replace this approach with one that uses solely the relative positions of triples of lines, the so-called *order type* of the arrangement. In this approach each comparison involves only *three* input lines, which, as we show, eventually leads to improved performance of the algorithm.

In standard settings, separating the order-type computation from the rest of the processing makes no sense. This is because obtaining the full order-type information for N lines already takes $\Theta(N^2)$ time. This makes the approach based on the order type noncompetitive, as one can just do point location in the line arrangement, in the uniform model, with $O(N^2)$ preprocessing. Nevertheless, in the applications considered in this paper (see Section 3 and

⁴ Disjointness then of course cannot be assumed in general, although it might occur when the lines in each set are parallel, as in the dual version of the 3SUM-hard GEOMBASE problem [19].

the full version [4, Section 4]), the input lines have a special representation, which allows us to avoid an explicit construction of their order type and obtain this information implicitly in subquadratic time in the decision-tree model. The rest of the preprocessing, which takes quadratic time and storage in the uniform model, costs nothing in the decision-tree model.

The problem of determining whether and how the order type of an arrangement is sufficient to construct an efficient point-location data structure has, to the best of our knowledge, never been addressed explicitly. As we believe that this kind of “handicapped” point location will be useful for other applications (some of which are mentioned in the full version [4, Section 4]), we present it in some detail in Section 2 and in the full version [4, Section 2.2]. We also present extensions of this technique to arrangements of constant-degree algebraic curves in \mathbb{R}^2 , and to arrangements of planes or hyperplanes in higher dimensions, which will be used in the applications presented in the full version [4, Section 4].

The algorithm for solving the within-triangle intersection-counting problem in the algebraic decision-tree model, and, consequently, also of the other three problems listed at the beginning of this section, is presented in Section 3. Additional applications of our technique are presented in the full version [4, Section 4]; they include: (i) counting intersections between two sets of pairwise disjoint circular arcs inside disks, and (ii) minimum distance problems between lines and two sets of points in the plane.

2 Order-type-based point location in arrangements

Order types. An arrangement of non-vertical lines in the plane (and, later, curves in the plane, or hyperplanes in higher dimension) can be described in the following combinatorial fashion. We use the notion of an *order type*, defined for a set L of lines as follows: Given any ordered triple of lines (ℓ_1, ℓ_2, ℓ_3) from L , where both ℓ_2 and ℓ_3 intersect ℓ_1 , we record the left-to-right order of the intersections $\ell_1 \cap \ell_2$ and $\ell_1 \cap \ell_3$ along ℓ_1 ; note that the intersections might coincide. The totality of this information gives, for each line in L , the left-to-right order of its intersections with every other line it meets. Furthermore, we assume the existence of an “infinitely steep” line ℓ_∞ , placed sufficiently far to the left, the order of whose intersections with the “normal” lines encodes the order of their slopes. This information is dual to the perhaps more familiar notion of an order type for a set of points in the plane (see, e.g., [21]). A higher-dimensional analog of this information involves recording the order in which a line that is the intersection of $d - 1$ hyperplanes in \mathbb{R}^d meets the remaining hyperplanes that meet but do not contain it. We also assume a suitable analog of the “infinitely steep line,” recursively defined over the dimension.

Back in the plane, the permutations along each line of the intersection points with the other lines are called *local sequences* [22]. This view allows us to extend the definition of the order type to x -monotone curves, where each pair of curves is assumed to intersect at most s points, for some constant s . In this case the order type gives, for each curve γ in the collection, the labeled left-to-right sequence of intersection points with the other curves, where each intersection point is labeled by the triple (i, j, k) , where i and j are the indices of the two curves that form the intersection, and k indicates that it is the k th leftmost intersection point of the two curves. The order type also includes the vertical order of the curves at $x = -\infty$. See the full version [4, Section 2.2] for further details.

The significance of the order type is that (a) it only records information for $(d + 1)$ -tuples of objects, and (b) it contains enough information that lets us construct the arrangement and preprocess it for fast point location, without having to access further the actual parameters that define the objects.

The problem we tackle now is the following: Given the order type of an arrangement, preprocess this information into a point location data structure. The preprocessing stage is not allowed to access the actual geometric description of the objects, such as the coefficients of the equations defining the lines or hyperplanes, but can only exploit the discrete data given by the order type. A query, in contrast, is allowed to examine the coefficients of the few objects that it encounters.

We present two solutions for this problem. First, we show that, for d -dimensional hyperplane arrangements, for any $d \geq 2$, the sampling method of Meiser [28] (see also [16]) can be implemented using only order-type information. Second, we show that for arrangements of x -monotone curves in the plane, a simple variant of the separating-chain method for point location [15, 26] can be implemented such that only order-type information is used during the preprocessing. We present only the first technique in this version of the paper, and delegate the second one to the full version [4, Section 2.2].

2.1 Sampling-based approach for hyperplane arrangements

Let H be a set of N non-vertical hyperplanes in \mathbb{R}^d , where $d \geq 2$ is a fixed constant. We want to construct a point-location data structure for the arrangement $\mathcal{A}(H)$ induced by H , where we are only given the order type of H . Essentially, we are given, for each intersection line formed by $d - 1$ hyperplanes, the order of its intersections with the other hyperplanes. (Alternatively, we are given, for each simplex σ formed by $d + 1$ of the hyperplanes, the x_1 -order of the vertices of σ .) We only require H not to contain vertical hyperplanes. We do permit more than d hyperplanes to share a point, as well as other degeneracies. This is indeed a natural scenario for our applications including segment intersection counting and its related problems.

We briefly sketch the randomized method first proposed by Meiser [28] and analyzed in detail by Ezra et al. [16] (see also [10]), and show that the order-type information is sufficient to construct the data structure.

