Intersection Queries for Flat Semi-Algebraic Objects in Three Dimensions and Related Problems*

Pankaj K. Agarwal
Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129 USA

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201 USA

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

Matthew J. Katz
Department of Computer Science, Ben Gurion University, Beer Sheva, Israel

Micha Sharir
School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Abstract

Let $\mathcal{T}$ be a set of n planar semi-algebraic regions in $\mathbb{R}^3$ of constant complexity (e.g., triangles, disks), which we call *plates*. We wish to preprocess $\mathcal{T}$ into a data structure so that for a query object γ, which is also a plate, we can quickly answer various *intersection queries*, such as detecting whether γ intersects any plate of $\mathcal{T}$, reporting all the plates intersected by γ, or counting them. We also consider two simpler cases of this general setting: (i) the input objects are plates and the query objects are constant-degree algebraic arcs in $\mathbb{R}^3$ (*arcs*, for short), or (ii) the input objects are arcs and the query objects are plates in $\mathbb{R}^3$. Besides being interesting in their own right, the data structures for these two special cases form the building blocks for handling the general case.

By combining the polynomial-partitioning technique with additional tools from real algebraic geometry, we obtain a variety of results with different storage and query-time bounds, depending on the complexity of the input and query objects. For example, if $\mathcal{T}$ is a set of plates and the query objects are arcs, we obtain a data structure that uses $O^*(n^{4/3})$ storage (where the $O^*(\cdot)$ notation hides subpolynomial factors) and answers an intersection query in $O^*(n^{2/3})$ time. Alternatively, by increasing the storage to $O^*(n^{3/2})$, the query time can be decreased to $O^*(n^\rho)$, where $\rho = (2t-3)/3(t-1) < 2/3$ and $t \geq 3$ is the number of parameters needed to represent the query arcs.

Our approach can be extended to many additional intersection-searching problems in three and higher dimensions, even when the input or query objects are not planar. We demonstrate the versatility of our technique by providing efficient data structures for answering segment-intersection queries amid a set of spherical caps in $\mathbb{R}^3$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Intersection searching, Semi-algebraic range searching, Point-enclosure queries, Ray-shooting queries, Polynomial partitions, Cylindrical algebraic decomposition, Multi-level partition trees

Funding *Pankaj K. Agarwal*: Work partially supported by NSF grants IIS-18-14493 and CCF-20-07556.

Boris Aronov: Work partially supported by NSF Grants CCF-15-40656 and CCF-20-08551, and by

* An abridged preliminary version of this work appears in the Proceedings of the 38th Symposium on Computational Geometry (SoCG), 2022.

Grant 2014/170 from the US-Israel Binational Science Foundation.
Esther Ezra: Work partially supported by NSF CAREER under Grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation.
Matthew J. Katz: Work partially supported by Grant 1884/16 from the Israel Science Foundation, and by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Micha Sharir: Work partially supported by Grant 260/18 from the Israel Science Foundation.

Acknowledgments.

We thank Peyman Afshani for sharing with us problems that have motivated our study of segment-intersection searching amid spherical caps. We also thank Ovidiu Daescu for suggesting the problems studied in the latter part of the paper and for providing their application to collision detection in robotics. Finally, thanks are due to the reviewers of the conference version of this work for their helpful comments.

1 Introduction

This paper studies intersection-searching problems in $\mathbb{R}^3$, where both input and query objects are planar semi-algebraic regions of constant complexity (e.g., triangles, disks), which we refer to as *plates*.[1] We also consider two simpler cases of this setup: (i) the input objects are plates and the query objects are constant-degree algebraic arcs in $\mathbb{R}^3$, referred to simply as *arcs*, and (ii) the input objects are arcs and the query objects are plates in $\mathbb{R}^3$. Besides being interesting in their own right, the data structures for these two simpler cases form the building blocks for handling the general case. In each case, we wish to preprocess a set $\mathcal{T}$ of input objects (plates or arcs) in $\mathbb{R}^3$ into a data structure that supports various *intersection queries* for a query object (again a plate or an arc) γ, where we want to determine whether γ intersects any object of $\mathcal{T}$ (*intersection-detection* queries), report all objects of $\mathcal{T}$ that γ intersects (*intersection-reporting* queries), count the number of objects of $\mathcal{T}$ that γ intersects or the number of intersection points of γ with the input objects (*intersection-counting* queries), or, when the query object is a directed arc γ, report the first input object intersected by γ (*ray-shooting* queries). Intersection queries arise in many applications, including robotics, computer-aided design, and solid modeling.

Notwithstanding a considerable amount of work on segment-intersection or ray-shooting queries amid triangles in $\mathbb{R}^3$ (see, e.g., the survey by Pellegrini [28]), little is known about more general intersection queries in $\mathbb{R}^3$, e.g., how quickly one can answer arc-intersection queries amid triangles in $\mathbb{R}^3$, or triangle-intersection queries amid arcs in $\mathbb{R}^3$. The present work makes significant and fairly comprehensive progress on the design of efficient solutions to general intersection-searching problems in $\mathbb{R}^3$.

1.1 Related work

The general intersection-searching problem asks to preprocess a set $\mathcal{O}$ of geometric objects in $\mathbb{R}^d$, so that one can quickly report or count all objects of $\mathcal{O}$ intersected by a query object γ, or just test whether γ intersects any object of $\mathcal{O}$ at all. One may also want to perform some

[1] Roughly speaking, a semi-algebraic set in $\mathbb{R}^d$ is the set of points in $\mathbb{R}^d$ satisfying a Boolean predicate over a set of polynomial inequalities; the complexity of the predicate and of the set is defined in terms of the number of polynomials involved and their maximum degree. See [13] for details.

other aggregate operations on these objects (see [2] for a general framework). Intersection searching is a generalization of range searching (in which the input objects are points) and point enclosure queries (in which the query objects are points).

A popular approach to answering intersection queries is to write a first-order formula for the intersection condition between an input object and a query object. Using quantifier elimination, intersection queries can be reduced to *semi-algebraic range* queries, by working in *object space*, where each input object $O \in \mathcal{O}$ is mapped to a point $\hat{O}$ and a query object γ is mapped to a semi-algebraic region $\hat{\gamma}$, such that $\hat{\gamma}$ contains a point $\hat{O}$ if and only if γ intersects the corresponding input object O. Alternatively, the problem can be reduced to a *point-enclosure* query, by working in *query space*, where now each input object O is mapped to a semi-algebraic region $\tilde{O}$ and each query object γ is mapped to a point $\tilde{\gamma}$, so that $\tilde{\gamma}$ lies in $\tilde{O}$ if and only if γ intersects O. The first approach leads to a linear-size data structure with sublinear query time, and the second approach leads to a large-size data structure with logarithmic or polylogarithmic query time; see, e.g., [6, 8, 17, 24, 31] for the first approach and [3, 14] for the second one.

The performance of these data structures depends on the number of parameters needed to specify the input and query objects. We refer to these numbers as the *parametric dimension* (or the number of *degrees of freedom* (dof)) of the input and query objects, respectively. Sometimes the performance can be improved using a *multi-level data structure*, where each level uses a lower-dimensional sub-predicate [2]. One can also combine these two approaches to obtain a query-time/storage trade-off. For example, using standard techniques (such as in [27]), a ray-shooting or segment-intersection query amid n triangles in $\mathbb{R}^3$ can be answered in $O^*(n^{3/4})$ time using $O^*(n)$ storage, in $O(\log n)$ time using $O^*(n^4)$ storage, or in $O^*(n/s^{1/4})$ time using $O^*(s)$ storage,[2] for $n \le s \le n^4$, by combining the first two solutions [27, 28]. As in the abstract, the $O^*(\cdot)$ notation hides subpolynomial factors, e.g., of the form $O(n^\varepsilon)$, for arbitrarily small $\varepsilon > 0$, and their coefficients which depend on ε. A similar multi-level approach yields data structures in which a ray-shooting query amid n planes or spheres in $\mathbb{R}^3$ can be answered in $O^*(n/s^{1/3})$ time using $O^*(s)$ storage, for $n \le s \le n^3$ [26, 27, 28, 30].

A departure from this approach is the *pedestrian* approach for answering ray-shooting queries. For instance, given a simple polygon P with n edges, a Steiner triangulation of P is constructed so that a line segment lying inside P intersects only $O(\log n)$ triangles. A query is answered by traversing the query ray through this sequence of triangles [23]. The pedestrian approach has also been applied to polygons with holes in $\mathbb{R}^2$ [4, 23], to a convex polyhedron in $\mathbb{R}^3$ [18], and to polyhedral subdivisions in $\mathbb{R}^3$ [4, 11]. Some of the ray-shooting data structures combine the pedestrian approach with the above range-searching tools [1, 9, 17].

Recently, Ezra and Sharir [19] proposed a new approach for answering ray-shooting queries amid triangles in $\mathbb{R}^3$, using the pedestrian approach in the context of the polynomial-partitioning scheme of Guth [20]. Roughly speaking, they construct a partitioning polynomial F of degree $O(D)$, for a sufficiently large constant D, using the algorithm in [3]. The zero set $Z(F)$ of F partitions $\mathbb{R}^3$ into *cells*, which are the connected components of $\mathbb{R}^3 \setminus Z(F)$. The partitioning scheme guarantees that, with a suitable choice of the degree, each cell τ is intersected by at most n/D input triangles, but for only at most n/D^2 of them their (relative) boundary intersects τ.[3] These latter triangles are called *narrow* at τ, and the other intersect-

[2] We sometimes refer to s as the "storage parameter," to distinguish it from the actual storage being used, which is $O^*(s)$.

[3] One actually has to construct two polynomials, one for ensuring the first property and one for the second property, and take their product, still a polynomial of degree $O(D)$, as the desired partitioning polynomial.

ing triangles are called *wide*. For each cell τ, the algorithm of [19] recursively preprocesses the narrow triangles of τ and constructs a secondary data structure for the wide triangles at τ. A major technical result of [19] is to reduce a ray-shooting or intersection-detection query amid wide triangles to a similar query amid a set of planes in $\mathbb{R}^3$ (those supporting the input triangles), and to use the fact that such a query amid planes can be answered in $O^*(n/s^{1/3})$ time when $O^*(s)$ storage is available, for any $n \leq s \leq n^3$; see [27, 28]. This leads to a data structure with $O^*(n^{3/2})$ storage and $O^*(n^{1/2})$ query time, which improves upon the earlier solution [27]. Concretely, the previous techniques, which were based on geometric cuttings and multi-level partition trees [2, 28], reduced the line-triangle-intersection query problem to a four-dimensional range searching problem, while the technique in [19] managed to reduce it to three-dimensional range searching, which led to the improved bounds. The approach of [19] can also support reporting queries in $O^*(n^{1/2} + k)$ time, where k is the output size, but, for certain technical reasons, it does not support counting queries.

1.2 Our results

We refer to a connected path π as an *(algebraic) arc* if it is the restriction of a real algebraic curve $\gamma\colon I \to \mathbb{R}^3$ to a subinterval $[a, b] \subseteq I$. The *parametric dimension* t of π, also referred to as *the number of degrees of freedom* (dof) of π, is the number of real parameters needed to describe π. Two of these parameters specify the endpoints a and b. We assume that the degree of the curve is also bounded by t.

We present efficient data structures for three broad classes of intersection searching in $\mathbb{R}^3$: (i) the input objects are plates and the query objects are arcs in $\mathbb{R}^3$, (ii) the input objects are arcs and the query objects are plates in $\mathbb{R}^3$, and (iii) both input and query objects are plates in $\mathbb{R}^3$. Our algorithms combine the polynomial-partitioning technique of Guth [20] and of Guth and Katz [21] with some additional tools from real algebraic geometry.

For simplicity, we mostly focus on answering *intersection-detection* queries. Our data structures extend to answering *intersection-reporting* queries by spending additional $O(k)$ time, where k is the output size. For type (i) intersection queries, using the parametric-search framework of Agarwal and Matoušek [7], our data structures can also answer *arc-shooting* queries, where the goal is to find the first plate of $\mathcal{T}$ hit by a (directed) query arc, if such a plate exists. Most of the data structures can be extended to answering *intersection-counting* queries as well — for type (i) and (ii) intersection queries, our data structures count the number of intersection points between the query arc/plate and the input plates/arcs, and for type (iii) queries, our approach can count the number of intersecting pairs if both input and query objects are triangles. Table 1 summarizes the main results of the paper. When we say that an intersection query can be answered in $O^*(t(n))$ time, we mean that detection, counting, and shooting queries can be answered in $O^*(t(n))$ time and reporting queries in $O^*(t(n)) + O(k)$ time, where k is the output size.

Intersection-searching with arcs amid plates. We present several data structures for answering arc-intersection queries amid a set $\mathcal{T}$ of n plates in $\mathbb{R}^3$ (cf. Sections 2, 4, and 5). Our first main result is an $O^*(n^{4/3})$-size data structure that can be constructed in $O^*(n^{4/3})$ expected time and that supports arc-intersection queries in $O^*(n^{2/3})$ time (see Section 2). The asymptotic query time bound depends neither on the parametric dimension of the query arc nor on that of the input plates, though the coefficients hiding in the O^*-notation do depend on them. Although our high-level approach is similar to that of Ezra and Sharir [19], handling wide plates in our setup is significantly more challenging because the query object is an arc instead of a line segment. We handle wide input plates using a completely different

Input	Query	Storage	Query Time	Reference
Plates	Arc/Curve	4/3	2/3	Theorem 2.2
Plates	Arc/Curve ($t \geq 3$ dof)	3/2	$(2t-3)/3(t-1)$	Theorem 2.4
Plates	Planar arc ($t \geq 4$ dof)	3/2	$(2t-7)/3(t-3)$	
Plates	Circular arc	3/2	3/5	Theorem 2.5
Triangles	Arc/Curve	1	4/5	Theorem 5.1
Triangles	Arc/Curve	11/9	2/3	Theorem 2.3
Spherical caps	Segment	5/4	3/4	Theorem 8.1
Spherical caps	Segment	3/2	27/40	Theorem 8.1
Segments	Plate	3/2	1/2	Theorem 6.3
Arcs/Curves (t dof)	Plate	3/2	$3(t-1)/4t$	Theorem 6.4
Triangles*	Triangle	3/2	1/2	Theorem 7.1
Plates (t_O dof)**	Plate (t_Q dof)	3/2	$\max\left\{\frac{2t_Q-7}{3(t_Q-3)}, \frac{3(t_O-1)}{4t_O}\right\}$	Theorem 7.3
Tetrahedra*	Tetrahedron	3/2	1/2	Theorem 7.1

Table 1 Summary of results. Storage and query time are $O^*(n^\alpha)$ and $O^*(n^\beta)$, respectively, and we specify the values of α and β for each result. The data structures for type (i) and (ii) intersection queries count the number of *intersection points* between the input objects and the query object, and not the number of input objects intersected by the query object.

* Counts the number of triangles/tetrahedra intersected by a query triangle/tetrahedron in $O^*(n^{5/9})$ time.

** This data structure does not extend to counting queries. In addition, the first term $\frac{2t_Q-7}{3(t_Q-3)}$ in the bound applies when t_Q is the maximum parametric dimension of the bounding *arcs* of the query plates; if each plate is bounded by a single endpoint-free curve, the first term in the bound becomes $\frac{2t_Q-3}{3(t_Q-1)}$.

approach that not only generalizes to algebraic arcs but also simplifies, in certain aspects, the technique of [19] for segment-intersection searching. Handling this much more general setup, using a battery of tools from range searching and real algebraic geometry, is one of the main technical contributions of this work (see Section 3). The most interesting among these tools is the construction of a carefully tailored *cylindrical algebraic decomposition* (CAD) (see [13, 15, 29] for details concerning this technique, which are also reviewed later in Section 3.1 in this work) of a suitable parametric space, where the CAD is induced by the partitioning polynomial. Consider a plate Δ and its supporting plane h_Δ. $\Delta \setminus Z(F)$ consists of several connected components. The CAD is used to further subdivide each component into smaller pieces (pseudo-trapezoids) and label each piece that is fully contained in the relative interior of Δ. The label is an explicit semi-algebraic representation of that piece, of constant complexity, that depends only on the equation of h_Δ (and not on Δ) and on the fixed polynomial F. These labels are used in a subsequent semi-algebraic range searching mechanism that detects the desired intersections for wide plates.

Next, we present a data structure for answering arc-intersection queries amid wide plates within a cell of the polynomial partition (see Section 4). It reduces the query time by increasing the storage used. Roughly, the improvement is a consequence of using a combined primal-dual range-searching approach, where the primal part works in object space, as in the aforementioned main algorithm. The dual part works in query space, regarding the query arc γ as a point $\tilde{\gamma}$ in $\mathbb{R}^t$, where t is the parametric dimension of the query arcs. Each input plate Δ is mapped to a semi-algebraic range $\tilde{\Delta}$ in query space, and the query reduces to a point-enclosure query that determines whether $\tilde{\gamma}$ lies in any of these semi-algebraic ranges $\tilde{\Delta}$. Specifically, we build a data structure of size $O^*(n^{3/2})$ with $O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$ query time, for

parametric dimensions $t \geq 3$.

