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

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham, NC, USA

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY, 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 focus on 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 $\mathbb{R}^3$. These interesting special cases form the building blocks for 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.

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, Collision detection

Digital Object Identifier 10.4230/LIPIcs.SoCG.2022.4

Related Version *Full Version*: `https://arxiv.org/abs/2203.10241` [3]

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 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.

Acknowledgements We thank Peyman Afshani for sharing with us problems that have motivated our study of segment-intersection searching amid spherical caps, Ovidiu Daescu for suggesting the collision-detection application that motivated some aspects of our work, and the reviewers 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 (*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 [23]), 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 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

[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 [11] for details.

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, 14, 20, 26] for the first approach and [4, 12] 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 [22]), 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 [22, 23]. 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 among n planes or spheres in $\mathbb{R}^3$ can be answered in $O^*(n/s^{1/3})$ time using $O^*(s)$ storage, for $n \le s \le n^3$ [21, 22, 23, 25].

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 [19]. The pedestrian approach has also been applied to polygons with holes in $\mathbb{R}^2$ [5, 19], to a convex polyhedron in $\mathbb{R}^3$ [15], and to polyhedral subdivisions in $\mathbb{R}^3$ [5, 10]. Some of the ray-shooting data structures combine the pedestrian approach with the above range-searching tools [1, 9, 14].

Recently, Ezra and Sharir [16] 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 [17]. Roughly speaking, they construct a partitioning polynomial F of degree $O(D)$, for a sufficiently large constant D, using the algorithm in [4]. 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 intersecting triangles are called *wide*. For each cell τ, the algorithm of [16] recursively preprocesses the narrow triangles of τ and constructs a secondary data structure for the wide triangles at τ. A major technical result of [16] is to reduce a ray-shooting or intersection-detection query among 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 $n \le s \le n^3$; see [22, 23]. 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 [22]. The approach of [16] 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.

[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 polynomial.

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 intersected by a query triangle 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 $\frac{2t_Q-3}{3(t_Q-1)}$.

Input	Query	Storage	Query Time
Plates	Arc/Curve	4/3	2/3
Plates	Arc/Curve ($t \geq 3$ dof)	3/2	$(2t-3)/3(t-1)$
Plates	Planar arc ($t \geq 4$ dof)	3/2	$(2t-7)/3(t-3)$
Plates	Circular arc	3/2	3/5
Triangles	Arc/Curve	1	4/5
Triangles	Arc/Curve	11/9	2/3
Spherical caps	Segment	5/4	3/4
Spherical caps	Segment	3/2	27/40
Segments	Plate	3/2	1/2
Arcs/Curves (t dof)	Plate	3/2	$3(t-1)/4t$
Triangles*	Triangle	3/2	1/2
Plates (t_O dof)**	Plate (t_Q dof)	3/2	$\max\{\frac{2t_Q-7}{3(t_Q-3)}, \frac{3(t_O-1)}{4t_O}\}$
Tetrahedra*	Tetrahedron	3/2	1/2

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 [17] and of Guth and Katz [18] 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 $O^*(t(n)+k)$ 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. Section 2 and the full version [3]). 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. 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 [16], 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 approach that not only generalizes to algebraic arcs but also simplifies, in certain aspects, the technique of [16] 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. The most interesting among these tools is the construction of a carefully tailored *cylindrical algebraic decomposition* (CAD) (see [11, 13, 24] 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.

Next, we present a data structure for answering arc-intersection queries amid wide plates within a cell of the polynomial partition. 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 t-dimensional point $\tilde{\gamma}$, 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 significant 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. 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 $O^*(n^{1/2})$ in [16]. 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.

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. Using this result in our main algorithm, we improve the storage size to $O^*(n^{11/9})$, keeping the query 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. 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. Next, we 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 the full version [3].

Intersection searching with plates amid plates. The above results can be used to provide simple solutions for the case where both input and query objects are plates. For simplicity, first assume 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 (resp., type (ii) queries). Using $O^*(n^{3/2})$ storage, this results in the query time bound $O^*(n^{1/2})$, if we use the data structure from [16] 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 input and query plates, respectively. We obtain a data structure of $O^*(n^{3/2})$ size with query time $O^*(n^\rho)$, where $\rho = \max\left\{\frac{2t_Q-7}{3(t_Q-3)}, \frac{3(t_O-1)}{4t_O}\right\}$.[4]

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. 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.

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 an arc-intersection query for an arc $\gamma \in \Gamma$ 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 [17]. 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

[4] 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 $\frac{2t_Q-3}{3(t_Q-1)}$.