Before considering the point-location structure, we note that the order type suffices to construct a discrete representation of the arrangement $\mathcal{A}(H)$, in which each j -dimensional cell of $\mathcal{A}(H)$, for $j = 1, \dots, d$, stores the set of all $(j - 1)$ -dimensional cells that form its boundary (and consequently of all cells, of all dimensions, on its boundary), with respective back pointers from each cell to all higher-dimensional cells that contain it on their boundary. This can be done, e.g., by the Folkman–Lawrence topological representation theorem for oriented matroids [18], which, roughly speaking, implies that, given the order type of H , one can construct a combinatorial representation for the arrangement $\mathcal{A}(H)$, consisting of all *sign conditions*. That is, each face f of $\mathcal{A}(H)$ (of any dimension) is encoded by a sign vector $\{-1, 0, +1\}^{|H|}$ representing the above/below/on relation of f with respect to each hyperplane in H ; see [9] for an inductive proof for the planar case, and its generalization to higher dimensions in [8]. Given this property, a naïve actual construction of the combinatorial representation of $\mathcal{A}(H)$ is easy to derive, and is free of charge in the decision-tree model, once the order type of H is computed. When we perform a point-location query we report a pointer to the sign vector of the cell of $\mathcal{A}(H)$ that contains the query point – see below.

Preprocessing. Given the arrangement $\mathcal{A}(H)$ and a fixed $\varepsilon > 0$, we first construct a random sample S of $O(\frac{d^2}{\varepsilon} \log \frac{d}{\varepsilon})$ hyperplanes of H ; the size of S does not depend on n . We then compute a *canonical* triangulation of the arrangement $\mathcal{A}(S)$. For each face of $\mathcal{A}(S)$, of any dimension at least 2, we use a fixed rule to designate a *reference vertex* p of this face. For example, we can take p to be the lexicographically smallest vertex of the face,

with each vertex represented by the lexicographically smallest d -tuple of the indices of the hyperplanes that contain it and whose intersection is a single point.⁵ We then triangulate each face f of $\mathcal{A}(S)$ by the *fan* obtained by adding the vertex p to each simplex in the triangulations of the lower-dimensional faces composing the boundary of f and not incident to p . Next, we construct the *conflict list* $L(\Delta)$ for each simplex Δ of the triangulation, of any dimension, defined as the set of hyperplanes of H that *cross* Δ , i.e., intersect, but not fully contain, it. $L(\Delta)$ can indeed be constructed using only the order type: Deciding whether a hyperplane $h \in H$ belongs to $L(\Delta)$ amounts to testing whether there exist two vertices of Δ that lie on different sides of h , and each such test is an orientation test of the corresponding $(d + 1)$ -tuple of hyperplanes: h and the d hyperplanes forming the vertex.

From standard results on ε -nets [24], a suitable choice of the constant of proportionality in the bound on the sample size guarantees that, with high probability, the conflict list size is not larger than εn , for each simplex Δ . It remains to recurse, for each simplex Δ of the triangulation, with the hyperplanes in $L(\Delta)$. (If Δ is not full-dimensional, we also record the hyperplanes containing it and the recursive processing involves building a lower-dimensional arrangement within Δ .) This leads to a hierarchical data structure in which the number of hyperplanes decreases by a factor ε at each level. The construction continues until the number of hyperplanes falls below a suitable constant, at which point we simply store the remaining hyperplanes. Let w be a leaf in this hierarchy. It will be convenient to further preprocess the set $H(w)$ of hyperplanes stored at w into a tree \mathcal{T}_w that allows us to locate a query point in the arrangement $\mathcal{A}(H(w))$. The structure \mathcal{T}_w is simply a ternary tree of depth $|H(w)| = O(1)$, where a node at level j stores the j -th hyperplane h_j of $H(w)$, so we can test if a query point is below, on, or above h_j . Observe that each leaf of \mathcal{T}_w corresponds to a unique cell in the arrangement $\mathcal{A}(H(w))$ and, hence, also in $\mathcal{A}(H)$ – indeed, the sign with respect to every hyperplane in $H \setminus H(w)$ is determined by the search path to the node w in the hierarchy.

Answering queries. Queries are answered as follows. First, we locate the (open) simplex Δ of the canonical triangulation of $\mathcal{A}(S)$ containing the query point q . Since d is assumed to be constant, S is also of constant size, and so locating Δ can be done in $O(1)$ time. Next, we recurse in the data structure attached to Δ . When we reach a leaf w of the hierarchy, we continue to search in the tree \mathcal{T}_w . When we reach a leaf in \mathcal{T}_w we have located q and can report (a pointer to) the sign vector of the cell containing q .

The overall number of these recursive steps is $O(\log n)$, and thus answering a query costs $O(\log n)$ arithmetic operations, where the hidden constant⁶ is polynomial in d . As noted, in our applications we only need to determine whether q lies on a hyperplane of H .

The following lemma summarizes the result.

► **Lemma 1.** *Let H be a set of n hyperplanes in \mathbb{R}^d , where $d \geq 2$ is a constant. Using only the order type of H , we can construct a polynomial-size data structure that guarantees $O(\log n)$ -time point-location queries in the arrangement $\mathcal{A}(H)$; the implied constant depends polynomially on d . The (polynomial) preprocessing time and the storage of the data structure cost nothing in the decision-tree model.*

⁵ When p is chosen as the bottommost vertex, the resulting triangulation is referred to as the *bottom-vertex triangulation*, but in general the order type does not provide us with this information.

⁶ The value of this constant depends on the storage allocated to the structure. For example, spending $n^{2d \log d + O(d)}$ on storage guarantees query cost of $O(d^4 \log n)$ [16].