Another contribution of this work is a general technique for reducing the parametric dimension t by 2, for *planar* query arcs, eliminating the dependence of the asymptotic query time bound on the endpoints of the arc (see Section 4.2). For example, if the query objects are circular arcs, their parametric dimension is eight (three for specifying the supporting plane, three for specifying the containing circle in that plane, and two for the endpoints). We show how to improve the query time from $O^*(n^{13/21})$ (the query time bound for $t = 8$) to $O^*(n^{3/5})$ (the bound for $t = 6$), with the same asymptotic storage complexity $O^*(n^{3/2})$. We note that $t = 6$ when the query objects are line segments in $\mathbb{R}^3$; by reducing this to $t = 4$, we get the query time $O^*(n^{5/9})$ for this case, which is slightly worse than the bound $O^*(n^{1/2})$ in [19]. This deterioration in the performance is the cost we pay for proposing a general approach that extends to query objects being arcs, as well as to answering counting queries. We leave it as an open problem whether the query bound of our approach can be further improved for general intersection-detection queries of the sort considered in this work.

Next, if $\mathcal{T}$ is a set of *triangles* in $\mathbb{R}^3$, we present an alternative near-linear-size data structure that can answer an arc-intersection query, for a constant-degree algebraic arc, in $O^*(n^{4/5})$ time (see Section 5). This is done by reducing arc-intersection searching to multi-level semi-algebraic range searching in a suitably defined sequence of five-dimensional parametric spaces and constructing a multi-level partition tree on the input set of triangles, to handle the resulting sequence of sub-queries. By using this structure at the bottom level of recursion in our main algorithm, we obtain a data structure of improved size $O^*(n^{11/9})$ that can still answer arc-intersection queries in time $O^*(n^{2/3})$.

Intersection searching with plates amid arcs. Next, we present data structures for the complementary setup where the input objects are arcs and we query with a plate (see Section 6). We first show that we can preprocess a set $\mathcal{T}$ of n line segments, in expected time $O^*(n^{3/2})$, into a data structure of size $O^*(n^{3/2})$, so that an intersection query with a plate can be answered in $O^*(n^{1/2})$ time. We then extend this result to the case where the input is a set of n arcs of (constant degree and) parametric dimension t, and the query object remains a plate. We obtain a data structure of size $O^*(n^{3/2})$ that can answer an intersection query in $O^*\left(n^{\frac{3(t-1)}{4t}}\right)$ time; see Theorem 6.4.

Intersection searching with plates amid plates.[4] The above results can be used to provide simple solutions for the case where both input and query objects are plates (see Section 7).

For simplicity, assume first that both input and query objects are triangles in $\mathbb{R}^3$. We observe that if a query triangle Δ intersects an input triangle Δ' then $\Delta \cap \Delta'$ is a line segment, and each of its endpoints is either an intersection of an edge of Δ with Δ' or of Δ with an edge of Δ'. The former (resp., latter) kind of intersection can be detected using type (i) intersection queries, of arcs amid plates (resp., type (ii) queries, of plates amid arcs). Using $O^*(n^{3/2})$ storage, this results in the query time bound $O^*(n^{1/2})$, if we use the data structure from [19] for type (i) queries. For counting queries, we have to use our arc-intersection data structure, leading to a query time of $O^*(n^{5/9})$.

The technique can be extended to the case where both input and query objects are arbitrary plates. In this case, the boundary of a plate consists of $O(1)$ algebraic arcs of constant complexity. Let t_O and t_Q be the parametric dimensions of the boundary arcs of

[4] This study was inspired by a question of Ovidiu Daescu related to collision detection in robotics.

input and query plates, respectively. We obtain a data structure of $O^*(n^{3/2})$ size with query time $O^*(n^\rho)$, where $\rho = \max\left\{\frac{2t_Q-7}{3(t_Q-3)}, \frac{3(t_O-1)}{4t_O}\right\}$.[5]

Our data structure for the plate-plate case also works if the input and query objects are constant-complexity, not necessarily convex three-dimensional polyhedra. This is because an intersection between two polyhedra occurs when their boundaries meet, unless one of them is fully contained in the other, and the latter situation can be easily detected. We can therefore just triangulate the boundaries of both input and query polyhedra and apply the triangle-triangle intersection-detection machinery.

The case of spherical caps. Finally, we present an application of our technique to an instance where the input objects are not flat (see Section 8). Specifically, we show how to answer segment-intersection queries amid spherical caps (each being the intersection of a sphere with a halfspace), using either a data structure with $O^*(n^{5/4})$ storage and $O^*(n^{3/4})$ query time, or a structure with $O^*(n^{3/2})$ storage and $O^*(n^{27/40})$ query time.

We conclude the paper with Section 9, which contains a brief discussion of our results and some open problems.

2 Intersection searching with query arcs amid plates

Let $\mathcal{T}$ be a set of n plates in $\mathbb{R}^3$, and let Γ be a family of algebraic arcs that has parametric dimension t for some constant $t \geq 3$. We present algorithms for preprocessing $\mathcal{T}$ into a data structure that can answer arc-intersection queries with arcs $\gamma \in \Gamma$ efficiently. We begin by describing a basic data structure, and then show how its performance can be improved.

2.1 The overall data structure

Our primary data structure consists of a partition tree Ψ on $\mathcal{T}$, which is constructed using the polynomial-partitioning technique of Guth [20]. More precisely, let $\mathcal{X} \subseteq \mathcal{T}$ be a subset of m plates and let $D > 1$ be a parameter. Using the result by Guth, a real polynomial F of degree at most $c_1 D$ can be constructed, where $c_1 > 0$ is an absolute constant, such that each open connected component (called a *cell*) of $\mathbb{R}^3 \setminus Z(F)$ is crossed by boundary arcs (which we refer to as *edges* from now on) of at most m/D^2 plates of $\mathcal{X}$ and by at most m/D plates of $\mathcal{X}$; the number of cells is at most $c_2 D^3$ for another absolute constant $c_2 > 0$. Agarwal et al. [3] showed that such a partitioning polynomial can be constructed in $O(m)$ expected time if D is a constant. Using such polynomial partitionings, Ψ can be constructed recursively in a top-down manner as follows.

Each node $v \in \Psi$ is associated with a cell τ_v of some partitioning polynomial and a subset $\mathcal{T}_v \subseteq \mathcal{T}$. If v is the root of Ψ, then $\tau_v = \mathbb{R}^3$ and $\mathcal{T}_v = \mathcal{T}$. Set $n_v = |\mathcal{T}_v|$. We set a threshold parameter $n_0 \leq n$, which may depend on n, and we fix a sufficiently large constant D. For the basic data structure described here, we set $n_0 = n^{1/3}$; the value of n_0 will change when we later modify the structure.

Suppose we are at a node v. If $n_v \leq n_0$ then v is a leaf and we store $\mathcal{T}_v$ at v. Otherwise, we construct a partitioning polynomial F_v of degree at most $c_1 D$, as described above, and store F_v at v. We construct a secondary data structure Σ_v^0 on $\mathcal{T}_v$ for answering arc-intersection

[5] The first term $\frac{2t_Q-7}{3(t_Q-3)}$ in the bound applies only when the query plate is bounded by more than one arc, of maximum parametric dimension t_Q. When the query plates are bounded by a single endpoint-free curve (such as circular or elliptical disks) with parametric dimension t_Q, the term becomes $\frac{2t_Q-3}{3(t_Q-1)}$.

queries with an arc $\gamma \in \Gamma$ that is contained in $Z(F_v)$. Σ_v^0 is constructed in an analogous manner as Ψ by using the polynomial-partitioning scheme of Agarwal et al. [3], which, given a (constant) parameter $D_1 \gg D$, constructs a polynomial G of degree at most $c_1 D_1$ so that each cell of $Z(F) \setminus Z(G)$ intersects at most n_v/D_1 plates of $\mathcal{T}_v$ and the boundaries of at most n_v/D_1^2 plates. Further details of Σ_v^0 are omitted from here (see [3]), and we conclude:

▶ **Proposition 2.1.** *For a partitioning polynomial F of sufficiently large constant degree and a set $\mathcal{T}$ of n plates, one can construct, in $O^*(n)$ expected time, a data structure of size $O^*(n)$ that can answer an arc-intersection query with an arc contained in $Z(F)$ in $O^*(n^{2/3})$ time.*

Next, we compute (semi-algebraic representations of) all cells of $\mathbb{R}^3 \setminus Z(F_v)$ [13]. Let τ be such a cell. We create a child w_τ of v associated with τ. We classify each plate $\Delta \in \mathcal{T}_v$ that crosses τ as *narrow* (resp., *wide*) at τ if an edge of Δ crosses τ (resp., Δ crosses τ, but none of its edges does). Let $\mathcal{W}_\tau$ (resp., $\mathcal{T}_\tau$) denote the set of the wide (resp., narrow) plates at τ. We construct a secondary data structure Σ_τ^1 on $\mathcal{W}_\tau$, as described in Section 3 below, for answering arc-intersection queries amid the plates of $\mathcal{W}_\tau$ with arcs of Γ that lie inside τ. Σ_τ^1 is stored at the child w_τ of v. The construction of Σ_τ^1 for handling the wide plates is the main technical step in our algorithm. By Proposition 3.2 in Section 3, Σ_τ^1 uses $O^*(|\mathcal{W}_\tau|)$ space, can be constructed in $O^*(|\mathcal{W}_\tau|)$ expected time, and answers an arc-intersection query in $O^*(|\mathcal{W}_\tau|^{2/3})$ time. Finally, we set $\mathcal{T}_{w_\tau} = \mathcal{T}_\tau$, and recursively construct a partition tree for $\mathcal{T}_{w_\tau}$ and attach it as the subtree rooted at w_τ. Note that two secondary structures are attached at each node v, namely, Σ_v^1 and Σ_v^0, for handling wide plates and for handling query arcs that are contained in $Z(F_v)$, respectively.

Denote by $S(m)$ the maximum storage used by the data structure for a subproblem involving at most m plates. For $m \le n_0$, $S(m) = O(m)$. For $m > n_0$, Propositions 2.1 and 3.2 imply that the secondary structures for a subproblem of size m require $O^*(m)$ space. Therefore $S(m)$ obeys the recurrence:

$$S(m) \le \begin{cases} c_2 D^3 S(m/D^2) + O^*(m) & \text{for } m \ge n_0, \\ O(m) & \text{for } m \le n_0, \end{cases} \tag{1}$$

where c_2 is the constant as defined above. Starting with $m = n$, we unfold the recurrence until we reach subproblems of size at most $n_0 = n^{1/3}$, that is, we reach level i with $n/D^{2i} \approx n^{1/3}$, or $D^i \approx O(n^{1/3})$. The overall storage that we accumulate so far, dominated by the storage at level i, is therefore

$$O^*\left(D^{3i} \cdot \frac{n}{D^{2i}}\right) = O^*(nD^i) = O^*(n^{4/3}).$$

At the bottom of the recursion the overall storage is also

$$O^*\left(D^{3i} \cdot \frac{n}{D^{2i}}\right) = O^*(nD^i) = O^*(n^{4/3}).$$

Therefore the solution of (1) is $S(n) = O^*(n^{4/3})$.

A similar analysis shows that the expected preprocessing time is also $O^*(n^{4/3})$.

2.2 The query procedure

Let $\gamma \in \Gamma$ be a query arc. We answer an arc-intersection query, say, intersection-detection, for γ by searching through Ψ in a top-down manner. Suppose we are at a node v of Ψ. Our goal is to determine whether $\gamma_v := \gamma \cap \tau_v$ intersects any plate of $\mathcal{T}_v$. For simplicity, assume that γ_v is connected, otherwise we query with each connected component of γ_v.

If v is a leaf, we answer the intersection query naïvely, in $O(n_0)$ time, by inspecting all plates in $\mathcal{T}_v$. If $\gamma_v \subset Z(F_v)$, then we query the secondary data structure Σ_v^0 with γ_v and return the answer. So assume that $\gamma_v \not\subset Z(F_v)$. We compute all cells of $\mathbb{R}^3 \setminus Z(F_v)$ that γ_v intersects; there are at most $c_3 D$ such cells for some absolute constant $c_3 > 0$ [13]. Let τ be such a cell. We first use the secondary data structure Σ_τ^1 to detect whether γ_v intersects any plate of $\mathcal{W}_\tau$, the set of wide plates at τ. We then recursively query at the child w_τ to detect an intersection between γ and $\mathcal{T}_\tau$, the set of narrow plates at τ.

For intersection-detection queries, the query procedure stops as soon as an intersection between γ and $\mathcal{T}$ is found. For reporting/counting queries, we follow the above recursive scheme, and at each node v visited by the query procedure, we either report all the plates of $\mathcal{T}_v$ intersected by the query arc, or add up the intersection counts returned by various secondary structures and recursive calls.

Denote by $Q(m)$ the maximum query time for a subproblem involving at most m plates. Then $Q(m) = O(m)$ for $m \le n_0$. For $m > n_0$, Propositions 2.1 and 3.2 imply that the query time of the auxiliary data structures for subproblems of size m is $O^*(m^{2/3})$. Therefore $Q(m)$ obeys the recurrence:

$$Q(m) \le \begin{cases} c_3 D Q(m/D^2) + O^*(m^{2/3}) & \text{for } m \ge n_0, \\ O(m) & \text{for } m \le n_0, \end{cases} \tag{2}$$

where c_3 is the constant as defined above. Unfolding the recurrence as in the storage analysis, one easily verifies that $Q(n) = O^*(n^{2/3})$. This is because up to level i the nonrecursive terms $O^*(m)$, over all visited nodes, add up to $O^*(n^{2/3})$, and at the bottom of the recurrence the brute-force search costs $O^*(D^i(n/D^{2i})) = O^*(n^{2/3})$ time too. Putting everything together we obtain:

▶ **Theorem 2.2.** *A given set $\mathcal{T}$ of n plates in $\mathbb{R}^3$ can be preprocessed, in expected time $O^*(n^{4/3})$, into a data structure of size $O^*(n^{4/3})$, so that an arc-intersection query amid the plates of $\mathcal{T}$ can be answered in $O^*(n^{2/3})$ time.*

2.3 Improved storage for triangles

In Section 5 we present a different technique for preprocessing a set $\mathcal{T}$ of n *triangles*, in expected time $O^*(n)$, into a data structure of $O^*(n)$ size that can answer arc-intersection queries in $O^*(n^{4/5})$ time. Using this data structure, we can modify our main structure Ψ, as follows: We set $n_0 = n^{5/9}$, i.e., a node v is a leaf if $n_v \le n^{5/9}$. We construct the structure of Section 5 at each leaf of Ψ. The recursion now terminates at depth i satisfying $n/D^{2i} \approx n^{5/9}$, or $D^i = n^{2/9}$. The overall storage is now $O^*(D^{3i} \cdot (n/D^{2i})) = O^*(nD^i) = O^*(n^{11/9})$. The query procedure is the same as above except that we use the query procedure of Section 5 at each leaf. The cost of a query is then the sum of $O^*(n^{2/3})$, for querying in the recursive structures for wide triangles, plus $O^*(D^i \cdot (n/D^{2i})^{4/5}) = O^*(n^{4/5}/D^{3i/5})$, for querying at the leaves of the structure; by our choice of i, the latter cost is also $O^*(n^{2/3})$. This can be shown to yield:

▶ **Theorem 2.3.** *A set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$ can be processed, in expected time $O^*(n^{11/9})$, into a data structure of size $O^*(n^{11/9})$, so that an arc-intersection query amid the triangles of $\mathcal{T}$ can be answered in $O^*(n^{2/3})$ time.*

2.4 Space/query-time trade-offs

As we show in Section 4.1, we can improve the query time for the secondary structure on wide plates by increasing the size of the structure. Specifically, we show that a set $\mathcal{W}_\tau$ of n wide plates at some partition cell τ can be preprocessed, in expected time $O^*(n^{3/2})$, into a data structure of size $O^*(n^{3/2})$, so that the query time improves to $O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$, where $t \geq 3$ is the parametric dimension of the query arcs. We adapt our primary data structure Ψ, as follows: (a) we now set n_0 to be a sufficiently large constant; and (b) we apply a standard primal-dual range-searching algorithm for the wide plates, instead of the primal-only approach of [24] used in the basic solution. We obtain similar recurrence relations for the storage complexity and query time as in (1) and (2), and plug into them the aforementioned modifications to the secondary data structure Σ^1_τ for the wide plates. It can be verified that the solutions are $S(n) = O^*(n^{3/2})$ and $Q(n) = O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$ for $t \geq 3$. We have thus shown:

▶ **Theorem 2.4.** *Let $\mathcal{T}$ be a set of n plates in $\mathbb{R}^3$, and let Γ be a family of arcs of parametric dimension $t \geq 3$. $\mathcal{T}$ can be preprocessed, in expected time $O^*(n^{3/2})$, into a data structure of size $O^*(n^{3/2})$, so that an arc-intersection query with an arc in Γ can be answered in $O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$ time.*

In Section 4.2 we present a further improvement in the query time for the case where the query arcs are *planar* algebraic arcs of constant degree. For example, for circular arcs, a naïve application of Theorem 2.4 would lead to a query time of $O^*(n^{13/21})$ ($t = 8$ in this case, see the introduction) with $O^*(n^{3/2})$ storage. We improve the query time bound, for general planar arcs, by eliminating the effect of the two parameters that specify the endpoints of the query arc γ, thereby effectively reducing the parametric dimension by 2. This improves the query time for circular arcs to $O^*(n^{3/5})$ (the bound for $t = 6$), still with storage and expected preprocessing time $O^*(n^{3/2})$ (see Proposition 4.4). A similar reduction holds for general planar query arcs, with similar improvements in the query time bounds. Specifically, we show:

▶ **Theorem 2.5.** *Let $\mathcal{T}$ be a set of n plates in $\mathbb{R}^3$, and let Γ be a family of planar arcs of parametric dimension $t \geq 4$. $\mathcal{T}$ can be preprocessed, in expected time $O^*(n^{3/2})$, into a data structure of size $O^*(n^{3/2})$, so that an arc-intersection query with an arc in Γ can be answered in $O^*\left(n^{\frac{2t-7}{3(t-3)}}\right)$ time. For circular arcs, where $t = 8$, the query time is $O^*(n^{3/5})$.*

▶ Remark. One example of this general technique is the case of vertical parabolas, describing the trajectories of stones thrown under gravity, say (see Sharir and Shaul [30] for an earlier study of this setup, where near-linear storage was used). Since a vertical parabola has $t = 5$ degrees of freedom (two for specifying its plane and three for its coefficients within that plane), we obtain $O^*(n^{7/12})$ query time, using $O^*(n^{3/2})$ storage. The queries improve further to $O^*(n^{5/9})$ if we just consider vertical parabolas that represent stone trajectories under the fixed constant of gravitation ($t = 4$ in this case), and deteriorates if we consider arbitrary parabolas.