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 some absolute constant $c_2 > 0$. Agarwal et al. [4] 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 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. [4], 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 (see [3]) are omitted from this version, 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 $\mathbb{R}^3 \setminus Z(F_v)$ [11]. 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 $m \leq n_0$, $S(m) = O(m)$. For $m > n_0$, Propositions 2.1 and 3.2 imply that $S(m)$ obeys the recurrence:

$$S(m) \leq c_2 D^3 S(m/D^2) + O^*(m), \tag{1}$$

where c_2 is the constant as defined above. Since the recursion terminates at $m \leq n_0 = n^{1/3}$, it can be shown that the solution to the above recurrence is $S(m) = O^*(m^{3/2}/n^{1/6} + m)$. Hence, the overall size of the data structure (for $m = n$) is $O^*(n^{4/3})$. 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 $c_3 > 0$ [11]. 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 $Q(m)$ obeys the recurrence:

$$Q(m) \le c_3 D Q(m/D^2) + O^*(m^{2/3}), \tag{2}$$

where c_3 is the constant as defined above. Again, using the fact that the recursion terminates at $m \le n_0 = n^{1/3}$, it can be shown that $Q(m) = O^*(m^{2/3} + m^{1/2}n^{1/6}) = O^*(m^{1/2}n^{1/6})$. For $m = n$ we get $Q(n) = O^*(n^{2/3})$. Putting together everything, we obtain the following:

▶ **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 $\mathcal{T}$ can be answered in $O^*(n^{2/3})$ time.*

In the full version [3], we present a different technique for preprocessing a set $\mathcal{T}$ of *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 above structure 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 query procedure is the same as above except that we use at each leaf the aforementioned improved procedure. 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.*

2.3 Space/query-time trade-offs

As we show in the full version [3], 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 $t \ge 3$ 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 [20] used in the basic solution. Omitting all the details, we conclude:

▶ **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 $O^*(n^{\frac{2t-3}{3(t-1)}})$ time.*

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 dimensions (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 [13]. This tool is a central ingredient of our algorithm – see Section 3.2. A detailed description can be found in [11, Chapter 5]; a possibly more accessible treatment is given in [24, 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 [11] and by Q_b in [24]). 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 $Z(f_i) \cap Z(f_j)$, $1 \leq i < j \leq s$, 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 [11, 24] 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 $t \geq 0$ 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 $0 \leq i \leq t$; 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 [11, 13, 24]), 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).

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.

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.

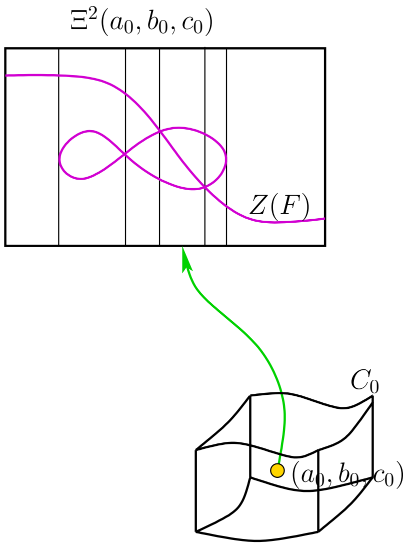

Figure 1 An illustration of the CAD construction. C_0 is a three-dimensional cell of Ξ_3. For a point $(a_0, b_0, c_0) \in C_0$, its two-dimensional fiber $\Xi^2(a_0, b_0, c_0)$ is shown. Formally, the purple curve is the xy-projection of $Z(F) \cap h(a_0, b_0, c_0)$.

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,[5] 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 $Q_{C,\gamma} := \{(a,b,c) \mid \Pi_C(\gamma; a,b,c) = 1\}$, which is of constant complexity too. By construction, γ crosses Δ_C if and only if the point $\Delta^* \in Q_{C,\gamma}$. Set $\mathcal{T}_C^* := \{\Delta^* \mid \Delta \in \mathcal{T}_C\}$.

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 [20] (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.

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

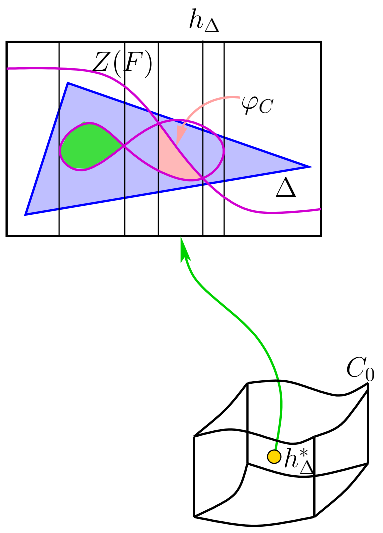

Figure 2 The labeling scheme provided by the CAD (the plate depicted in this figure is a triangle). The cell C labels, by an explicit semi-algebraic expression, the highlighted inner pseudo-trapezoidal subcell φ_C within the plate Δ. Another inner subcell, with a different label, in a different partition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

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.

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, M. J. Katz, and M. Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. `arXiv:2203.10241`.

4 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`.

5 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.

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

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

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

12 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.

13 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.

14 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.

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

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

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

18 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`.

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

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

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

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

23 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.

24 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.

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

26 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.