Remark. The query time for $d = 2$ is better than the time in the second, level-based approach presented in the full version [4, Section 2.2] for curves in the plane (which is $O(\log^2 n)$). However, the sampling-based method does not extend to non-straight curves, since there is no obvious way to extend the notion of a canonical triangulation to the case of curves. The only viable way of doing this seems to use the standard vertical-decomposition technique. Unfortunately (for us), constructing the vertical decomposition requires that we compare the x -coordinates of vertices defined by different, unrelated pairs of curves. Such a comparison involves *four* input curves and it cannot be resolved from the order-type information alone. For lines in the plane, however, the above technique does yield the improved logarithmic query time.

3 The algorithm for within-triangle intersection-counting

Our input consists of two sets A, B , each of n pairwise disjoint segments in the plane, and of a set C of n triangles in the plane. To simplify the presentation, we assume that the input is in general position, namely that, among the segments of A, B , and edges of triangles of C , no two share a supporting line, and no endpoint of one segment lies on another (with the obvious exception of the vertices of a triangle in C).⁷ These are the only general position assumptions that we need. A triple of segments (one from A , one from B , and an edge of a triangle from C) are allowed to be concurrent.

A high-level roadmap of the algorithm. To avoid various technical issues that complicate the description of our algorithm, we focus in this overview on the simpler segment concurrency problem, where C is a set of (not necessarily disjoint) segments, and the goal is to determine whether there is a triple $(a, b, c) \in A \times B \times C$ of concurrent segments. To make the overview even simpler, assume that C is a set of lines.

We fix a parameter $g \ll n$ and put $r := n/g$. We construct a $(1/r)$ -cutting $\Xi(A)$ for the segments of A , and another such cutting $\Xi(B)$ for the segments of B . Since the segments of A are pairwise disjoint, we can construct $\Xi(A)$ of size $O(r)$, and similarly for $\Xi(B)$ (see [14]). We overlay the two cuttings and obtain a planar decomposition Ξ . While the complexity of Ξ is $O(r^2)$, any line of C crosses only $O(r)$ of its cells.

For each two-dimensional cell σ of Ξ (lower-dimensional cells are simpler to handle), we preprocess the sets $A_\sigma \subseteq A$ and $B_\sigma \subseteq B$ of those segments that cross σ , each of size at most $n/r = g$, into a data structure that supports efficient queries, each specifying a line c and asking whether c passes through an intersection point of a segment of A_σ and a segment of B_σ . We pass to the dual plane, obtain sets A_σ^* and B_σ^* of at most g points (dual to the lines containing the segments) each. (We ignore here “short” segments that have an endpoint inside σ ; see below.) The query is a point c^* and the task is to determine whether c^* is collinear with a pair of points $(a^*, b^*) \in A_\sigma^* \times B_\sigma^*$. For $a \in A_\sigma$ and $b \in B_\sigma$ we define $\gamma_{a,b}$ to be the line that passes through a^* and b^* , and let Γ_σ denote the collection of these lines. The query with c^* then reduces to point location in the arrangement $\mathcal{A}(\Gamma_\sigma)$, where we only need to know whether c^* lies on any of the lines.

We cannot perform this task explicitly in an efficient manner, since the complexity of $\mathcal{A}(\Gamma_\sigma)$ is $O(g^2)$ and we have $O(r^2) = O(n^2/g^2)$ such arrangements, of overall size $O(n^2)$. We can do it, though, in the algebraic decision-tree model, in an implicit manner, using the so-called *Fredman’s trick*; see [23] for a simpler yet representative application of Fredman’s

⁷ For technical reasons, we also allow a triangle in C to degenerate to a segment.

trick, as well as [5, 6] for geometric applications of Fredman’s trick. Concretely, we apply the order-type-based machinery of Section 2 to construct $\mathcal{A}(\Gamma_\sigma)$ and preprocess it for fast point location. More precisely, we first construct the order type of Γ_σ : this involves, for each triple of lines $\gamma_{a_1,b_1}, \gamma_{a_2,b_2}, \gamma_{a_3,b_3}$, determining the ordering of their intersection points along each of these lines. We express this test, in a straightforward manner, as the sign test of some 12-variate constant-degree polynomial $G(a_1, a_2, a_3; b_1, b_2, b_3)$.

We map the triple (b_1, b_2, b_3) to a point in a six-dimensional parametric space, and (a_1, a_2, a_3) to an algebraic surface ψ_{a_1,a_2,a_3} in this space, which is the locus of all triples (b_1, b_2, b_3) with $G(a_1, a_2, a_3; b_1, b_2, b_3) = 0$. We now need to locate the points (b_1, b_2, b_3) in the arrangement of the surfaces ψ_{a_1,a_2,a_3} , from which all the sign tests can be resolved, at no extra cost in the algebraic decision-tree model, thereby yielding the desired order type. The subsequent construction of the arrangement $\mathcal{A}(\Gamma_\sigma)$, and its preprocessing for fast point location, using the machinery in Section 2, also cost nothing in our model.

To make this process efficient, we group together all the points (b_1, b_2, b_3) , for b_1, b_2, b_3 in the same cell σ , over all cells σ , into one global set P , and group the surfaces ψ_{a_1,a_2,a_3} into another global set Ψ . We have $|P|, |\Psi| = O(r) \cdot O(g^3) = O(ng^2)$ (since there are only $O(r)$ cells of $\Xi(A)$ (resp., of $\Xi(B)$) from which the triples (a_1, a_2, a_3) (resp. (b_1, b_2, b_3)) are drawn).

Using the recent machinery of Agarwal et al. [2] or of Matoušek and Patáková [27], we can perform this batched point location in 6-space in time

$$O\left(|P|^{6/7+\varepsilon} |\Psi|^{6/7+\varepsilon} + |P|^{1+\varepsilon} + |\Psi|^{1+\varepsilon}\right) = O\left((ng^2)^{12/7+2\varepsilon}\right),$$

for any $\varepsilon > 0$. Full details of this step, crucial, albeit rather technical, are given in the full version [4, Section 3.1].