3 Handling wide plates

Let $\mathcal{T}$ be a set of n plates in $\mathbb{R}^3$, Γ a family of arcs, and F a partitioning polynomial, as described in Section 2. In this section we describe the algorithm for preprocessing the set of wide plates, $\mathcal{W}_\tau$, for each cell τ of $\mathbb{R}^3 \setminus Z(F)$, for intersection queries with arcs of Γ. Fix a cell τ. Let $\Delta \in \mathcal{W}_\tau$ be a plate that is wide at τ, and let h_Δ be the plane supporting Δ. Since

Δ is wide at τ, each connected component of $\Delta \cap \tau$ is also a connected component of $h_\Delta \cap \tau$ (though some connected components of $h_\Delta \cap \tau$ may be disjoint from Δ). Roughly speaking, by a careful construction of a *cylindrical algebraic decomposition* (CAD) Ξ of F (see Section 3.2), we decompose $\Delta \cap \tau$ into $O(1)$ pseudo-trapezoids, each contained in a connected component of $\Delta \cap \tau$. We collect these pseudo-trapezoids of all wide plates at τ and cluster them into $O(1)$ families, using Ξ so that, for each family Φ, all pseudo-trapezoids within Φ can be represented by a fixed constant-complexity semi-algebraic expression (that is, predicate). Each such predicate only depends on F and on the (coefficients of the) plane supporting the pseudo-trapezoid φ (but not on the boundary of the plate containing φ). Roughly speaking, the predicate is of the form $\sigma(a, b, c, x, y)$, so that a plane $z = a_0x + b_0y + c_0$ contains a pseudo-trapezoid φ_σ so that $\sigma(a_0, b_0, c_0, x, y)$ holds precisely for those points (x, y, z) in that plane that lie in φ_σ; see Section 3.3. This semi-algebraic representation of Φ enables us to reduce the arc-intersection query on Φ to semi-algebraic range searching in only three parametric dimensions (see Section 3.4).

3.1 An overview of cylindrical algebraic decomposition

We begin by giving a brief overview of *cylindrical algebraic decomposition* (CAD), also known as Collins' decomposition, after its originator Collins [15]. This tool is a central ingredient of our algorithm—see Section 3.2. A detailed description can be found in [13, Chapter 5]; a possibly more accessible treatment is given in [29, Appendix A].

Given a finite set $\mathcal{F} = \{f_1, \ldots, f_s\}$ of d-variate polynomials, a cylindrical algebraic decomposition induced by $\mathcal{F}$, denoted by $\Xi(\mathcal{F})$, is a (recursive) decomposition of $\mathbb{R}^d$ into a finite collection of relatively open simply-shaped semi-algebraic cells of dimensions $0, \ldots, d$, each homeomorphic to an open ball of the respective dimension. These cells refine the arrangement $\mathcal{A}(\mathcal{F})$ of the zero sets of the polynomials in $\mathcal{F}$, as described next.

Set $F = \prod_{i=1}^{s} f_i$. For $d = 1$, let $\alpha_1 < \alpha_2 < \cdots < \alpha_t$ be the distinct real roots of F. Then $\Xi(\mathcal{F})$ is the collection of cells $\{(-\infty, \alpha_1), \{\alpha_1\}, (\alpha_1, \alpha_2), \ldots, \{\alpha_t\}, (\alpha_t, +\infty)\}$. For $d > 1$, regard $\mathbb{R}^d$ as the Cartesian product $\mathbb{R}^{d-1} \times \mathbb{R}$ and assume that x_d is a *good* direction, meaning that for any fixed $a \in \mathbb{R}^{d-1}$, $F(a, x_d)$, viewed as a polynomial in x_d, has finitely many roots.

$\Xi(\mathcal{F})$ is defined recursively from a 'base' $(d-1)$-dimensional CAD Ξ_{d-1}, as follows. One constructs a suitable set $\mathcal{E} := \mathcal{E}(\mathcal{F})$ of polynomials in $x_1, \ldots, x_{d-1}$ (denoted by $\textsc{Elim}_{X_k}(\mathcal{F})$ in [13] and by Q_b in [29]). Roughly speaking, the zero sets of polynomials in $\mathcal{E}$, viewed as subsets of $\mathbb{R}^{d-1}$, contain the projection onto $\mathbb{R}^{d-1}$ of all intersections $Z(f_i) \cap Z(f_j)$, $1 \le i < j \le s$, as well as the projection of the loci in each $Z(f_i)$ where $Z(f_i)$ has a tangent hyperplane parallel to the x_d-axis, or a singularity of some kind. The actual construction of $\mathcal{E}$, based on *subresultants* of $\mathcal{F}$, is somewhat complicated, and we refer to [13, 29] for more details.

One recursively constructs $\Xi_{d-1} = \Xi(\mathcal{E})$ in $\mathbb{R}^{d-1}$, which is a refinement of $\mathcal{A}(\mathcal{E})$ into topologically trivial open cells of dimensions $0, 1, \ldots, d-1$. For each cell $\tau \in \Xi_{d-1}$, the sign of each polynomial in $\mathcal{E}$ is constant (zero, positive, or negative) and the (finite) number of distinct real x_d-roots of $F(\mathbf{x}, x_d)$ is the same for all $\mathbf{x} \in \tau$. $\Xi(\mathcal{F})$ is then defined in terms of Ξ_{d-1}, as follows. Fix a cell $\tau \in \Xi_{d-1}$. Let $\tau \times \mathbb{R}$ denote the *cylinder* over τ. There is an integer $t \ge 0$ such that for all $\mathbf{x} \in \tau$, there are exactly t distinct real roots $\psi_1(\mathbf{x}) < \cdots < \psi_t(\mathbf{x})$ of $F(\mathbf{x}, x_d)$ (regarded as a polynomial in x_d), and these roots are algebraic functions that vary continuously with $\mathbf{x} \in \tau$. Let ψ_0, ψ_{t+1} denote the constant functions $-\infty$ and $+\infty$, respectively. Then we create the following cells that decompose the cylinder over τ:

- $\sigma = \{(\mathbf{x}, \psi_i(\mathbf{x})) \mid \mathbf{x} \in \tau\}$, for $i = 1, \ldots, t$; σ is a section of the graph of ψ_i over τ, and

- $\sigma = \{(\mathbf{x}, y) \mid \mathbf{x} \in \tau,\ y \in (\psi_i(\mathbf{x}), \psi_{i+1}(\mathbf{x}))\}$, for $0 \le i \le t$; σ is a portion ('layer') of the cylinder $\tau \times \mathbb{R}$ between the two consecutive graphs ψ_i, ψ_{i+1}.

The main property of Ξ is that, for each cell $\tau \in \Xi$, the sign of each polynomial in $\mathcal{F}$ is constant for all $\mathbf{x} \in \tau$. Omitting all further details (for which see [13, 15, 29]), we have the following lemma:

▶ **Lemma 3.1.** *Let $\mathcal{F} = \{f_1, \ldots, f_s\}$ be a set of s d-variate polynomials of degree at most D each. Then, assuming that all coordinates are good directions, $\Xi(\mathcal{F})$ consists of $O(Ds)^{2^d}$ cells, and each cell can be represented semi-algebraically by $O(D)^{2^d}$ polynomials of degree at most $O(D)^{2^{d-1}}$. $\Xi(\mathcal{F})$ can be constructed in time $(Ds)^{2^{O(d)}}$ in a suitable standard model of algebraic computation.*

3.2 Constructing a CAD of the partitioning polynomial

Let $\mathbb{E}_3$ denote the space of all planes in $\mathbb{R}^3$. More precisely, $\mathbb{E}_3$ is the (dual) three-dimensional space where each plane $h\colon z = ax+by+c$ is mapped to the point (a, b, c). For $(a_0, b_0, c_0) \in \mathbb{E}_3$, we use $h(a_0, b_0, c_0)$ to denote the plane $z = a_0x + b_0y + c_0$. We consider the five-dimensional parametric space $\mathbb{E} := \mathbb{E}_3 \times \mathbb{R}^2$ with coordinates (a, b, c, x, y). We construct in $\mathbb{E}$ a CAD of the single 5-variate polynomial $F(x, y, ax + by + c)$. We use a generic choice of coordinates to ensure that all the axes of the coordinate frame are in good directions for the construction of the CAD, coming up next. Such a generic choice of coordinates also allows us to assume that none of the input plates lies in a vertical plane.

The construction of the CAD recursively eliminates the variables in the order y, x, c, b, a. That is, unfolding the recursive definition given in Section 3.1, each cell of the CAD is given by a sequence of equalities or inequalities (one from each row) of the form:

$$\begin{array}{lcc}
a = a_0 & \text{or} & a_0^- < a < a_0^+ \\
b = f_1(a) & \text{or} & f_1^-(a) < b < f_1^+(a) \\
c = f_2(a, b) & \text{or} & f_2^-(a, b) < c < f_2^+(a, b) \\
x = f_3(a, b, c) & \text{or} & f_3^-(a, b, c) < x < f_3^+(a, b, c) \\
y = f_4(a, b, c; x) & \text{or} & f_4^-(a, b, c; x) < y < f_4^+(a, b, c; x),
\end{array} \tag{3}$$

where a_0, a_0^-, a_0^+ are real parameters, and $f_1, f_1^-, f_1^+, \ldots, f_4, f_4^-, f_4^+$ are constant-degree continuous algebraic functions (any of which can be $\pm\infty$), so that, whenever we have an inequality involving two reals or two functions, we then have $a_0^- < a_0^+$, and/or $f_1^-(a) < f_1^+(a)$, $f_2^-(a, b) < f_2^+(a, b)$, $f_3^-(a, b, c) < f_3^+(a, b, c)$, and $f_4^-(a, b, c; x) < f_4^+(a, b, c; x)$, over the cell defined by the preceding set of equalities and inequalities in (3).

As a simple illustration of this concept, consider the special case where only horizontal planes of the form $z = c$ are considered. The function is then $F(x, y, c)$, and the CAD is of the three-dimensional cxy-space. The CAD partitions the c-axis into intervals and delimiting points. For each c-interval C_0, the CAD over each $c \in C_0$ is a CAD partition of the xy-plane (which, by construction, is a decomposition of the plane into vertical pseudo-trapezoids, each given by a simpler version of the set of the last two equations or inequalities in (3)). This is in fact a refinement of the partition induced by the cross section of $Z(F)$ with the plane $z = c$, which has a fixed combinatorial structure, for each interval C_0, independent of $c \in C_0$. In other words, the topology of this refined partition of the cross section does not change as c varies in C_0. At a delimiting endpoint c, the combinatorial representation of the cross section changes, which implies a change in its topology. In other words, the CAD in this

case is a refinement of the Morse structure for $Z(F)$ with respect to the z-height (or rather c-height) function; see [25] for a classical reference for Morse theory.

Returning to the construction of CAD for the general case, let $\Xi_5 = \Xi_5(F)$ denote the five-dimensional CAD just defined, and let Ξ_3 denote the projection of Ξ_5 onto $\mathbb{E}_3$, which we refer to as the *base* of Ξ_5 and which itself is a CAD of a suitable set of polynomials. Each base cell of Ξ_3 is given by a set of equalities and inequalities from the first three rows of (3), one per row. For a point $(a_0, b_0, c_0) \in \mathbb{E}_3$, let $\Xi^2(a_0, b_0, c_0)$ denote the decomposition in the xy-subspace that is induced by Ξ_5 over (a_0, b_0, c_0). This is the decomposition of the xy-plane into pseudo-trapezoids, each of which is given by equalities and/or inequalities from the last two rows of (3), with $a = a_0$, $b = b_0$, $c = c_0$. We refer to $\Xi^2(a_0, b_0, c_0)$ as the two-dimensional *fiber* of Ξ_5 over (a_0, b_0, c_0). As a matter of fact, and this is the main rationale for the CAD construction, $\Xi^2(a_0, b_0, c_0)$ can be identified with the xy-projection of a refinement of the partition induced by $Z(F)$ in the plane $h(a_0, b_0, c_0)$. That is, each 2-cell of this two-dimensional fiber of Ξ_5 is contained in the projection of a single connected component of $h(a_0, b_0, c_0) \setminus Z(F)$, and each 0-cell, as well as each 1-cell that is not y-vertical, of the fiber is contained in the projection of a portion of $Z(F) \cap h(a_0, b_0, c_0)$. See Figure 1 for an illustration.

Moreover, similar to the Morse theory example mentioned above, the topology of the partition induced by $Z(F)$ in $h(a_0, b_0, c_0)$ does not change as long as (a_0, b_0, c_0) stays in the same cell C_0 of Ξ_3, and changes in the topology occur only when we cross between cells of Ξ_3. In particular, each cell C of Ξ_5 can be associated with a fixed cell of $\mathbb{R}^3 \setminus Z(F)$, denoted as τ_C, such that for all points (a_0, b_0, c_0) in the base cell $C^{\downarrow} \subset \mathbb{E}_3$ of C, which is the projection of C onto $\mathbb{E}_3$, the two-dimensional portion C^2 of the fiber $\Xi^2(a_0, b_0, c_0)$ for which $\{(a_0, b_0, c_0)\} \times C^2 \subseteq C$ is the xy-projection of a pseudo-trapezoid of a connected component of $h(a_0, b_0, c_0) \cap \tau_C$. This property will be useful in constructing the data structure to answer arc-intersection queries amid the wide plates at τ.

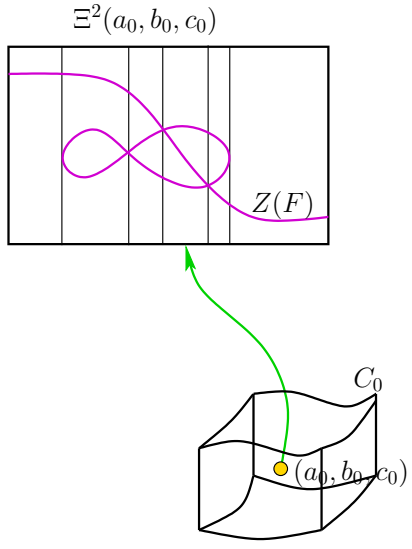

Figure 1 An illustration of the CAD construction. C_0 is a three-dimensional cell of Ξ_3. For a point $(a_0, b_0, c_0) \in C_0$, its two-dimensional fiber $\Xi^2(a_0, b_0, c_0)$ is shown. Formally, the purple curve is the xy-projection of $Z(F) \cap h(a_0, b_0, c_0)$.

3.3 Decomposing wide plates into pseudo-trapezoids

We are now ready to describe how to decompose each plate $\Delta \in \mathcal{W}_\tau$, for each cell τ of $\mathbb{R}^3 \setminus Z(F)$, into pseudo-trapezoids, and how to cluster the resulting pseudo-trapezoids. Let $\Delta \in \mathcal{T}$ be a plate, let h_Δ be the plane supporting Δ, and let $\Delta^* = (a_0, b_0, c_0)$ be the point in the *abc*-subspace $\mathbb{E}_3$ dual to h_Δ. We locate (in constant time, by brute force) the cell C_0 of Ξ_3 (in $\mathbb{E}_3$) that contains Δ^*. Let φ be a cell of $\Xi^2(\Delta^*)$, let $\varphi^\uparrow = \{(x, y, a_0x+b_0y+c_0) \mid (x, y) \in \varphi\}$ be the lifting of φ onto h_Δ, and let C be the cell of Ξ_5 that contains $\{\Delta^*\} \times \varphi$. We determine whether $\varphi^\uparrow$ is fully contained in Δ, lies fully outside Δ, or intersects $\partial\Delta$. We keep φ only if $\varphi^\uparrow$ is contained in Δ, and associate $\varphi^\uparrow$, as well as the plate Δ, with C. (In general, Δ is associated with many cells C, one for each cell φ of $\Xi^2(\Delta^*)$ whose lifting is contained in Δ.) In this case, we use Δ_C to denote the pseudo-trapezoid $\varphi^\uparrow$, which is uniquely determined by Δ and C and which lies in a connected component of $\Delta \cap \tau_C$. For a cell $C \in \Xi_5$, let $\mathcal{T}_C \subseteq \mathcal{T}$ be the subset of plates that are associated with C, and let $\Phi_C = \{\Delta_C \mid \Delta \in \mathcal{T}_C\}$ be the subset of pseudo-trapezoids associated with C. Finally, for a plate $\Delta \in \mathcal{T}$, let Ξ_Δ be the set of all cells of Ξ_5 with which Δ is associated. Again, see Figure 1 for an illustration.

The advantage of this approach is that for each plate $\Delta \in \mathcal{T}$, the set $\Delta^\parallel := \{\Delta_C \mid C \in \Xi_\Delta\}$ is a refinement into pseudo-trapezoids of those cells of $h_\Delta \setminus Z(F)$, referred to as *inner* cells, that lie fully inside Δ. Furthermore, the set Ξ_Δ provides an operational "labeling" scheme for the pseudo-trapezoids in $\Delta^\parallel$ — the pseudo-trapezoid Δ_C is labeled with C, or rather with the semi-algebraic representation that it inherits from C. That is, each such pseudo-trapezoid $\varphi^\uparrow$ on the plate Δ, with the point Δ^* belongs to some base cell C_0 of Ξ_3, is represented by equalities and inequalities of the form

$$x = f_3(\Delta^*) \quad \text{or} \quad f_3^-(\Delta^*) < x < f_3^+(\Delta^*) \quad \text{and} \quad y = f_4(\Delta^*) \quad \text{or} \quad f_4^-(\Delta^*) < y < f_4^+(\Delta^*),$$

where $f_3, f_3^-, f_3^+, f_4, f_4^-, f_4^+$ are constant-degree continuous algebraic functions over the corresponding domains, as in (3). This is a simple semi-algebraic representation, of constant complexity, of the xy-projection φ of $\varphi^\uparrow$, which *does not explicitly depend on* Δ (but only on its plane h_Δ). Moreover, this representation is fixed for all plates Δ for which the points Δ^* lie in the same cell of Ξ_3, and is therefore also independent of h_Δ,[6] as long as Δ^* belongs to that cell. See Figure 2 for an illustration. This constant-size "labeling" is used for clustering the pseudo-trapezoids into which the inner cells of h_Δ, for $\Delta \in \mathcal{T}$, are partitioned. Namely, we put all pseudo-trapezoids labeled with the same cell C of Ξ_5 into one cluster, and $\{\Phi_C \mid C \in \Xi_5\}$ is the desired clustering of the pseudo-trapezoids.

3.4 Reduction to semi-algebraic range searching

Fix a cell C of Ξ_5. For an arc $\gamma \in \Gamma$, contained in the cell τ_C of $\mathbb{R}^3 \setminus Z(F)$, we wish to answer an arc-intersection query on Δ_C with γ. To this end, we define the predicate $\Pi_C \colon \Gamma \times \mathbb{E}_3 \to \{0, 1\}$ so that $\Pi_C(\gamma; a, b, c)$ is 1 if and only if γ crosses $h(a, b, c)$ at a point (x, y, z) such that (x, y) belongs to C (that is, $(a, b, c, x, y) \in C$), and (x, y, z) lies in τ_C. It is easy to verify that $\Pi_C(\gamma; a, b, c)$ is a semi-algebraic predicate of constant complexity (that depends on D and t, the parametric dimension of arcs in Γ). We now define the semi-algebraic range

$$Q_{C,\gamma} := \{(a, b, c) \mid \Pi_C(\gamma; a, b, c) = 1\}, \tag{4}$$

[6] More precisely, its dependence on h_Δ is only in terms of the coeffcients (a, b, c) of h_Δ that are substituted in the fixed semi-algebraic predicate given above.

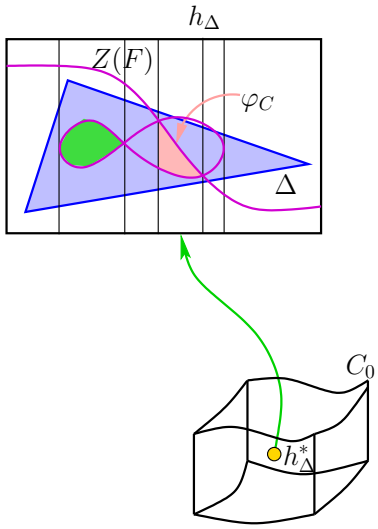

Figure 2 The labeling scheme provided by the CAD (the plate depicted in this figure is a triangle). The cell C labels, by an explicit semi-algebraic expression, the highlighted inner pseudo-trapezoidal subcell φ_C within the plate Δ. Another inner subcell, with a different label, in a different partition cell τ, is also highlighted.

which is of constant complexity too. By construction, γ crosses Δ_C if and only if the point $\Delta^* \in Q_{C,\gamma}$. Set $\mathcal{T}^*_C := \{\Delta^* \mid \Delta \in \mathcal{T}_C\}$.

▶ Remark. The semi-algebraic predicate Π_C can be replaced by b predicates $\Pi^{(i)}_C$, for $i = 1, \ldots, b$, where b is the maximum number of intersections of a query arc with a plane, so that $\Pi^{(i)}_C(\gamma; a, b, c)$ asserts that (is equal to 1 when) γ meets $h(a, b, c)$ at exactly i points that belong to the region labeled by C. These predicates, which are formed using quantifiers that can then be eliminated, are also of constant complexity, albeit larger than that of Π_C. This enhancement will be used for answering intersection-counting queries.

For each cell $C \in \Xi_5$, we preprocess $\mathcal{T}^*_C \subset \mathbb{E}_3$, in $O(|\mathcal{T}_C| \log n)$ expected time, into a data structure Σ_C of size $O(|\mathcal{T}_C|)$, using the range-searching mechanism of Matoušek and Patáková [24] (see also [8]). For a query range $Q_{C,\gamma}$, the range query on $\mathcal{T}^*_C$ can be answered in $O^*(|\mathcal{T}_C|^{2/3})$ time.

Finally, for a cell τ of $\mathbb{R}^3 \setminus Z(F)$, let

$$\Xi_\tau = \{C \in \Xi_5 \mid \tau_C = \tau\}$$

be the set of all CAD cells associated with τ. We store the structures Σ_C, for all $C \in \Xi_\tau$, at τ as the secondary structure Σ^1_τ. To test whether an arc $\gamma \in \Gamma$, which lies inside τ, intersects a plate of $\mathcal{W}_\tau$, we query each of the structures Σ_C stored at τ with $Q_{C,\gamma}$ and return yes if any of them returns yes. Putting everything together, we obtain the following:

▶ **Proposition 3.2.** *A set $\mathcal{W}$ of n wide plates at some cell τ can be preprocessed into a data structure of size $O^*(n)$, in $O^*(n)$ expected time, so that an arc-intersection query, for intersections within τ, can be answered in $O^*(n^{2/3})$ time.*

This at last completes the analysis for the wide plates, which implies the main result of this paper.

4 Space/query-time trade-off for arc-intersection queries

In this section we show that the query-time of arc-intersection searching amid plates can be improved by increasing the size of the data structure. As above, let $\mathcal{T}$ be a set of n plates in $\mathbb{R}^3$ and Γ a family of algebraic arcs of parametric dimension $t \geq 3$. We first describe the data structure for general algebraic arcs in $\mathbb{R}^3$ of constant parametric dimension (see Section 4.1). Recall that the parameter t depends on whether the arcs in Γ are proper bounded portions of their containing curves, with two delimiting endpoints, or full (close or unbounded) algebraic curves. If we need t parameters to specify the arc, $t-2$ parameters are enough to specify the curve. In Section 4.2, we show that the effect of these two extra parameters can be eliminated if Γ is a family of *planar* algebraic (bounded) arcs, thereby improving the performance bounds further.[7]

4.1 The case of general arcs

In this subsection we present a data structure for arc-intersection searching amid plates by working in query space, where a query arc γ, with parametric dimension t, is represented as a point q_γ in $\mathbb{R}^t$ and each wide plate Δ is represented as a constant-complexity semi-algebraic region Q_Δ which does not depend on $\partial\Delta$, as in the main algorithm. Here too we use the CAD construction to label all possible intersections between an arc and a plate. An arc-intersection query is now formulated as a point-enclosure query, namely, determining whether the query point q_γ lies in any of the regions Q_Δ. This latter task can be solved using the technique of Agarwal et al. [3], which, for n input regions, answers a point-enclosure query in $O(\log n)$ time using $O^*(n^t)$ storage. We actually use a primal-dual approach, in a manner detailed shortly, that combines the point-enclosure data structure with the range searching data structure described in the previous sections.

Processing arcs with t degrees of freedom.

For each cell C of the full CAD Ξ_5 and each query arc γ, we consider the corresponding semi-algebraic predicate $\Pi_C(\gamma; a, b, c)$, as defined in Section 3. We map γ to the point $q_\gamma \in \mathbb{R}^t$ whose coordinates are the t parameters specifying γ. For each input plate $\Delta \in \mathcal{T}$ whose supporting plane h_Δ is stored at C, we map Δ to the region

$$Q^*_{C,\Delta} = \{q_\gamma \in \mathbb{R}^t \mid \Pi_C(\gamma; a, b, c) = 1\},$$

where (a, b, c) is the parameterization of h_Δ. $Q^*_{C,\Delta}$ is a semi-algebraic region in $\mathbb{R}^t$ of constant complexity. We denote by $\mathcal{T}^*_C$ the collection of all these regions.

As before, for each cell C of Ξ_5, there is a unique partition cell τ_C with the following property. For any plane $h = h(a, b, c)$, where (a, b, c) lies in the abc-projection of C, the fiber $\Xi_2(a, b, c)$ contains a trapezoidal cell ξ such that $\{(a, b, c)\} \times \xi \subset C$, and the copy of ξ lifted to h is contained in τ_C. We refer to τ_C as the partition cell associated with C.

Let γ be a query arc in Γ. We process the cells τ of the polynomial partition crossed by γ, in their order along γ. [8] Let τ be such a cell, and assume for simplicity, as before, that γ is fully contained in τ (simply consider each maximal connected portion of γ within τ).

---

[7] For semi-unbounded arcs (such as rays), $t-1$ parameters specify the containing curve. The analysis in Section 4.2 can be applied for such arcs too, with a straightforward adaptation of the analysis. It yields a different, albeit similar improvement, but we omit the details of this variant.

[8] The order is important for arc-shooting queries, but is immaterial for the other kinds of intersection queries.

We iterate over all CAD cells C for which $\tau = \tau_C$. Fix such a cell C. By construction, if Δ is a wide plate at τ that is included in $\mathcal{T}_C$ then $q_\gamma \in Q^*_{C,\Delta}$ if and only if γ crosses Δ at a point (x, y, z) such that $(a, b, c, x, y) \in C$ (which also holds if and only if $\Delta^* \in Q_{C,\gamma}$ in our primal parametric three-space).

Our primal-dual approach proceeds as follows. The top part of the structure implements the semi-algebraic range-searching technique of [8, 24] in the object three-space (as in Section 2.1). We stop the recursive construction prematurely, when the size of the subproblems reaches some threshold value, determined by the storage we are willing to allocate to the structure, and then solve each of these subproblems in the query t-space, using the point-enclosure technique of [3] in the setup described above. Using standard arguments in range searching (see, e.g., [2, 5]), this results in a structure that, with storage parameter s, answers an arc-intersection query in $O^*\left(n^{\frac{2t}{3(t-1)}}/s^{\frac{2}{3(t-1)}}\right)$ time.

We then need to repeat this for every cell C for which $\tau = \tau_C$, and later for other partition cells τ that γ crosses.[9] As there are only $O(1)$ such repetitions (recalling that D is constant), this does not affect the asymptotic performance bounds.

The overall structure.

We run the recursive polynomial partitioning machinery until we reach a level k satisfying $D^k \approx s/n$; to simplify the calculations, assume that $D^k = s/n$. We obtain $O^*(D^{3k})$ subproblems, each with at most n/D^{2k} plates. Starting with storage parameter s, each subproblem, at any recursive level j, is allocated the storage parameter s/D^{3j} for its at most n/D^{2j} plates. Note that at the bottom of recursion we have $s/D^{3k} = n/D^{2k}$, so $s/D^{3j} \geq n/D^{2j}$ at each intermediate level j. Hence, answering a query on the wide plates at τ, with this value of the storage parameter, takes time[10]

$$O^*\left(\frac{(n/D^{2j})^{\frac{2t}{3(t-1)}}}{(s/D^{3j})^{\frac{2}{3(t-1)}}}\right) = O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}} D^{\frac{2(2t-3)j}{3(t-1)}}}\right).$$

Summing over all levels j, the total cost so far is $O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}}}\right)$, for $t \geq 2$.

At the bottom level of the recursion, we handle the plates by a brute force inspection, i.e., in $O(n/D^{2k}) = O(n^3/s^2)$ time per leaf subproblem. Thus, summing over the $O^*(s/n)$ leaves that the query reaches, the total cost of a query is $O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}}} + \frac{n^2}{s}\right)$. If we set $s = n^{3/2}$, the cost of a query is $O^*\left(n^{\frac{2t-3}{3(t-1)}} + n^{1/2}\right) = O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$, for $t \geq 3$. For example, for full circles, where $t = 6$, the query time becomes $O^*(n^{3/5})$, and for circular arcs, where $t = 8$, it is $O^*(n^{13/21})$ (both with $s = n^{3/2}$). One can of course allocate less storage and get a larger query time, with the tradeoff given above. In summary, we have

▶ **Proposition 4.1.** *A set of n plates in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(s)$, in expected time $O^*(s)$, for any prescribed parameter $n \leq s \leq n^3$, so that*

[9] As already noted, for intersection detection and arc-shooting queries, there is no need to perform this repetition once an intersection has been detected.

[10] The factors hidden in the $O^*(\cdot)$ notation depend on D but not on j, which is important in bounding the query time.

intersection queries with algebraic arcs of parametric dimension $t \geq 3$ can be answered in time $O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}}} + \frac{n^2}{s}\right)$. For $s = n^{3/2}$, the query cost is $O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$.

▶ Remark. The first term in the bound $O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}}} + \frac{n^2}{s}\right)$ dominates the second term when $s \geq n^{\frac{4t-6}{3t-5}}$. For smaller values of s the second term dominates. For example, for circles (with $t = 6$) the threshold value is $s = n^{18/13}$.

Further improvement for triangles. In the previous approach, we handled the plates at the bottom of the recursion by a brute-force inspection. As already mentioned, this can be improved for triangles. As we show in Section 5, we can construct, for n triangles, a data structure of size $O^*(n)$, in $O^*(n)$ time, that can answer an arc-intersection query in $O^*(n^{4/5})$ time. We construct this structure at each cell at the bottom level of the recursion. The overall storage and expected preprocessing cost of this structure are both $O^*(D^{3k} \cdot (n/D^{2k})) = O^*(nD^k) = O^*(s)$. The cost of querying this structure at a single leaf cell is $O^*\left((n/D^{2k})^{4/5}\right) = O^*\left(n^{4/5}/D^{8k/5}\right)$, so, summed over the $O^*(D^k)$ cells crossed by the query arc, yields the total cost $O^*\left(\frac{n^{4/5}}{D^{3k/5}}\right) = O^*\left(\frac{n^{7/5}}{s^{3/5}}\right)$. In conclusion, the overall cost of a query is $O^*\left(\frac{n^{\frac{2t}{3(t-1)}}}{s^{\frac{2}{3(t-1)}}} + \frac{n^{7/5}}{s^{3/5}}\right)$. Comparing the two terms, we see that the first term dominates the bound when $s \geq n^{\frac{11t-21}{9t-19}}$ and the second term dominates when $s \leq n^{\frac{11t-21}{9t-19}}$. That is, as in the general case, when s is smaller than that threshold, we can improve upon the bound obtained by the simpler approach described above.