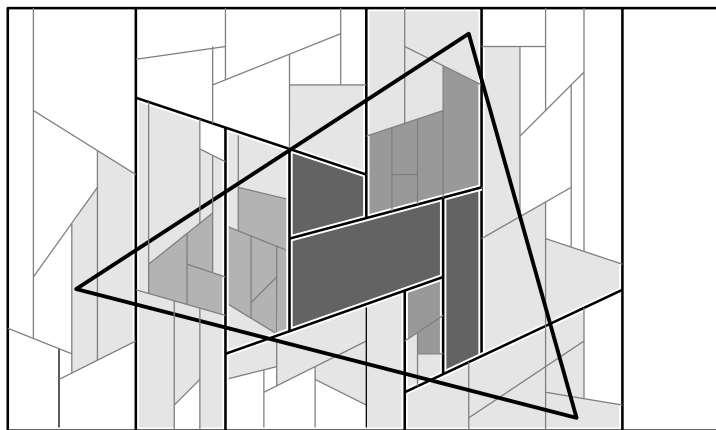
Searching with the dual points c^* takes $O\left(\frac{n^2}{g} \log g\right)$ time, because we have n query lines c , each line crosses $O(r) = O(n/g)$ cells σ , and each point location with c^* in each of the encountered arrangements takes $O(\log g)$ time. Balancing (roughly) this cost with the preprocessing cost, we choose $g = n^{2/31}$, and obtain the total subquadratic running time $O(n^{2-2/31+\varepsilon}) = O(n^{60/31+\varepsilon})$.

Quite a few issues were glossed over in this overview. Since the segments of A and of B are bounded, a cell σ may contain endpoints of these segments, making the passage to the dual plane more involved. The same applies in the original within-triangle intersection-counting problem, where the triangles of C may have vertices or more than one bounding edge that lie in or meet σ . We thus need to handle the presence of such “short” segments and/or “short” triangles. Moreover, we need to count intersection points within each triangle, and the number of cells in overlay of the cuttings $\Xi(A), \Xi(B)$ that a triangle can fully contain is much larger than $O(r)$. All these issues require more involved techniques, which are developed below, with some details delegated to the full version [4, Section 3]. Still, the overall runtime of the resulting algorithm remains $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$.

Hierarchical cuttings. This ingredient is needed for counting intersection points in cells that are fully contained inside a query triangle. The application of hierarchical cuttings to our problem significantly reduces the query time – see below. Fix a parameter $g \ll n$ and put $r := n/g$. We construct a *hierarchical* $(1/r)$ -cutting $\Xi(A)$ for the segments of A , which is a hierarchy of $(1/r_0)$ -cuttings, where r_0 is some sufficiently large constant. The top-level cutting $\Xi_1(A)$ is constructed for A . Since the segments of A are pairwise disjoint, we can construct $\Xi_1(A)$ so that it consists of only $O(r_0)$ trapezoids (for concreteness, we write this bound as br_0 , for some absolute constant b), each of which is crossed by at most

n/r_0 segments of A , which comprise the so-called *conflict list* of the cell σ , denoted as A_σ . The construction time of $\Xi_1(A)$, in the real-RAM model, is $O(n \log r_0) = O(n)$. See [14, Theorem 1] for details.

For each cell σ of $\Xi_1(A)$, we clip the segments in its conflict list A_σ to within σ and apply the cutting-construction step recursively to this set, clipping also the cells of the new cutting to within σ (and ignoring cells, or portions thereof, that lie outside σ , as they are not met by any of the clipped segments of A_σ). We denote the union of all the resulting $(1/r_0)$ -cuttings as $\Xi_2(A)$. We continue recursively in this manner, until we reach a level s at which all the cells are crossed by at most n/r segments. We thus obtain a hierarchy of cuttings $\Xi_1(A), \Xi_2(A), \dots, \Xi_s(A)$, for some index $s = O(\log r)$. We denote the collective hierarchy as $\Xi(A)$. Since we stop the recursion as soon as $n/r_0^s \leq n/r$, the overall number of cells of all the levels is $O((br_0)^s) = O(r^{1+\varepsilon})$, for any prespecified $\varepsilon > 0$, for a suitable choice of $r_0 = r_0(\varepsilon)$. Technically, the trapezoids in the cutting are relatively open, and the cutting also includes one- and zero-dimensional cells; as the latter are easier to deal with, we will focus below on the two-dimensional cells of the cutting. At any level j of the hierarchy, the cells of $\Xi_j(A)$ are pairwise disjoint. As these cells partition the plane, each intersection point between a segment of A and a segment of B lies in precisely one cell of a suitable dimension at each level. See Figure 1 for an illustration.



■ **Figure 1** Interaction of a hierarchical cutting with a triangle. The dark gray cells are the ones inside the triangle at the top level of the hierarchy; the medium gray cells are the ones inside the triangle at the second level (and whose parent cells are not inside the triangle). The light gray cells will be refined and handled at lower levels, since they intersect the triangle boundary.

We apply a similar hierarchical construction for B , and let $\Xi(B) = \{\Xi_j(B)\}_{j \leq s}$ denote the resulting hierarchical cutting, which has analogous properties. (We assume for simplicity that the highest index s is the same in both hierarchies.)