4.2 A further improvement for planar query arcs

In this subsection we present a further improvement in the query time for the case where the query arcs are planar algebraic arcs of constant degree. For example, for circular arcs, a naïve application of Theorem 2.4 would use $t = 8$ (which is the parametric dimension of circular arcs, as already noted in the introduction), and the current improvement amounts to reducing t to 6, by getting rid of the two parameters that specify the endpoints of the query arc γ. This improves the query time from $O^*(n^{13/21})$ (which is the bound for $t = 8$) to $O^*(n^{3/5})$ (the bound for $t = 6$), still with storage and preprocessing time $O^*(n^{3/2})$ (see Proposition 4.4). A similar reduction holds for general planar query arcs, with similar t-dependent improvements in the query time bounds.

To simplify the presentation, we consider first the case of circular arcs, and then discuss the fairly straightforward extension to general planar arcs. Extending it further to nonplanar arcs remains an open problem.

The case of circular arcs.

We show how to effectively reduce the parametric dimension t for circular arcs from eight to six. To do that, we construct a multi-level structure where each level uses at most six of the eight parameters specifying a query arc γ. Concretely, each level uses either the circle c_γ containing γ, or an endpoint of γ, or some other feature of γ with no more than six degrees of freedom.

For technical reasons that will be clear shortly, we assume that γ is directed, and slightly modify the definition of the semi-algebraic predicate Π_C, so that $\Pi_C(\gamma; a, b, c)$ is 1 if and only if γ crosses $h(a, b, c)$ at a point (x, y, z) such that $(a, b, c, x, y) \in C$, and (x, y, z) is the *first crossing* of γ with $h(a, b, c)$. Here too, one can use quantifiers (and then eliminate them) to express (the modified) $\Pi_C(\gamma; a, b, c)$ as a semi-algebraic predicate of constant (albeit larger) complexity.

In what follows we assume that the angular span of γ is not too large, say at most $10°$. Longer arcs are simply broken into $O(1)$ smaller subarcs with this property, and the following procedure is applied to each subarc in order.

The approach is to replace γ by a directed semicircle $\hat{\gamma} \supseteq \gamma$ whose endpoints belong to a set of $O(1)$ canonical points, uniformly and sufficiently densely placed along c_γ, in such a manner that $\hat{\gamma}$ intersects a plane h_Δ so that its first intersection point belongs to C (in the above sense) if and only if γ intersects h_Δ so that its first intersection point belongs to C. The advantage of this replacement is that $\hat{\gamma}$ has only six degrees of freedom, modulo the constantly many choices of its endpoints, as opposed to γ which has eight. The disadvantage is the possibility of false-positive answers: we want to ensure that any detected (first) intersection of $\hat{\gamma}$ with h_Δ is also an intersection of γ with h_Δ.

So let E_γ be a set of $O(1)$ canonical points,[11] uniformly and sufficiently densely placed along c_γ. For example, put in E_γ the point w_0 on c_γ with the smallest x-coordinate, say, and then place in E_γ all the points w_j that lie at respective angular distances $2\pi j/k$ from w_0, for $j = 1, \ldots, k-1$, where k is some sufficiently large constant.[12]

The following two cases can arise:

(i) The endpoints of γ lie on the same side of h_Δ. This is the simplest situation. Informally, either γ points towards h_Δ, in a sense to be made precise shortly, from both its endpoints (as depicted in Figure 3), or γ points away from h_Δ from both its endpoints, or γ moves monotonically towards (or away from) h_Δ, so that the distance to h_Δ increases (or decreases) monotonically. In the second and third scenarios we conclude right away that γ does not hit h_Δ, but additional effort is needed to determine whether it hits h_Δ in the first scenario.

We use the following lemma.

▶ **Lemma 4.2.** *Let γ be a directed circular arc, whose angular span is at most $10°$. Let π_γ be the plane supporting γ, let c_γ be the circle supporting γ, and let p_γ, q_γ be the initial and terminal endpoints of γ, respectively. Let h be a plane with the property that p_γ and q_γ lie on the same side of h.*

Let $\hat{\gamma}$ be a semicircle that contains γ, and let $\hat{p}_\gamma$ and $\hat{q}_\gamma$ be the initial and terminal endpoints of $\hat{\gamma}$ (so that the points $\hat{p}_\gamma$, p_γ, q_γ and $\hat{q}_\gamma$ appear in this order along $\hat{\gamma}$). Then γ crosses h if and only if

(a) *$\hat{\gamma}$ crosses h, and*

(b) *both tangents $\tau_\gamma(p_\gamma)$, $\tau_\gamma(q_\gamma)$ to γ at p_γ and q_γ, respectively, oriented towards γ, point towards h, meaning that the angles between the normal to h that points away from p_γ, q_γ and the tangents $\tau_\gamma(p_\gamma)$ and $\tau_\gamma(q_\gamma)$ are both acute; see Figure 3.*

[11] These points are not 'globally canonical' as they depend on c_γ, but they do not depend on the endpoints of γ.

[12] The construction has to be modified in straightforward ways for an arc γ contained in plane parallel to the yz-plane, for example, by subdividing the circle c_γ at equispaced points, as follows: Add to E_γ the point w_0 on c_γ with the smallest z-coordinate and then proceed as above.

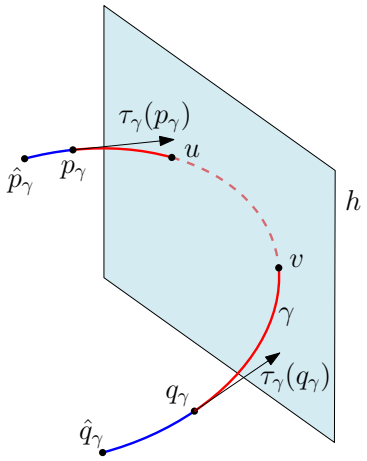

Figure 3 Condition (b) in Lemma 4.2.

When this is the case, the first intersection point of γ with h is also the first intersection point of $\hat{\gamma}$ with h.

Proof. Assume first that γ crosses h, necessarily at two points u, v; the case where γ is tangent to h can be handled by a slight variant of the same argument. Property (a) holds trivially. For (b), assume that the points p_γ, u, v, and q_γ appear in this order along γ. The planes π_γ and h meet at a line $\ell = uv$. Consider the tangents $\tau_\gamma(u)$, $\tau_\gamma(v)$ to γ at u, v, respectively, that point to the other side of h. Since the angular span of γ is at most $10°$, and γ has a tangent (at a point between u and v) that is parallel to h, it follows that the tangents $\tau_\gamma(p_\gamma)$, $\tau_\gamma(q_\gamma)$ also point towards h, and (b) holds.

Conversely, assume that (a) and (b) hold but γ does not cross h. Then the complementary arc γ^c of γ crosses h twice, and let u and v be the two corresponding intersection points. If at least one of the tangents $\tau_{\gamma^c}(p_\gamma)$, $\tau_{\gamma^c}(q_\gamma)$ points towards h then the corresponding tangent $\tau_\gamma(p_\gamma)$ or $\tau_\gamma(q_\gamma)$ points away from h, which contradicts the assumption in (b). Hence both tangents $\tau_{\gamma^c}(p_\gamma)$, $\tau_{\gamma^c}(q_\gamma)$ point away from h, which is impossible since γ^c has a tangent parallel to h, and the angular span of each of the two portions of γ^c that are delimited at this tangency point would then have to be larger than π, which is clearly impossible. See Figure 4 for an illustration.

The final claim in the lemma is clear because, by construction, the portions of $\hat{\gamma}$ between $\hat{p}_\gamma$ and p_γ, and between $\hat{q}_\gamma$ and q_γ are disjoint from h. ◀

▶ Remark. We note that the condition that both tangents $\tau_\gamma(p_\gamma)$, $\tau_\gamma(q_\gamma)$ point towards h is crucial. Indeed, it is possible that γ does not intersect h while $\hat{\gamma}$ does, but in such a case one of the tangents $\tau_\gamma(p_\gamma)$, $\tau_\gamma(q_\gamma)$ must point away from h.

We therefore proceed as follows. We prepare a multi-level data structure, where the first five levels collect canonical sets of wide plates, so that all plates Δ in any such set are such that γ and h_Δ satisfy (a) and (b) of Lemma 4.2, for some canonical choice of a semicircle $\hat{\gamma} \supseteq \gamma$. The first two levels ensure that p_γ and q_γ lie on the same side of h_Δ, the next two levels ensure that the tangents $\tau_\gamma(p_\gamma)$, $\tau_\gamma(q_\gamma)$ point towards h, and the fifth level ensures that $\hat{\gamma}$ crosses h_Δ. The number of degrees of freedom that a query object has is three for the first two levels, two for the next two levels (a direction in three-space has two degrees of freedom, and testing whether a direction forms an acute angle with a query direction is easily reduced to halfplane, or rather hemisphere, range searching), and six for the fifth level.

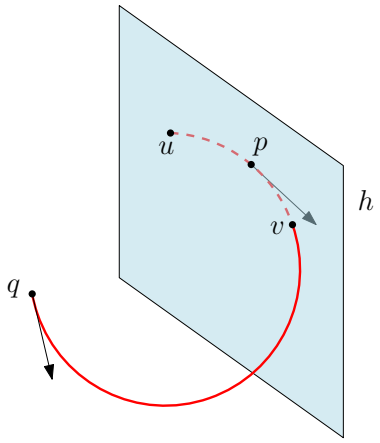

Figure 4 Illustration of the property that a circular arc pq, that starts behind h with a tangent parallel to h and ends in front of h with a tangent pointing away from h, must have angular span greater than π.

We then apply the machinery presented in Section 4.1 to each of the canonical sets produced at the fifth level except that now we replace γ by its containing semicircle $\hat{\gamma}$. We actually prepare a separate data structure for each of the $O(1)$ choices of the endpoints of $\hat{\gamma}$. In each such structure we modify the definition of the range $Q_{C,\gamma}$ in (4), so that it involves $\hat{\gamma}$ instead of γ. In this modification, the endpoints of $\hat{\gamma}$, instead of being two additional parameters, are expressed in terms of c_γ and the specific discrete choice of $\hat{\gamma}$ along that circle.[13] Thus $Q_{C,\hat{\gamma}}$ has only $t = 6$ degrees of freedom (in its dependence on $\hat{\gamma}$).

This leads to an algorithm that uses $s = O^*(n^{3/2})$ storage and answers a query in $O^*\left(n^{\frac{2t}{3(t-1)}}/s^{\frac{2}{3(t-1)}}\right) = O^*(n^{3/5})$ time (with $t = 6$).

(ii) The endpoints of γ lie on opposite sides of h_Δ. Again, we replace γ by some canonical directed semicircle $\hat{\gamma}$ that contains it, whose endpoints $\hat{p}_\gamma$ and $\hat{q}_\gamma$ both belong to E_γ.

It is easily checked that at least one of the portions $\hat{\gamma}^-(p)$ and $\hat{\gamma}^+(q)$ of $\hat{\gamma}$, between $\hat{p}_\gamma$ and p_γ, and between $\hat{q}_\gamma$ and q_γ, respectively, does not meet h_Δ. When $\hat{p}_\gamma$ and $\hat{q}_\gamma$ also lie on opposite sides of h_Δ, this holds for both portions $\hat{\gamma}^-(p)$, $\hat{\gamma}^+(q)$. When $\hat{p}_\gamma$ and $\hat{q}_\gamma$ lie on the same side of h_Δ, this holds for $\hat{\gamma}^-(p)$ (resp., $\hat{\gamma}^+(q)$) if p_γ (resp., q_γ) is the endpoint of γ that lies on that side.

If this endpoint is p_γ, we use the same machinery as above, of range searching with the region $Q_{C,\hat{\gamma}}$, with the original orientation of $\hat{\gamma}$. When the relevant endpoint is q_γ, we use $Q_{C,\hat{\gamma}}$, with the orientation of $\hat{\gamma}$ reversed (from $\hat{q}_\gamma$ to $\hat{p}_\gamma$).

We recall the peculiar way in which $Q_{C,\gamma}$ has been (re-)defined at the beginning of this section, that is, that we consider the first crossing of γ with $h(a, b, c)$. The situation we want to avoid is that $\hat{\gamma}$ crosses Δ at a point (x, y, z) (with $(a, b, c, x, y) \in C$), but this is its second crossing point with h_Δ, and this point does not belong to γ, whereas its first crossing point, which does belong to γ, does not satisfy the appropriate condition with C and may well lie outside Δ; see Figure 5 for an illustration. This bad situation is indeed avoided when $Q_{C,\gamma}$, or rather $\Pi_C(\gamma; a, b, c)$, is defined to be 1 only when the above condition is enforced for the

[13] Recall that the endpoints of $\hat{\gamma}$ are not known at preprocessing time, because γ is not known then, and all we know in advance is their angular placements along c_γ, up to a constant number of possibilities.

first crossing point. We summarize this property in the following lemma.

▶ **Lemma 4.3.** *Let γ, h_Δ, and $\Pi_C(\gamma; a, b, c)$ be as defined above, and suppose that the endpoints of γ lie on opposite sides of h_Δ. Then γ can be replaced by an appropriate oriented canonical semicircle $\hat{\gamma}$ that contains γ, such that γ crosses h_Δ if and only if $\hat{\gamma}$ crosses h_Δ. Moreover, when this is the case, the first intersection point of γ with h is also the first intersection point of $\hat{\gamma}$ with h.*

Lemma 4.3 implies that the replacement of γ by $\hat{\gamma}$ does not affect the output to a query, but it involves ranges (at each level of the structure) that have at most $t = 6$ degrees of freedom, allowing us to get the improved query time $O^*(n^{3/5})$, as in case (i).

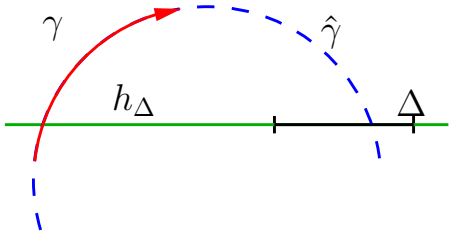

Figure 5 An instance where only the second crossing point of $\hat{\gamma}$ with h_Δ lies in C but this point does not belong to γ.

As all cases have been covered, we have obtained a data structure for circular arc intersection queries with respect to the wide plates of τ that uses $O^*(n^{3/2})$ storage and answers a query in $O^*(n^{3/5})$ time. As discussed in Section 2.1, we can modify the semi-algebraic test described by $\Pi_C(\gamma; a, b, c)$ to include the exact number of intersections (one or two) using additional quantifier elimination.

We have thus shown:

▶ **Proposition 4.4.** *For a cell τ of the polynomial partition, and a set $\mathcal{W}$ of n wide plates at τ, one can construct a data structure of size and expected preprocessing cost $O^*(n^{3/2})$, so that a circular-arc intersection query within τ can be answered in $O^*(n^{3/5})$ time.*

By plugging this improved procedure into the algorithm in Section 4.1, we obtain the improved performance asserted in Theorem 2.5.

The case of general planar arcs.

The machinery presented above also works, with minor modifications, for intersection queries with arbitrary constant-degree planar algebraic arcs. To obtain this extension, we first break the curve c_γ containing the query arc γ, and γ too if needed, at its $O(1)$ inflection points, and also at a constant number of additional points, so that the turning angle of each resulting portion of the curve is sufficiently small, say, at most $10°$. (The overall number of these breakpoints is linear in the degree of γ.) By construction, each of these portions is a convex arc. We then apply the intersection searching algorithm to each portion separately. Since each portion is convex, it can intersect a plane in at most two points. The algorithm presented above for circular arcs, and its analysis, can then be easily extended to handle such general arcs, with suitable and straightforward modifications; the routine details are omitted. (Note in particular that Lemmas 4.2 and 4.3 continue to hold for general convex planar arcs with a small turning angle.)

The performance of the resulting algorithm depends on the number of degrees of freedom needed to specify the curve supporting the query arc, which is[14] $t-2$. That is, as already argued, with $O^*(n^{3/2})$ storage, a query takes

$$O^*\left(n^{\frac{2t-4}{3(t-3)}}/s^{\frac{2}{3(t-3)}}\right) = O^*\left(n^{\frac{2t-7}{3(t-3)}}\right)$$

time. We observe that the top levels of the structure deal with queries whose parametric dimension is only three or two, and only the bottom level uses all t degrees of freedom of the supporting curve. With this procedure for handling wide plates, the rest of the algorithm proceeds as described in Section 2, and yields Theorem 2.5.