We now overlay $\Xi(A)$ with $\Xi(B)$, that is, at each level j of the hierarchy, we overlay the cells of $\Xi_j(A)$ with the cells of $\Xi_j(B)$. We denote the j th level overlay as Ξ_j , and the entire hierarchical overlay structure as $\Xi = \{\Xi_j\}_{j \leq s}$. Since each of $\Xi_j(A)$ and $\Xi_j(B)$ consists of at most $(br_0)^j$ cells, the number of cells of Ξ_j is at most $O((br_0)^{2j})$. Since we have $r_0^s \approx r$ (up to a factor of r_0), it follows that the overall complexity of all the overlays is $O(r^{2+2\varepsilon})$, provided that we choose r_0 , as above, to be sufficiently large, as a function of ε .

For simplicity of exposition, we ignore lower-dimensional faces of the cuttings, and regard each of the overlays Ξ_j as a decomposition of the plane into pairwise openly disjoint convex polygons, each of complexity linear in $j \leq s = O(\log r)$. Each cell σ of the overlay is

identified by the pair (τ, τ') , where τ and τ' are the respective cells of $\Xi_j(A)$ and $\Xi_j(B)$ whose intersection is σ ; we simply write $\sigma = (\tau, \tau')$. Each bottom-level cell σ of the final overlay Ξ_s is crossed by at most $n/r = g$ segments of A and by at most g segments of B .

Classifying the segments and triangles. Let $\sigma = (\tau, \tau')$ be a cell of Ξ_j , for any level j of the hierarchy. Call a segment e of A *long* (resp., *short*) *within* σ if e crosses σ and neither of its endpoints lies in σ (resp., at least one endpoint lies in σ). Let A_σ^l (resp., A_σ^s) denote the set of long (resp., short) segments of A within σ . Apply analogous definitions and notations to the segments of B . Denote by C_σ (resp., $C_\sigma^{(0)}$) the set of triangles with at least one edge that crosses σ (resp., that fully contain σ). Call a triangle $\Delta \in C_\sigma$ *long* (resp., *short*) in σ if σ does not (resp., does) contain a vertex of Δ , and denote by C_σ^l (resp., C_σ^s) the set of long (resp., short) triangles in C_σ .

For each triangle $\Delta \in C$, each of its edges crosses only $O((br_0)^j)$ cells of Ξ_j . Indeed, as such an edge crosses from one cell of Ξ_j to an adjacent cell, it does so by crossing the boundary of either a cell of $\Xi_j(A)$ or a cell of $\Xi_j(B)$, and the total number of such crossings is $O((br_0)^j)$. In particular, the edge crosses at most $O(r^{1+\varepsilon})$ cells of the final overlay Ξ_s . It follows that $\sum_{\sigma \in \Xi} |C_\sigma^l| \leq \sum_{\sigma \in \Xi} |C_\sigma| = O(nr^{1+\varepsilon})$, but clearly $\sum_{\sigma \in \Xi} |C_\sigma^s|$ is only $O(n \log r)$. In contrast, Δ can fully contain many more cells of Ξ_s , perhaps almost all of them, but the hierarchical nature of the construction allows us to deal with a much smaller number of such interior cells, by collecting them at higher levels of the hierarchy; see below for details.

The algorithm: A quick review. The high-level structure of the algorithm is as follows (see also the “roadmap” overview given earlier). We construct the hierarchies $\Xi(A) = \{\Xi_j(A)\}_{j \geq 1}$ and $\Xi(B) = \{\Xi_j(B)\}_{j \geq 1}$. For each cell τ of $\Xi_j(A)$ (resp., τ' of $\Xi_j(B)$), we compute its conflict list A_τ (resp., $B_{\tau'}$), which, as we recall, is the set of all segments of A that cross τ (resp., segments of B that cross τ'). We then form the hierarchical overlay $\Xi = \{\Xi_j\}_{j \geq 1}$, and for each cell $\sigma = (\tau, \tau')$ of any overlay Ξ_j , we compute the subset A_σ of the segments of A_τ that cross σ , and the subset B_σ of the segments of $B_{\tau'}$ that cross σ . We partition A_σ into the subsets A_σ^l and A_σ^s of long and short segments (within σ), respectively, and apply an analogous partition to B_σ . The additional overall cost for constructing these sets, over all hierarchical levels, is $O(r^{2+\varepsilon} \cdot n/r) = O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$. (The cost at the bottom level dominates the entire cost over all levels.)

We also trace each triangle $c \in C$ through the cells of Ξ that are crossed by its edges, and form, for each cell σ of the overlay, the list C_σ of triangles of C with at least one edge that crosses σ . We partition C_σ into the subsets C_σ^l and C_σ^s , as defined earlier. As we show below, we can handle, in a much more efficient way, the short triangles of C_σ^s , as well as the triangles of C_σ^l all three of whose edges cross σ , simply because the overall number of such triangle-cell interactions is small. We therefore focus on the triangles of C_σ^l that have only one or two edges crossing σ . For triangles with two crossing edges we use a standard two-level data structure (where in each level we consider only one crossing edge). This lets us assume, without loss of generality, that each triangle in C_σ^l is a halfplane. Each of these halfplanes can be represented by its bounding line, that is the line supporting the appropriate crossing edge of the triangle. We flesh out the details below.

We also assume, for now, that all the segments of A_σ and of B_σ are long in σ (and so we drop the superscript l). This is the hard part of the analysis, requiring the involved machinery presented below. After handling this case, we will address the much simpler situations that involve short segments and/or short triangles (or triangles with three edges crossing σ). The cost of handling short segments or short triangles within cells is lower, even in the uniform model, since the overall number of short objects within cells is smaller.

Handling the long segments. We preprocess each level j of the overlay, to compute, for each of its cells $\sigma = (\tau, \tau')$, the number of intersection points between the (long) segments of A_σ and those of B_σ (which, due to the clipping, lie in σ). This is a standard procedure that involves computing the number of pairs of segments from $A_\sigma \times B_\sigma$ whose intersection points with the boundary of σ interleave (these are precisely the pairs of intersecting segments), and can be implemented to take $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$ time; see, e.g., [1]. We store the resulting count at σ .

Consider a two-dimensional cell σ , a segment $a \in A_\sigma$, a segment $b \in B_\sigma$, and a triangle $\Delta \in C_\sigma$. By assumption, Δ has only one edge c or two edges c_1, c_2 crossing σ . When a and b intersect inside σ , the intersection lies in Δ if and only if the triple (a, b, c) , or each of the triples (a, b, c_1) , (a, b, c_2) , has a prescribed orientation, reflecting the condition that the point $a \cap b$ lies on the side of c (or the sides of c_1, c_2) that contain Δ . This orientation (or orientations) can be positive, negative, or zero, depending on the relative order of the slopes of a, b , and c (or of c_1 and c_2), and on whether Δ lies to the left or to the right of c (or of c_1, c_2).

For each halfplane c^+ that represents a triangle $\Delta \in C_\sigma$ (the halfplane contains Δ and is bounded by the line supporting the single (relevant) edge c of Δ that crosses σ), we want either (i) to represent the set of pairs $(a, b) \in A_\sigma \times B_\sigma$ that have a prescribed orientation of the triple (a, b, c) , as the disjoint union of complete bipartite graphs, or (ii) to count the number of such pairs. The subtask (i) arises in cases where Δ has two edges crossing σ and is needed for the first level of the data structure, which we query with the first crossing edge of Δ . The subtask (ii) arises in the second level of the structure, which we query with the second crossing edge of Δ , and in cases where only one edge of Δ crosses σ .

We also count the number of intersections within σ , in $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$ time. As a matter of fact, with a simple modification of the procedure, we can, within the same time bound, represent the set of all pairs of segments $(a, b) \in A_\sigma \times B_\sigma$ that intersect each other (inside σ) as the disjoint union of complete bipartite graphs, so that the overall size of their vertex sets is $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$. This follows from standard planar segment-intersection range searching machinery; see, e.g., [1]. In what follows we focus on just one such graph, and to simplify the presentation we denote it as $A_\sigma \times B_\sigma$, with a slight abuse of notation.

Preparing for Fredman's trick. We use the infrastructure developed by Aronov et al. [5], with suitable modifications, but adapt it to the order-type context. We preprocess A and B into a data structure that we will then search with the points dual to the lines supporting the edges of the triangles of C . For each $a \in A$, $b \in B$, we define $\gamma_{a,b}$ to be the line that passes through a^* and b^* , where a^* (resp., b^*) is the point dual to a (resp., b). By our general position assumption, $a^* \neq b^*$, so $\gamma_{a,b}$ is well defined. Let Γ_0 denote the set of these n^2 lines. Our goal in task (ii) is to count, for each cell σ of any of the overlays, for each point c^* dual to an edge of a triangle $\Delta \in C_\sigma$, the number of lines of Γ_0 that lie above c^* , the number of lines that are incident to c^* , and the number of lines that lie below c^* . In task (i), we want to represent each of these sets of lines as the disjoint union of a small number of precomputed canonical sets. This calls for preprocessing the arrangement $\mathcal{A}(\Gamma_0)$ into a suitable point location data structure, which we will then search with each $c^* \in C^*$, and retrieve the desired data from the outcome of each query.

As in, e.g., [5], a naïve implementation of this approach will be too expensive. Instead, we return to the hierarchical partitions $\Xi(A)$, $\Xi(B)$, and Ξ , and iterate, over all cells $\sigma = (\tau, \tau')$ of the bottom level Ξ_s , defining $\Gamma_\sigma := \{\gamma_{a,b} \mid (a, b) \in A_\sigma \times B_\sigma\}$. In principle, we want to

construct the separate arrangements $\mathcal{A}(\Gamma_\sigma)$, over the cells σ , preprocess each of them into a point location data structure, and search, for each triangle $\Delta \in C$, in the structures that correspond to the cells of Ξ that are either crossed by (at most) one or two edges of Δ , or fully contained in Δ . This is also too expensive if implemented naïvely, so we use instead Fredman’s trick, combined with the machinery developed in Section 2.

We first observe that, for each triangle $\Delta \in C$, finding the cells σ (at any level of the hierarchy) that Δ fully contains is easy and inexpensive. We go over the hierarchy of the overlays Ξ_j . At the root we find, by brute force, all the (constantly many) cells of Ξ_1 that Δ fully contains, and add their intersection counts to our output counter. We then recurse, in the same manner, in the at most br_0 cells of Ξ_1 that Δ crosses. Thus the number of cells we visit is at most $O(r_0^2) \cdot (1 + br_0 + (br_0)^2 + \dots + (br_0)^s) = O(r^{1+\varepsilon})$, so the overall cost of this step⁸ is $O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$.

We therefore focus, for each triangle Δ of C , only on the cells that it crosses (at every level of the hierarchy), and restrict the analysis for now to cells at which Δ is long, with at most two of its edges crossing the cell. Repeating most of the analysis just given, the number of these cells is $O(r^{1+\varepsilon})$ (with a smaller constant of proportionality, since we now do not have the factor $O(r_0^2)$, as above).

Constructing $\mathcal{A}(\Gamma_\sigma)$ in the decision-tree model. Consider the step of constructing $\mathcal{A}(\Gamma_\sigma)$ for some fixed bottom-level cell σ . Following the technique in Section 2, we perform this step using only the order type of Γ_σ , and we begin by considering the task of obtaining the order-type information. That is, we want to determine, for each ordered triple $(\gamma_{a_1, b_1}, \gamma_{a_2, b_2}, \gamma_{a_3, b_3})$ of lines of Γ_σ , whether the point $\gamma_{a_1, b_1} \cap \gamma_{a_2, b_2}$ lies to the left or to the right of the point $\gamma_{a_1, b_1} \cap \gamma_{a_3, b_3}$. Let $G(a_1, a_2, a_3; b_1, b_2, b_3)$ denote the 12-variate polynomial (of constant degree) whose sign determines the outcome of the above comparison. (The immediate expression for G is a rational function, which we turn into a polynomial by multiplying it by the square of its denominator, without affecting its sign; our general position assumption ensures that none of the denominators vanishes.)