5 Arc intersection queries amid triangles with near-linear storage

The main technique in this paper is based on polynomial partitioning, which results in improved query time at the cost of using (substantially) superlinear storage (see, e.g., the survey in [2] for the current state-of-the-art, and recall the discussion in the introduction, including Table 1, regarding our bounds). It is therefore instructive to study the case where only near-linear storage is allowed. Besides being an interesting problem in itself, which does not seem to have been studied earlier, it is also needed as a subroutine in the main algorithm of Section 3, for obtaining some further improvement in the storage bound; see Theorem 2.3.

Since polynomial partitioning inherently leads to superlinear storage in the way we employ it, we will not use it here. As a consequence, we do not have wide and narrow triangles anymore, and thus cannot replace a triangle by its supporting plane, as was done in the main algorithms. Instead, we use the following approach. Let γ be a constant-degree algebraic arc, contained in an algebraic curve $\delta = c_\gamma$, which we assume to be parameterizable, so a point on δ can be represented as $\mathbf{x}_\delta(\xi) = (x_\delta(\xi), y_\delta(\xi), z_\delta(\xi))$, for ξ real. We also assume that δ is taken from some fixed collection Γ of algebraic curves that have t degrees of freedom. Thus each of $x_\delta(\xi)$, $y_\delta(\xi)$, $z_\delta(\xi)$ depends on t additional real variables, and we simply denote the t-tuple of these variables as δ, so each of these expressions is a constant-degree algebraic function of $t+1$ variables.

Let Δ be a triangle, lying on some plane h_Δ. We break the predicate that γ crosses Δ into a conjunction of several sub-predicates, and use a multi-level data structure, each level of which tests for one of those sub-predicates. The purpose of this step is, as above, to reduce the number of parameters of Δ on which each sub-predicate depends.

Specifically, γ intersects Δ if and only if the following conditions hold.

- γ intersects h_Δ.
- One of the intersection points lies, within h_Δ, on the positive side of each of the lines ℓ_1, ℓ_2, ℓ_3 supporting the respective edges e_1, e_2, e_3 of Δ, where the positive side of a line in h_Δ is the halfplane of h_Δ bounded by the line and containing Δ.

For technical reasons, we rewrite these conditions as follows:

There exists $\xi \in \mathbb{R}$ such that

(i) $\mathbf{x}_\delta(\xi) \in h_\Delta$.

(ii) $\xi^-(\gamma) \le \xi \le \xi^+(\gamma)$, where $\xi^-(\gamma)$ and $\xi^+(\gamma)$ are the parameters specifying the endpoints of γ (this sub-condition is vacuous when γ is a full algebraic curve).

[14] More precisely, on the maximum number of degrees of freedom used at each level of the structure.

(iii) $\mathbf{x}_\delta(\xi)$ lies on the positive side of ℓ_1.

(iv) $\mathbf{x}_\delta(\xi)$ lies on the positive side of ℓ_2.

(v) $\mathbf{x}_\delta(\xi)$ lies on the positive side of ℓ_3.

As said, we use a five-level data structure to test for these five sub-conditions. A major technical issue that arises is that δ may intersect h_Δ in several points, and we need to test these conditions for each point separately. For each of these separate tests we need to specify which point (i.e., which value of ξ) is to be used, and all five levels must use the same value. More specifically, condition (i) gives ξ as a root of the algebraic equation $z_\delta(\xi) = ax_\delta(\xi) + by_\delta(\xi) + c$, where (a, b, c) are the coefficients of h_Δ, and we need to specify which root to use in testing for conditions (ii)–(v).

The solution is to use, once again, a CAD, which is constructed for the function $G(a, b, c; \delta; \xi) := z_\delta(\xi) - ax_\delta(\xi) - by_\delta(\xi) - c$, in the $(t + 4)$-dimensional (a, b, c, δ, ξ)-space, where the order of coordinate elimination is first ξ, then δ, and then c, b, a. (Technically, the functions x_δ, y_δ, z_δ need not be polynomials, but algebraic functions, and the CAD construction is defined over polynomials. To address this formally, we construct the CAD over the three polynomials $\tilde{G}(a, b, c; x, y, z) := z - ax - by - c$, $\Phi_1(x, y, z)$, and $\Phi_2(x, y, z)$, where Φ_1 and Φ_2 are polynomials, having together t degrees of freedom, so that δ is the intersection of their zero sets. This calls for a CAD in $t + 6$ dimensions, and does not affect the algorithm in any significant way, but we prefer to describe the construction under the parametric representation of δ, in the interest of clarity. We thus assume, solely for the sake of presentation, that δ has a polynomial parameterization.

Many of the technical details are similar to those used for the CAD introduced earlier, but we spell them out, albeit briefly, because the setup is different here. Denote the CAD as Ξ_{t+4}, denote its projection onto the three-dimensional abc-space as Ξ_3, and its projection onto the $(t+3)$-dimensional $abc\delta$-space as Ξ_{t+3}. For each point (a, b, c) denote by $\Xi_{t+1}(a, b, c)$ the $(t + 1)$-dimensional fiber of Ξ_{t+4} over (a, b, c), and for each point $(a, b, c; \delta)$ denote by $\Xi_1(a, b, c; \delta)$ the one-dimensional fiber of Ξ_{t+4} over $(a, b, c; \delta)$. For each cell C_0 of Ξ_{t+3}, $\Xi_1(a, b, c; \delta)$ has a fixed combinatorial structure for all $(a, b, c; \delta) \in C_0$. This structure is a partition of the ξ-axis into a fixed number of intervals and delimiting points, where each delimiter is given as a fixed constant-degree algebraic function of a, b, c, and δ. Moreover, some delimiters are zeros of $G(a, b, c; \delta; \xi)$ for every $(a, b, c; \delta) \in C_0$, while the others are not zeros, again for every $(a, b, c; \delta) \in C_0$, and the CAD tells us which delimiters are zeros and which are not.

For each triangle Δ, whose supporting plane is parameterized by some triple (a, b, c), we find the cell C' of Ξ_3 that contains (a, b, c). We collect all cells C of Ξ_{t+4} whose abc-projection is C', for which the ξ-component of C is a delimiter that is a zero of G, and associate (i.e., store) Δ with each of these cells. Denote the set of triangles stored at a cell C as $\mathcal{T}_C$.

Answering a query.

Let γ be a query arc and let $\delta \in \Gamma$ be its supporting curve. We iterate over all cells of Ξ_{t+4} whose ξ-component is a zero of G. For each such cell C we want to pick out of $\mathcal{T}_C$ those triangles Δ for which δ appears in the fiber $\Xi_{t+1}(a, b, c)$, where (a, b, c) are the coefficients of h_Δ (more precisely, δ appears in the base of this fiber, obtained by ignoring the ξ-coordinate). Concretely, the tuple $\delta = (\delta_1, \ldots, \delta_t)$ has to satisfy equalities or inequalities

of the form

$$\begin{aligned}
&\delta_1 = f_1(a,b,c) && \text{or} && f_1^-(a,b,c) < \delta_1 < f_1^+(a,b,c)\\
&\delta_2 = f_2(a,b,c;\delta_1) && \text{or} && f_2^-(a,b,c;\delta_1) < \delta_2 < f_2^+(a,b,c;\delta_1)\\
&\qquad\vdots\\
&\delta_t = f_t(a,b,c;\delta_1,\ldots,\delta_{t-1}) && \text{or} && f_t^-(a,b,c;\delta_1,\ldots,\delta_{t-1}) < \delta_t < f_t^+(a,b,c;\delta_1,\ldots,\delta_{t-1}),
\end{aligned} \tag{5}$$

for suitable constant-degree continuous algebraic functions $f_1, f_1^-, f_1^+, \ldots, f_t, f_t^-, f_t^+$.

This set of equalities and inequalities defines a semi-algebraic range $Q_C(\delta)$ in the three-dimensional *abc*-space. Using the semi-algebraic range searching technique of [8, 24], we obtain all the triangles $\Delta \in \mathcal{T}_C$ that satisfy (5) as the union of prestored canonical sets.

For each such canonical set $\mathcal{T}'$, we can express the parameter ξ of the specific intersection point $\mathbf{x}_\delta(\xi)$ of δ and h_Δ, for any $\Delta \in \mathcal{T}'$, as an explicit algebraic function of the coefficients (a,b,c) of h_Δ and of δ. We denote this function as $\varphi_C(a,b,c;\delta)$.

This completes the description of the first (and most involved) level of our structure. We apply this procedure to each relevant cell C of Ξ_{t+4} separately. Let C be a fixed such cell. We pass to the next level the canonical sets $\mathcal{T}'$ that comprise $\mathcal{T}_C$, as well as the specific expression $\xi = \varphi_C(a,b,c;\delta)$, which will be used in the four other levels too.

Once we know how to express ξ algebraically, the other levels are easier to implement. The second level, which tests for condition (ii), is implemented as a semi-algebraic range searching query, in the *abc*-space, with the range

$$\{(a,b,c) \mid \xi^-(\gamma) \le \varphi_C(a,b,c;\delta) \le \xi^+(\gamma)\}.$$

We then build, for each canonical set of triangles of the second-level structure, three subsequent levels, each testing one of the conditions (iii)–(v). Specifically, let $\mathcal{T}_0$ be such a canonical set of triangles. Let Δ be a triangle in $\mathcal{T}_0$, whose plane h_Δ is parameterized by (a,b,c), let e_1 denote the first designated edge of Δ, and let ℓ_1 denote its supporting line. We need two additional parameters to specify ℓ_1 within h_Δ; denote them as μ, ν.

Let $\Pi_C^{(1)}(\gamma;a,b,c,\mu,\nu)$ denote the predicate that is equal to 1 when $\mathbf{x}_\delta(\xi)$ lies on the positive side of ℓ_1, and put

$$Q_C^{(1)}(\gamma) := \{(a,b,c,\mu,\nu) \mid \Pi_C^{(1)}(\gamma;a,b,c,\mu,\nu) = 1\}.$$

(The subscript C reminds us that $\Pi_C^{(1)}$ and $Q_C^{(1)}$ depend on the particular cell C of Ξ_{t+4} that we are processing.) $Q_C^{(1)}(\gamma)$ is a semi-algebraic region of constant complexity in a five-dimensional parametric space, each point of which represents a plane h_Δ and a line ℓ_1 within that plane. We query with $Q_C^{(1)}(\gamma)$ in the set H_1^* consisting of all 5-tuples (a,b,c,μ,ν) that correspond to the triangles of $\mathcal{T}_0$, each with some designated edge e_1 that spans the line given by (μ,ν), together with its positive side that contains the corresponding triangle.

The two other levels of the structure are essentially identical to this level, except that they handle the two other respective edges of each of the corresponding triangles.

By repeating the whole procedure for each relevant cell C of Ξ_{t+4}, we ensure that all possible intersection points of γ with any triangle Δ will be detected. That is, the final output of the query is nonempty, for at least one cell C (and corresponding multi-level structure), if and only if γ intersects a triangle of $\mathcal{T}$.

The first two levels of the structure are semi-algebraic range searching structures in three dimensions, and the last three levels are semi-algebraic range searching structures in five dimensions. It follows from standard arguments in range searching (see [2] and [8, 24]) that

one can implement the overall structure so that it uses $O^*(n)$ storage and preprocessing time, and answers an algebraic arc intersection query in $O^*(n^{4/5})$ time (the query cost at the last three levels dominates the cost for the entire structure). That is, we have:

▶ **Theorem 5.1.** *A set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$ can be processed, in expected $O^*(n)$ time, into a data structure of size $O^*(n)$ so that an arc-intersection query amid the triangles of $\mathcal{T}$ can be answered in $O^*(n^{4/5})$ time.*

6 The case of plate queries and input lines

We now move to the second part of the paper, in which the nature of the input and query objects is interchanged: the input now consists of a collection of arcs, and we query with plates. The goal is the same: detect, count, or report intersections between the query object and the input objects (shooting has no reasonable parallel in this reverse setup).

We begin by focusing on the special case where the input consists of lines and the queries are plates. Later we will show how to extend the technique to handle the more general context of input arcs and of query plates.

Let L be a set of n lines in $\mathbb{R}^3$. For concreteness, consider the intersection detection task. We construct, in linear time, a partitioning polynomial F of degree $O(D)$, for some sufficiently large constant parameter D, so that each cell of the partition that the zero set $Z(F)$ of F induces, namely each connected component of $\mathbb{R}^3 \setminus Z(F)$, is crossed by at most n/D^2 lines of L; see [3, 20], as well as [10]. We note that the zero set may contain any number of input lines; these lines will be handled separately.

For a cell τ of the partition and a query plate Δ, we call Δ *narrow* (resp., *wide*) at τ if an edge of Δ crosses τ (resp., Δ crosses τ but none of its edges does).

Each line $\ell \in L$ not fully contained in $Z(F)$ crosses $Z(F)$ at $O(D)$ points, which break ℓ into $O(D)$ segments (and rays), so that the relative interior of each segment e is fully contained in a single cell τ of the partition, and the endpoints of e lie on the boundary of τ. A cell τ may contain several such segments of the same line ℓ. We denote by L_τ the set of input lines that cross τ, and recall that[15] $|L_\tau| = O(n/D^2)$.

Let Δ be a query plate. There are $O(D)$ partition cells at which Δ is narrow, and $O(D^2)$ cells at which Δ is wide. We process the query by a direct nonrecursive procedure at cells τ where Δ is wide, and handle the cells at which Δ is narrow recursively.

Handling cells where Δ is wide.

Let τ be such a cell, and let L_τ be as defined above.

Solely for the purpose of this subprocedure, we partition τ further into subcells with a simple structure and a simple topology. Concretely, we construct the CAD of $Z(F)$ in $\mathbb{R}^3$ (see again [13, 15, 29] and Section 3.1 for details). Each full-dimensional cell π of the CAD (as well as some lower-dimensional cells[16]) is fully contained in some cell τ of the partition.

[15] The number of segments within τ into which these lines are broken may be larger, but this will not affect our analysis.

[16] The vertical walls of π are easier to handle, and the ceiling and floor of π are contained in $Z(F)$ and are handled separately—see below.

As in Section 3, π is a prism-like cell (we refer to it simply as a prism) of the form

$$\begin{aligned} \alpha_1 < x < \alpha_2 \\ f_1(x) < y < f_2(x) \\ g_1(x,y) < z < g_2(x,y), \end{aligned}$$

where α_1, α_2 are reals, and f_1, f_2, g_1, g_2 are continuous algebraic functions of constant degree (which depends on D); some of α_1, α_2 and these functions might be $\pm\infty$. In general, π has six two-dimensional faces, contained respectively in the algebraic surfaces $x = \alpha_1$, $x = \alpha_2$, $y = f_1(x)$, $y = f_2(x)$, $z = g_1(x,y)$ and $z = g_2(x,y)$. Each face is simply connected, monotone (with respect to a suitable coordinate plane), and of constant complexity (again, which depends on D); π has fewer faces when some of α_1, α_2, f_1, f_2, g_1, g_2 are $\pm\infty$.

For each partition cell τ, each segment of L_τ is split into several subsegments, each of which is fully contained in some prism π of the CAD, and its endpoints lie on the boundary of π. Let L_π denote the set of these subsegments in a prism π.

Let π be a CAD sub-prism of τ. Since Δ is wide at τ, it is also wide at π. Let $\pi(\Delta)$ be the arrangement of $\pi \cup \{\Delta\}$, i.e., the decomposition of π by Δ. (It is important that we decompose π by Δ and not by the plane supporting Δ.) The complexity of this arrangement is a constant that depends on D.

▶ **Lemma 6.1.** *A segment e of L_π intersects Δ if and only if its endpoints lie on the boundary of different three-dimensional cells of $\pi(\Delta)$.*

Proof. Since Δ is wide at π, its intersection $\Delta \cap \pi$ consists of one or several regions, all fully contained in the relative interior of Δ. Consequently, each of these pieces fully slices π. Informally, the purpose of the lemma is to argue that a point on one side of such a slice cannot reach a point on the other side, within π, without crossing the slice.