Once the signs of all expressions $G(a_1, a_2, a_3; b_1, b_2, b_3)$ are determined, we can apply Lemma 1. The rest of the preprocessing, which constructs a discrete representation of the arrangement, say, in the DCEL format [13], and turns this representation into an efficient point location data structure, can be carried out at no cost in the algebraic decision-tree model.

We search the structure with each triangle $\Delta \in C_\sigma$. We may assume that Δ is long in σ and that only one or two edges of Δ cross σ , as the other cases are easy to handle. Assuming further that there is only one such edge c , locating the dual point c^* in $\mathcal{A}(\Gamma_\sigma)$ takes $O(\log g)$ time, as shown in Section 2 (noting that Γ_σ consists of only g^2 lines). With suitable preprocessing, locating c^* gives us, for free in our model, the three sets of the lines that pass above c^* , are incident to c^* , or pass below c^* . The case where two edges of Δ cross σ is handled using a two-level version of the structure; see below for details. The point location cost now goes up to $O(\log^2 g)$.

Consider then the step of computing the order type of the lines of Γ_σ , that is, of computing the sign of $G(a_1, a_2, a_3; b_1, b_2, b_3)$, for every triple of segments $a_1, a_2, a_3 \in A_\sigma$ and every triple of segments $b_1, b_2, b_3 \in B_\sigma$. To this end, we play Fredman’s trick. We fix a bottom-level cell τ of $\Xi(A)$. For each triple $(a_1, a_2, a_3) \in A_\tau^3$, we define the surface

$$\psi_{a_1, a_2, a_3} = \{(b_1, b_2, b_3) \in \mathbb{R}^6 \mid G(a_1, a_2, a_3; b_1, b_2, b_3) = 0\},$$

⁸ It is for making this step efficient that we use hierarchical partitions. A single-shot partition would have forced the query to visit up to $\Theta(r^2)$ such cells, which would make it too expensive.

and denote by Ψ the collection of these surfaces, over all cells τ . We have $N := |\Psi| = O((n/g)^{1+\varepsilon} \cdot g^3) = O(n^{1+\varepsilon}g^2)$. Similarly, we let P denote the set of all triples (b_1, b_2, b_3) , for $b_1, b_2, b_3 \in B_{\tau'}$, over all cells τ' of $\Xi(B)$. We have $M := |P| = O(n^{1+\varepsilon}g^2)$. These bounds pertain to the bottommost level of the hierarchy; they are smaller at levels of smaller indices.

We apply a batched point-location procedure to the points of P and the surfaces of Ψ . The output of this procedure is a collection of complete bipartite subgraphs of $P \times \Psi$, so that, for each such subgraph $P_\alpha \times \Psi_\alpha$, $G(a_1, a_2, a_3; b_1, b_2, b_3)$ has a fixed sign for all $(b_1, b_2, b_3) \in P_\alpha$ and all $(a_1, a_2, a_3) \in \Psi_\alpha$, see, e.g., [3, 12] for the use of such structures in similar contexts. This tells us the desired signs of $G(a_1, a_2, a_3; b_1, b_2, b_3)$, for every pair of triples $(a_1, a_2, a_3) \in A_\tau^3$, $(b_1, b_2, b_3) \in B_{\tau'}^3$, over all pairs of cells $(\tau, \tau') \in \Xi(A) \times \Xi(B)$, and these signs give us the orientation (i.e., the order of the intersection points) of every triple of lines $\gamma_{a,b}$. That is, we obtain the order type of the lines. As remarked in Section 2, we may assume that this also includes the sorting of the lines at $x = -\infty$, but, for the sake of concreteness, we address this simpler task in the full version [4, Section 3].

The batched point location step proceeds by using the recent multilevel polynomial partitioning technique of Agarwal et al. [2, Corollary 4.8]. We delegate the full, and rather technical, details of the analysis to the full version [4, Section 3.1], and just summarize the result here.

► **Proposition 2.** *Let $T(M, N)$ denote the maximum possible sum of the sizes of the vertex sets of the complete bipartite graphs produced by the recursive process described above, over all input sets of at most M points and at most N surfaces. Then we have*

$$T(M, N) = O\left(M^{6/7+\varepsilon}N^{6/7+\varepsilon} + M^{1+\varepsilon} + N^{1+\varepsilon}\right),$$

for any $\varepsilon > 0$, where the constant of proportionality depends on ε . The same asymptotic bound also holds for the cost (in the uniform model) of constructing these graphs.

In summary, the information collected so far allows us to obtain the combinatorial structure of each of the arrangements $\mathcal{A}(\Gamma_\sigma)$, over all cells σ of Ξ , and subsequently construct an order-type-based point-location data structure for each of them, at no extra cost in the algebraic decision-tree model. The overall cost of this phase, in this model, is thus $O((n^{1+\varepsilon}g^2)^{12/7+\varepsilon})$, for any $\varepsilon > 0$. By replacing ε by some small multiple thereof, we can write this bound as $O((ng^2)^{12/7+\varepsilon})$, for any $\varepsilon > 0$.