The segment e lies inside π, so if the endpoints of e lie in different cells of $\pi(\Delta)$, then e has to intersect Δ to go from one cell to another; it has to be through Δ because the relative interior of e does not meet the boundary of π. On the other hand, assume that the two endpoints lie in the same three-dimensional cell ψ but e intersects Δ. At each such intersection, e has to move from one three-dimensional cell of $\pi(\Delta)$ to another cell. See Aronov et al. [12] for a (nontrivial) proof of this seemingly obvious property. Informally, it holds because each cell of $\pi(\Delta)$ is topologically a ball. We note that (i) this property also holds, as shown in [12], for arbitrary connected arcs e, and (ii) this property may fail for more general, non-CAD cells π (such as, e.g., a torus). See Figure 6 for an illustration for the case of a CAD cell. It follows that e leaves ψ when it crosses Δ, and has to return to ψ, necessarily at a second such crossing. Thus e intersects Δ (at least) twice, which is impossible. (Note that this last part of the argument fails when e is not straight, as illustrated in Figure 7.) ◀

Data structure.

We build a family of $O(1)$ two-level two-dimensional semi-algebraic range searching data structures as follows. For each two-dimensional face φ_0 of π, we collect the subset E_{φ_0} of endpoints of segments in L_π that lie in φ_0. We construct a separate structure for each pair φ_0, φ_0' of (not necessarily distinct) faces of π. The first level of the structure associated with the pair is constructed over the set E_{φ_0}. We use the fact that φ_0 is monotone with respect to one of the coordinate planes, so we simply project it, and the set E_{φ_0}, onto that plane, and construct a two-dimensional semi-algebraic range searching structure for the projected

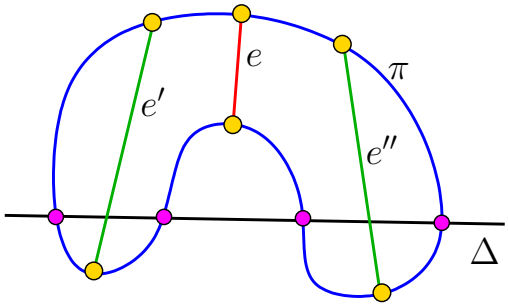

Figure 6 A two-dimensional rendering of the scenario analyzed in Lemma 6.1. The red segment e has endpoints in the same three-dimensional cell of $\pi(\Delta)$ and does not cross Δ, whereas each of the green segments e', e'' has endpoints in different cells of $\pi(\Delta)$ and crosses Δ.

set; the ranges with which we will be searching will be of constant complexity, and will be specified shortly. The construction uses the algorithm of Matoušek and Patáková [24] (see also [8]), which, for N input points, uses $O^*(N)$ storage and preprocessing, and answers a range searching query in $O^*(N^{1/2})$ time.

For each canonical set E produced by the structure, we take the set $E'_{\varphi'_0}$ that consists of the second endpoints of those segments whose first endpoint is in E, for which that second endpoint lies in φ'_0. We construct a secondary range searching structure for $E'_{\varphi'_0}$, projected onto the coordinate plane over which φ'_0 is monotone, using the same machinery of [24], as above.

Standard analysis of multi-level data structures (see, e.g., [2]) implies that the overall structure uses $O^*(n)$ storage and preprocessing, and answers a query in $O^*(n^{1/2})$ time. This completes the description of the data structure.

Query procedure.

To answer a query for a wide plate Δ at a CAD subcell π, we construct the corresponding decomposition $\pi(\Delta)$ of π. We iterate over all pairs of two-dimensional faces of π. For each such pair φ_0, φ'_0, we iterate over all pairs of subfaces $\varphi \subseteq \varphi_0$, $\varphi' \subseteq \varphi'_0$ in $\pi(\Delta)$, so that φ and φ' lie on the boundaries of different three-dimensional cells of $\pi(\Delta)$. For each such pair φ, φ', we query the data structure associated with (φ_0, φ'_0) with the pair of semi-algebraic ranges φ, φ' (projected onto the corresponding respective coordinate planes), where the query at the first level is with the projection of φ, and the query at the second level is with the projection of φ'. Clearly these are semi-algebraic ranges of constant complexity, as promised. Each segment in the output of any of these sub-queries crosses Δ (within π), and every segment that crosses Δ within π appears in such an output, exactly once over all sub-queries.

As already noted, the cost of a query is $O^*(n^{1/2})$.

Handling input lines and queries on the zero set.

If a query plate is contained in $Z(F)$, then it must be contained in one of the at most $O(D)$ planar components of $Z(F)$. We simply take each such component h, collect the set E_h of the intersection points $\ell \cap h$, over the lines $\ell \in L$ that are not contained in $Z(F)$, and preprocess E_h for planar range searching, where the query ranges are plates. Since there are only $O(1)$ components h, the overall storage and preprocessing cost of this step is $O^*(n)$, and a query takes $O^*(n^{1/2})$ time.

To handle input lines that are fully contained in $Z(F)$, we process each irreducible component of $Z(F)$ separately. Computing the irreducible components of $Z(F)$ can be done in constant time, since D is a constant. We can also retrieve these components from the CAD of $Z(F)$. We omit the further technical details of this step, some of which can be found in [13, 16]. Any such component V that is not ruled by lines can contain at most $O(D^2)$ lines (this is the Cayley-Salmon theorem; see, e.g., [21], and see [22] for a review of ruled surfaces), so only $O(1)$ lines of L can lie in V, and we simply store them with V and process them by brute force, in constant time, during a query. For each component V, we check, for each line $\ell \in L$, whether it is contained in V (say, using Bézout's theorem). If the number of such lines does not exceed the Cayley-Salmon threshold, then this set of lines is processed as above by brute force. Otherwise, we conclude that V must be a ruled surface and proceed as described below.

For planar components V, a typical query plate crosses V in a constant number of segments, and our problem then reduces to a two-dimensional intersection detection problem, with the lines contained in V as input and with segments as queries. The degenerate cases where the query plate is contained in V are also easy to handle, in a similar manner, since this problem can be formulated as a planar semi-algebraic range searching.

Consider then the case where the component V is singly or doubly ruled by lines. Suppose for specificity that V is singly ruled; doubly ruled components can be handled by a variant of this argument. As observed in [21], for example, with the exception of at most two lines, all lines that are contained in V belong to the single ruling family, and these lines are parameterized by a single real parameter t. We form the set of the values of t that correspond to the input lines, and preprocess them into a trivial one-dimensional range searching structure, over the parameter t, which uses $O^*(n)$ storage and $O(\log n)$ query time. We map a query plate Δ into a range that is a union of a constant number of intervals along the t-axis, representing the values of t for which the corresponding line in the ruling crosses Δ. We then query our structure with each of these intervals.

Putting everything together, we obtain a data structure that uses $O^*(n)$ storage, and answers an intersection detection query in $O^*(n^{1/2})$ time.

Analysis.

To recap, we have shown above how to handle the nonrecursive processing in cells where the query plate is wide, as well as the nonrecursive processing of the zero set. We recall that we handle cells τ at which the query Δ is narrow recursively. Putting all these ingredients together, we obtain that the overall storage $S(n)$ of the structure for n lines obeys the recurrence

$$S(n) \leq c_1 D^3 \left(S(n/D^2) + S_W(n/D^2) \right) + O^*(n),$$

where $c_1 > 0$ is an appropriate absolute constant, $S_W(n) = O^*(n)$ is the storage used by the structure for querying on n lines with a wide plate at a cell τ, and the third term is the storage used for handling the zero set. We thus obtain $S(n) = O^*(n^{3/2})$.

The cost $Q(n)$ of a query obeys the recurrence

$$Q(n) \leq c_2 D Q(n/D^2) + c_3 D^2 Q_W(n/D^2) + O^*(n^{1/2}),$$

where $c_2, c_3 > 0$ are appropriate absolute constants, $Q_W(n) = O^*(n^{1/2})$ is the cost of the query on n lines with a wide plate at a cell, and the third term is for querying within the zero set. Substituting this into the above recurrence, we obtain $Q(n) = O^*(n^{1/2})$.

In summary, we have shown:

▶ **Theorem 6.2.** *A set L of n lines in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that, for any query plate Δ, we can perform an intersection query with Δ in L in $O^*(n^{1/2})$ time.*

For counting queries, we add up the sizes of the canonical sets produced at the bottom level of the structure, exploiting the fact that each intersection with the query plate Δ is encountered exactly once in the entire process. For the reporting version, we output the elements of the above canonical sets.

6.1 Extensions

The technique presented above can be easily extended to the following setups.

Queries on a set of segments.

Assume that the input objects are n straight segments, instead of lines. An input segment e is said to be *short* (resp., *long*) at a cell τ if τ contains an endpoint of e (resp., crossed by e but does not contain an endpoint); e is long at all the cells that it crosses, except for at most two. A variant of the construction of Guth [20] and of Guth and Katz [21] lets us construct a partitioning polynomial F of degree $O(D)$ so that each cell contains at most n/D^3 short segments, and is crossed by at most n/D^2 segments.

Handling long segments at a cell is done exactly as before. To handle short segments, for each cell τ, we prepare a two-level halfspace range searching data structure that allows us, for a query plate Δ that crosses τ, to obtain the set of all the short segments in τ that cross the plane h_Δ supporting Δ, as the union of a few canonical sets. (Clearly, Δ cannot cross any segment that is not crossed by h_Δ.) This can be done so that, with $s = O^*(n^{3/2})$ storage, a query costs $O^*(n/s^{1/3}) = O^*(n^{1/2})$ time (see, e.g., [2]). For each canonical set of the structure, we replace each original segment e by its extension e', which is connected component of $\tau \cap \ell$ containing e, where ℓ is the line supporting e. Since a query plate has to process only sets of segments that h_Δ crosses, this replacement does not cause any new (false positive) intersections to arise.

We now process the extended segments as before. By standard analysis of multi-level data structures [2], adding the above extra levels to the structure (for handling cells at which the query plate is wide) does not affect the performance bounds in any significant manner: the storage and preprocessing for the structure remain $O^*(n^{3/2})$ and the query still takes $O^*(n^{1/2})$ time. Handling the zero set is done in a similar manner as in the case of input lines, after a similar construction of canonical sets of segments that intersect h_Δ, and replacing these segments by the lines containing them. The recurrence for handling cells at which the query plate is narrow retains the same performance bounds as before. We thus obtain:

▶ **Theorem 6.3.** *A set L of n line segments in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that, for any query plate Δ we can perform an intersection query with Δ in L in $O^*(n^{1/2})$ time.*

Queries on a set of arcs.

We use a suitable extension of the above algorithm. We face however the following major new issue. When the input objects were straight segments, Lemma 6.1 provided us with a necessary and sufficient condition for an input segment to intersect the query plate Δ, namely, that the endpoints of the segment lie in different cells of the arrangement $\pi(\Delta)$.

However, when the input objects are curved arcs, this criterion remains sufficient but in general not necessary; see Figure 7 for an illustration.

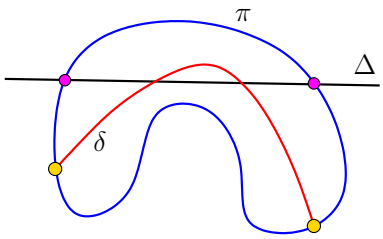

Figure 7 Lemma 6.1 may fail when the input consists of curved arcs. The arc δ has both endpoints in the same cell of $\pi(\Delta)$ but it still intersects Δ.

We therefore need to modify the algorithm for cells at which the query plate Δ is wide, which we do as follows, borrowing ideas and tools from the earlier part of the paper.

First, there is no need to construct the CAD of $Z(F)$. Instead, let τ be a cell of the polynomial partition at which the query plate Δ is wide. $\Delta \cap \tau$ is a collection of connected semi-algebraic regions, all lying fully in the relative interior of Δ, and we want to test each of them for intersection with the input arcs.

We use the labeling scheme of Section 3, which constructs a CAD Ξ_5 of the polynomial $F(x, y, ax+by+c)$, in the five-dimensional $abcxy$-space $\mathbb{E}$, where a, b, c are the coefficients of the plane h_Δ (that is, $h^*_\Delta = (a, b, c)$). The two-dimensional fiber $\Xi^2(a, b, c)$ over h^*_Δ partitions the xy-projection of h_Δ into pseudo-trapezoids, and each of the regions of $\Delta \cap \tau$ is the disjoint union of some of these trapezoids. Each trapezoid has a discrete label, which is its semi-algebraic representation and which has constant complexity.

For each cell C of Ξ_5, and its three-dimensional abc-projection C_0, there is a fixed semi-algebraic representation of the trapezoid ξ on Δ, for which $\{h^*_\Delta\} \times \xi^\downarrow \subseteq C$ (so $h^*_\Delta \in C_0$). We prepare a semi-algebraic range searching data structure where the objects are the boundary curves of the input regions, and the query range is the trapezoid ξ that corresponds to the cell C. We use a primal-dual approach, where in the primal setup the input curves are represented as points in $\mathbb{R}^t$, where t is the parametric dimension of these arcs (including the two parameters for specifying the endpoints), and we query with the semi-algebraic range Q_{C,h^*_Δ}, consisting of all points in $\mathbb{R}^t$ representing arcs δ that cross the trapezoid ξ as defined above. The dual setup is cast in $\mathbb{R}^3$, and here too there is a separate data structure for every possible CAD cell C. The query region Δ is represented as a point $h^*_\Delta = (a, b, c)$, as above, and each boundary arc δ is represented as a semi-algebraic region $Q^*_{C,\delta}$, consisting of all points h^*_Δ such that δ intersects the trapezoid corresponding to C on h_Δ.

Standard analysis of the primal-dual approach (see, e.g., [2, 5]) shows that, with $s = O^*(n^{3/2})$ storage, a query takes $O^*(n^{\frac{3(t-1)}{4t}})$ time. Note that this bound is different than the one obtained earlier, for querying with an arc amid plates, since the primal-dual scheme for the current scenario consists of point-enclosure queries in $\mathbb{R}^3$ and range-searching queries in $\mathbb{R}^t$. For example, when the input arcs are circles and the query plates are disks, we have $t = 6$ and the cost of an intersection query is $O^*(n^{5/8})$.

So far we have handled partition cells at which the query plate is wide. Handling cells at which the query plate is narrow is done similarly as in the case of lines described earlier.

Handling the zero set. Consider next the task of handling input arcs that lie in $Z(F)$. Using arguments as above, we may assume, without loss of generality, that $Z(F)$ is irreducible and xy-monotone. Project $Z(F)$ and the input arcs that it contains on the xy-plane. We

get a collection Γ of (at most) n arcs of constant complexity, with the same parametric dimension t as the original arcs. The query plate Δ crosses $Z(F)$ in $O(1)$ arcs (where the constant of proportionality depends on the parametric dimension of the bounding arcs of Δ). Let δ be such an arc, and we denote by δ^* its xy-projection. The curve containing δ^* is an algebraic curve of constant complexity and of parametric dimension 3 (inherited from the plane h_Δ). The parametric dimension of δ^* itself is 5, including its endpoints, but we will get rid of the effect of these two extra parameters.

Construct a partitioning polynomial G, of degree $O(D_1)$, for $D_1 \gg D$, which partitions the plane into $O(D_1^2)$ cells, each crossed by at most n/D_1 arcs of Γ. The arc δ^* crosses $O(D_1)$ cells, and it is *long* (i.e., the cell does not contain any of its endpoints) in all of them except for at most two.

For each cell τ, we let Γ_τ denote the set of arcs of Γ that cross τ, where each arc is clipped to within τ (so the number of connected components of the clipped arcs may be larger than n/D_1, but this will not affect the analysis). Note that each arc in Γ_τ still has t degrees of freedom. We preprocess Γ_τ into a data structure for intersection queries with arcs that have three degrees of freedom. Specifically, each query arc is a clipped connected portion of δ^* within a cell τ at which δ^* is long. Some care has to be exercised here, to ensure that each query subarc still has only three degrees of freedom, but we omit here the details, which are similar to the CAD labeling scheme used earlier.

To answer a query with an arc δ^*, we first query with each subarc of δ^* at the cells where δ^* is long, using the procedure sketched above, and then recurse at the at most two cells where δ^* is short. We repeat this step for each of the $O(1)$ subarcs δ^* whose lifted arc δ appears on the intersection of the query plate Δ and $Z(F)$. The cost is easily seen to be asymptotically the same as that of the above procedure. Using a primal-dual approach, as before, the previous analysis yields the summary result:

▶ **Theorem 6.4.** *A set Γ of n constant-degree algebraic arcs in $\mathbb{R}^3$, with parametric dimension t, can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that, for any query plate Δ, we can perform an intersection query with Δ in Γ in $O^*(n^{\frac{3(t-1)}{4t}})$ time.*

7 The case where both input and queries are triangles or plates

Finally, we move to the third part of the paper, in which both input and query objects are plates. We first focus on the case where both input and query objects are triangles, and later comment on the relatively easy extension to the general case.

Consider then the case where the input is a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$ and the query is a triangle, and the goal is to detect, count, or report intersections between the query triangle and the input triangles. Consider first the detection version.

The solution is quite simple, and is based on a combination of the analysis in [19] and Section 6 of this paper. Note that if two triangles Δ, Δ' intersect (in general position) then their intersection is a line segment $e = pq$, where each of the endpoints p, q is an intersection point of an edge of one triangle with the other triangle.

It follows that if a query triangle Δ_q intersects some input triangle Δ then either

(i) an edge of Δ_q crosses Δ, or
(ii) Δ_q crosses an edge of Δ.

(Any combination of (i) and (ii) can occur at the two respective endpoints p, q.)

To detect intersections of type (i), we apply the algorithm of [19], which uses $O^*(n^{3/2})$ storage and answers a query in $O^*(n^{1/2})$ time. To detect intersections of type (ii), we use the algorithm of Section 6, which also uses $O^*(n^{3/2})$ storage and answers a query in $O^*(n^{1/2})$ time.

The reporting version is an easy extension of the detection procedure just described. We note that, in general position, each intersection of a query triangle with an input triangle is detected exactly twice, once for each endpoint of the intersection segment, and each of these detections can be either of type (i) or of type (ii). Omitting the further straightforward details, we therefore conclude:

▶ **Theorem 7.1.** *A set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that, for any query triangle Δ, we can detect whether Δ intersects any triangle of $\mathcal{T}$ in time $O^*(n^{1/2})$. We can report all k such intersected triangles in time $O^*(n^{1/2} + k \log k)$.*

The counting version is also easy, observing that, in general position, each pair of intersecting triangles (the query triangle and an input triangle) is encountered by the above procedure exactly twice, once for each endpoint of the intersection segment. However, the algorithm of [19] is unable to count intersections, so we need to use the alternative technique in Theorem 2.4, which, still with $O^*(n^{3/2})$ storage, answers a query in $O^*(n^{5/9})$ time. We thus obtain (by the above observation, we need to divide the resulting count by 2):

▶ **Theorem 7.2.** *A set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that, for any query triangle Δ, we can count the number of input triangles that Δ intersects in time $O^*(n^{5/9})$.*

7.1 Extensions

Consider next the general setup, where both the input and query objects are plates.

We begin by considering the detection version. Many aspects of the algorithm of Theorem 7.1 are fairly easy to generalize. Here the intersection of a query plate Δ_q with an input plate Δ is the union of $O(1)$ pairwise disjoint segments, all lying on the intersection line of the two supporting planes, and each endpoint of each of these segments is an intersection of either (i) Δ with a boundary arc of Δ_q, or (ii) a boundary arc of Δ with Δ_q. (There is only one intersection segment when both plates are convex.)

Detecting intersections of type (i).

We use the techniques of the first part of the paper. Among the variety of solutions that it provides, we apply Theorem 2.5, observing that the query arcs are planar. We therefore obtain a data structure of size $O^*(n^{3/2})$, which is constructed in expected time $O^*(n^{3/2})$, so that an arc intersection query, with a constant-degree algebraic arc of parametric dimension $t \geq 3$, can be answered in $O^*\left(n^{\frac{2t-7}{3(t-3)}}\right)$ time. We comment that if the query plate is bounded by a single endpoint-free curve, the query time bound is $O^*\left(n^{\frac{2t-3}{3(t-1)}}\right)$ (where t is now the parametric dimension of the bounding curve).

Detecting intersections of type (ii).

For this case we use the technique presented in Section 6.1. That is, we apply Theorem 6.4 on the boundary arcs of the input plates, and obtain a data structure of size $O^*(n^{3/2})$ (constructed

in expected time $O^*(n^{3/2})$), which supports plate-intersection queries in $O^*\left(n^{\frac{3(t-1)}{4t}}\right)$ time, where $t \geq 3$ is the parametric dimension of the input arcs.

Combining the bound in Theorem 6.4 with the one in Theorem 2.4, we obtain:

▶ **Theorem 7.3.** *A set $\mathcal{T}$ of n plates in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^{3/2})$, in expected time $O^*(n^{3/2})$, so that an intersection detection query with a plate can be answered in time*

$$\max\left\{O^*(n^{\frac{2t_Q-7}{3(t_Q-3)}}),\ O^*(n^{\frac{3(t_O-1)}{4t_O}})\right\},$$

where t_Q is the parametric dimension of the boundary arcs of the query region, and t_O is the parametric dimension of the boundary arcs of the input regions.

8 Segment-intersection queries for spherical caps

In the final result of this study, we consider the case where the queries are segments and the input objects are n spherical caps (where a cap is the portion of a sphere cut off by a halfspace). This is different from the previous cases in that the input objects are not flat. We use this case as an example of how far the techniques of this paper can be extended to non-flat inputs.

We go over the steps of the algorithms presented in Sections 2–5 and discuss the modifications that they require for the new problem.

(i) Constructing the CAD.

In the spirit of the technique in Sections 2–5, we replace the caps by their containing spheres, and construct the CAD for the pair of polynomials

$$F(x,y,z) \quad \text{and} \quad (x-a)^2+(y-b)^2+(z-c)^2-r^2,$$

where (a,b,c) is the center of a sphere and r is its radius. The CAD is thus constructed in $\mathbb{R}^7$, with coordinates a,b,c,r,x,y,z, in the reverse elimination order (starting with z).

The 'base' (projected) CAD is now in the four-dimensional $abcr$-space, each point of which represents a sphere. The xyz-fiber over each point (a,b,c,r), restricted to the sphere S, represented by (a,b,c,r), is a refinement of the partition of S induced by $Z(F)$. As before, each (now curvy) pseudo-trapezoidal cell σ of the partition on S is labeled by the cell of the CAD that contains it (in the same sense as before), and we use this label to represent σ by an explicit semi-algebraic expression that will be used in the subsequent range searching step.

Note that, contrary to the setup in Sections 2–5, where the input consisted of plates and it was sufficient to consider the xy-projection of the CAD cells induced on the planes $h(a,b,c)$, in the case of spherical caps we also need to consider the z-coordinate of the pseudo-trapezoidal cells.

(ii) The range searching mechanism.

For each cell C of the CAD and for the corresponding cell $\tau = \tau_C$ of the polynomial partition, we take the set $\mathcal{S}_C$ of all caps stored at C and wide at τ, and run a semi-algebraic range searching procedure that finds, for a query segment s that is contained in τ, the set of all caps $\kappa \in \mathcal{S}_C$ that are crossed by s at a point (x,y,z) that belongs to C; that is, $(a,b,c,r,x,y,z) \in C$, where (a,b,c,r) are the parameters of the sphere containing κ.

(iii) The overall performance.

This range searching step has six degrees of freedom for the queries (segments in $\mathbb{R}^3$) and four for the input objects (spheres). As we show, this implies that we can either (i) perform a query, with $O^*(n^{5/4})$ storage, in $O^*(n^{3/4})$ time, in full analogy to the bounds $O^*(n^{4/3})$ for storage and $O^*(n^{2/3})$ for query time in our basic algorithm (see Sections 2 and 3), or (ii) perform a query, with $O^*(n^{3/2})$ storage, in $O^*(n^{27/40})$ time, similar to the previous bounds $O^*(n^{3/2})$ for storage and $O^*(n^{\frac{2t-3}{3(t-1)}})$ for query time (see Theorem 2.4).

To obtain these results, we first observe that since a sphere is represented by a point in a four-dimensional space, the bound $O^*(n^{2/3})$, which is based on the primal-only technique of [24], is now replaced by $O^*(n^{3/4})$. Thus, to obtain the first result, we choose the threshold $n_0 = n^{1/2}$ in the construction of the partition tree Ψ. The solutions of recurrences (1) and (2) now become $O^*(n^{5/4})$ for storage (and expected preprocessing time) and $O^*(n^{3/4})$ for query time.

To obtain the second result, we set $n_0 = O(1)$ in the construction of Ψ, and apply a primal-dual range searching algorithm for the wide caps. The primal part of the structure is constructed for points (representing spheres) in a four-dimensional space, and the dual part of the structure is constructed for regions in a six-dimensional space (corresponding to the input segments). Aiming for $s = O^*(n^{3/2})$ (which is also the size of Ψ), we stop the construction of the primal part as soon as the size of the subproblems becomes at most $x = n^{1/10}$ (which, as is easily verified, is the correct threshold for keeping the storage $O^*(n^{3/2})$), which yields an improved query time of $O^*(n^{27/40})$.

▶ **Theorem 8.1.** *We can perform segment-intersection queries amid n spherical caps in $\mathbb{R}^3$ by a structure that uses $O^*(n^{5/4})$ storage and expected preprocessing, and answers a query in $O^*(n^{3/4})$ time, or, alternatively, by a structure that uses $O^*(n^{3/2})$ storage and expected preprocessing, and answers a query in $O^*(n^{27/40})$ time.*

Remarks. **(1)** Our main goal in studying segment intersection queries amid spherical caps was to demonstrate the versatility of our technique, and we did not make an effort to optimize the bounds. In particular, we did not try to devise a data structure of near-linear size and sublinear query time, as we did in Section 5. Such a data structure would allow us to improve the storage bound in the first result. We also did not attempt to reduce the number of degrees of freedom of the query objects (which are segments), by replacing a query segment with its supporting line, as we did for arc intersection amid plates.

(2) We only considered the case where the query object is a line segment. However, the technique easily extends to queries with arcs, similar to the extensions noted earlier for input plates (except for the algorithm that reduces the effective parametric dimension by 2, by getting rid of the effect of the endpoints of the query arc, as this reduction relies on the assumption that the input objects are flat). The performance deteriorates with the parametric dimension of the query arcs, and the concrete bounds (which can easily be worked out, although we skip this step in this brief discussion) result from using a primal-dual range searching technique, where in the primal we have a semi-algebraic range searching problem in $\mathbb{R}^4$ (the parametric dimension of the spheres containing the input caps), and in the dual we have a point enclosure problem in $\mathbb{R}^t$, where t is the parametric dimension of the query arcs.

9 Discussion

The analysis in this paper raises many open issues that would be interesting to pursue, and we mention a few here.

Detection vs. counting. It is constructive to compare our results with the previous work of Ezra and Sharir [19], for the special case of intersection detection where the queries are line segments (or lines or rays). In this case, with $O^*(n^{3/2})$ storage, the algorithm of [19] yields query time $O^*(n^{1/2})$, which is better than what we get, namely $O^*(n^{5/9})$. Nevertheless, there are two major advantages of our algorithm. First, it completely bypasses the second (and rather involved) recursive mechanism that is applied in [19] to the wide input triangles at a partition cell τ, making this aspect of the algorithm much simpler conceptually. Moreover, the second recursion in [19] has the disadvantage that it might process a wide triangle at τ many times, and therefore cannot handle intersection-counting queries, where we want to report either the number of triangles that the query segment hits, or the number of intersection points of the query with the triangles (these two quantities are the same for querying with straight segments but may differ for querying with algebraic arcs). Since we do not need this recursion in our approach, it can handle intersection-counting queries, but still only queries that seek the number of *intersection points* of the query arc with the input triangles.[17] It therefore remains an open challenge to perform intersection-counting queries for, say, circular arcs, where the goal is to report *the number of triangles* (rather than the number of the intersection points) that a query arc crosses.

Detecting intersections of anything with anything. The technique presented in this paper is sufficiently versatile to support many other types of intersection queries, where the input objects are non-flat surface patches (as in the case of spherical caps considered earlier), or when the query objects are non-flat patches. We give a high-level view of possible such extensions, based on the approach used so far. This view merely sketches the approach that has to be taken, but the actual implementation, in many cases, remains an open challenge.

In general, the CAD construction facilitates the passage from an input consisting of surface patches (such as triangles or spherical caps) to the full surfaces containing them (such as planes or spheres). This leads to a reduction (often significant) in the number of degrees of freedom of the input objects, which in turn leads to improved performance bounds for the resulting algorithm. However, this reduction becomes possible only if we use polynomial partitioning. The partitioning allows us to focus on wide input objects within each partition cell τ, namely objects that cross τ but whose relative boundary does not cross τ. The CAD technique, appropriately modified, can give us a succinct semi-algebraic representation of each trapezoidal piece of each connected portion of a wide input object within τ, which can then be used to facilitate the subsequent range searching step.

A similar reduction in the number of degrees of freedom is also desirable for the query object, but it is far from obvious and has to be worked out, especially when the objects are non-planar arcs or patches. For arcs, as in the cases studied in the paper, it typically amounts to eliminating the effect on the bounds of the endpoints of the query arc, effectively replacing it by its full containing curve.

The query time bound of the resulting algorithm would depend on the number of degrees of freedom of (a) the surfaces containing the input objects, and (b) the curve of surface containing the query object, and also on the amount of storage we allocate to the structure.

[17] We do not know how to count the actual number of intersected input regions, because the intersection points of an input region with the query arc may be detected (counted) at different instances of the algorithm, and we do not see how to avoid a multiple count of that region in such cases.

References

1 P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM J. Comput.*, 21(3):540–570, 1992.

2 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

3 P. K. Agarwal, B. Aronov, E. Ezra, and J. Zahl. An efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50:760–787, 2021. Also in *Proc. Sympos. on Computational Geometry (SoCG)*, 2019, 5:1–5:14. Also in arXiv:1812.10269.

4 P. K. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in 3D. In *Proc. 11th Annu. Sympos. on Computational Geometry*, pages 267–276, 1995.

5 P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, volume 223 of *Contemp. Math.*, pages 1–56. AMS Press, Providence, RI, 1999.

6 P. K. Agarwal and J. Matousek. On range searching with semialgebraic sets. *Discret. Comput. Geom.*, 11:393–418, 1994.

7 P. K. Agarwal and J. Matoušek. Ray shooting and parameric search. *SIAM J. Comput.*, 22:794–806, 1993.

8 P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets II. *SIAM J. Comput.*, 42:2039–2062, 2013. Also in arXiv:1208.3384.

9 P. K. Agarwal and M. Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM J. Comput.*, 25(1):100–116, 1996.

10 B. Aronov, E. Ezra, and J. Zahl. Constructive polynomial partitioning for algebraic curves in $\mathbb{R}^3$ with applications. *SIAM J. Comput.*, 49(6):1109–1127, 2020. URL: `https://doi.org/10.1137/19M1257548`, `doi:10.1137/19M1257548`.

11 B. Aronov and S. Fortune. Approximating minimum-weight triangulations in three dimensions. *Discret. Comput. Geom.*, 21(4):527–549, 1999.

12 B. Aronov, E. Y. Miller, and M. Sharir. Eliminating depth cycles among triangles in three dimensions. *Discret. Comput. Geom. (Special Issue in memory of Ricky Pollack)*, 64(3):627–653, 2020.

13 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry.* Algorithms and Computation in Mathematics 10. Springer-Verlag, Berlin, 2003.

14 B. Chazelle. Fast searching in a real algebraic manifold with applications to geometric complexity. In *Colloquium on Trees in Algebra and Programming*, pages 145–156, 1985.

15 G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Proc. 2nd GI Conf. Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.

16 David Cox, John Little, and Donal O'Shea. Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra. 2007. URL: `https://link.springer.com/book/10.1007/978-0-387-35651-8`.

17 M. de Berg, D. Halperin, M. H. Overmars, J. Snoeyink, and M. J. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12(1):30–53, 1994.

18 D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theor. Comput. Sci.*, 27:241–253, 1983.

19 E. Ezra and M. Sharir. On ray shooting for triangles in 3-space and related problems. *SIAM J. Comput.*, accepted. Also in *Proc. 37th Sympos. on Computational Geometry* (2021), 34:1–34:15. Also in arXiv:2102.07310.

20 L. Guth. Polynomial partitioning for a set of varieties. *Math. Proc. Camb. Phil. Soc.*, 159:459–469, 2015. Also in arXiv:1410.8871.

21 L. Guth and N. H. Katz. On the Erdős distinct distances problem in the plane. *Annals Math.*, 181:155–190, 2015. Also in arXiv:1011.4105.

22 Larry Guth. *Ruled Surface Theory and Incidence Geometry*, pages 449–466. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-44479-6_18, doi:10.1007/978-3-319-44479-6_18.

23 J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18(3):403–431, 1995.

24 J. Matoušek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015.

25 J. Milnor. *Morse Theory*. Princeton University Press, 1963.

26 S. Mohaban and M. Sharir. Ray shooting amidst spheres in 3 dimensions and related problems. *SIAM J. Comput.*, 26:654–674, 1997.

27 M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

28 M. Pellegrini. Ray shooting and lines in space. In *Handbook on Discrete and Computational Geometry*, chapter 41, pages 1093–1112. CRC Press, Boca Raton, Florida, 3rd edition, 2017.

29 J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Appl. Math.*, 4:298–351, 1983.

30 M. Sharir and H. Shaul. Ray shooting and stone throwing with near-linear storage. *Comput. Geom. Theory Appls.*, 30:239–252, 2005.

31 A. C. Yao and F. F. Yao. A general approach to d-dimensional geometric queries (extended abstract). In *Proc. 17th Annu. ACM Symp. Theory of Computing*, pages 163–168, 1985.