Fredman’s trick, as applied above, separates the handling of the conflict lists A_τ , over the trapezoids τ of $\Xi(A)$, and the conflict lists $B_{\tau'}$, over the trapezoids τ' of $\Xi(B)$. For a cell $\sigma = (\tau, \tau')$ of Ξ , not all the segments in A_τ necessarily cross σ , so we have to retain (for σ) only those that do cross it, and apply a similar pruning to $B_{\tau'}$. As we show in the full version [4, Section 3], the cost of this filtering step is $O(g)$ for each σ , for an overall cost of $O((n/g)^2 \cdot g) = O(n^2/g)$.

Searching with the elements of C . We now need to search the structures computed in the preceding phase with the dual features of the triangles of C . Due to lack of space, we delegate the description to the full version [4, Section 3], where we show that the total searching time, for all the elements of C , is $O\left(\frac{n^{2+\varepsilon} \log^2 g}{g}\right)$, concluding (see once again our high-level roadmap):

► **Theorem 3.** *Let A and B be two sets each consisting of n pairwise disjoint segments in the plane, and let C be a set of n triangles in the plane. We can count, for each triangle $\Delta \in C$, the number of intersection points of segments of A with segments of B that lie inside Δ , in the algebraic decision-tree model, at the subquadratic cost $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$.*

► **Corollary 4.** *We can solve, in the algebraic decision-tree model, at the cost of $O(n^{60/31+\varepsilon})$, for any $\varepsilon > 0$, each of the problems (i) intersection of three polygons, (ii) coverage by three polygons, and (iii) segment concurrency, as listed in the introduction.*

References

- 1 Pankaj K. Agarwal. Parititioning arrangements of lines II: Applications. *Discret. Comput. Geom.*, 5:533–573, 1990. doi:10.1007/BF02187809.
- 2 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, and Joshua Zahl. Efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50(2):760–787, 2021. doi:10.1137/19M1268550.
- 3 Pankaj K. Agarwal, Marco Pellegrini, and Micha Sharir. Counting circular arc intersections. *SIAM J. Comput.*, 22(4):778–793, 1993. doi:10.1137/0222050.
- 4 Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir. Subquadratic algorithms for some 3SUM-hard geometric problems in the algebraic decision tree model. *CoRR*, abs/2109.07587, 2021. arXiv:2109.07587.
- 5 Boris Aronov, Esther Ezra, and Micha Sharir. Testing polynomials for vanishing on cartesian products of planar point sets. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 8:1–8:14, 2020. doi:10.4230/LIPIcs.SocG.2020.8.
- 6 Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic algorithms for algebraic 3SUM. *Discret. Comput. Geom.*, 61(4):698–734, 2019. doi:10.1007/s00454-018-0040-y.
- 7 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 80–86. ACM, 1983. doi:10.1145/800061.808735.
- 8 Jürgen Bokowski, Simon King, Susanne Mock, and Ileana Streinu. The topological representation of oriented matroids. *Discret. Comput. Geom.*, 33(4):645–668, 2005. doi:10.1007/s00454-005-1164-4.
- 9 Jürgen Bokowski, Susanne Mock, and Ileana Streinu. On the Folkman-Lawrence topological representation theorem for oriented matroids of rank 3. *Eur. J. Comb.*, 22(5):601–615, 2001. doi:10.1006/eujc.2000.0482.
- 10 Jean Cardinal, John Iacono, and Aurélien Ooms. Solving k -SUM using few linear queries. In *24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 25:1–25:17, 2016. doi:10.4230/LIPIcs.ESA.2016.25.
- 11 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.
- 12 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994. doi:10.1007/BF01182771.
- 13 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 14 Mark de Berg and Otfried Schwarzkopf. Cuttings and applications. *Int. J. Comput. Geom. Appl.*, 5(4):343–355, 1995. doi:10.1142/S0218195995000210.
- 15 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986. doi:10.1137/0215023.
- 16 Esther Ezra, Sarel Har-Peled, Haim Kaplan, and Micha Sharir. Decomposing arrangements of hyperplanes: VC-dimension, combinatorial dimension, and point location. *Discret. Comput. Geom.*, 64(1):109–173, 2020. doi:10.1007/s00454-019-00141-7.

- 17 Esther Ezra and Micha Sharir. A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model. *Discret. Comput. Geom.*, 61(4):735–755, 2019. doi:10.1007/s00454-018-0043-8.
- 18 Jon Folkman and Jim Lawrence. Oriented matroids. *J. Comb. Theory, Ser. B*, 25(2):199–236, 1978. doi:10.1016/0095-8956(78)90039-4.
- 19 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 20 Omer Gold and Micha Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *25th Annual European Symposium on Algorithms, ESA 2017*, volume 87 of *LIPICs*, pages 42:1–42:13, 2017. doi:10.4230/LIPICs.ESA.2017.42.
- 21 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM J. Comput.*, 12(3):484–507, 1983. doi:10.1137/0212032.
- 22 Jacob E. Goodman and Richard Pollack. Semispaces of configurations, cell complexes of arrangements. *J. Comb. Theory, Ser. A*, 37(3):257–293, 1984. doi:10.1016/0097-3165(84)90050-5.
- 23 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. doi:10.1145/3185378.
- 24 David Haussler and Emo Welzl. ε -nets and simplex range queries. *Discret. Comput. Geom.*, 2:127–151, 1987. doi:10.1007/BF02187876.
- 25 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-SUM and related problems. *J. ACM*, 66(3):16:1–16:18, 2019. doi:10.1145/3285953.
- 26 D. T. Lee and Franco P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6(3):594–606, 1977. doi:10.1137/0206043.
- 27 Jirí Matousek and Zuzana Patáková. Multilevel polynomial partitions and simplified range searching. *Discret. Comput. Geom.*, 54(1):22–41, 2015. doi:10.1007/s00454-015-9701-2.
- 28 Stefan Meiser. Point location in arrangements of hyperplanes. *Inf. Comput.*, 106(2):286–303, 1993. doi:10.1006/inco.1993.1057.
- 29 Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.