Discrete & Computational Geometry
https://doi.org/10.1007/s00454-024-00656-8

Intersection Searching amid Tetrahedra in Four Dimensions

Esther Ezra[1] · **Micha Sharir**[2]

Received: 21 January 2023 / Revised: 3 April 2024 / Accepted: 3 April 2024

Abstract

We develop data structures for intersection queries in four dimensions that involve segments, triangles and tetrahedra. Specifically, we study three main problems: (i) Preprocess a set of n tetrahedra in $\mathbb{R}^4$ into a data structure for answering segment-intersection queries amid the given tetrahedra (referred to as *segment-tetrahedron intersection queries*). (ii) Preprocess a set of n triangles in $\mathbb{R}^4$ into a data structure that supports triangle-intersection queries amid the input triangles (referred to as *triangle-triangle intersection queries*). (iii) Preprocess a set of n segments in $\mathbb{R}^4$ into a data structure that supports tetrahedron-intersection queries amid the input segments (referred to as *tetrahedron-segment intersection queries*). In each problem we want either to detect an intersection, or to count or report all intersections. As far as we can tell, these problems have not been previously studied. For problem (i), we first present a "standard" solution which, for any prespecified value $n \leq s \leq n^6$ of a so-called storage parameter s, yields a data structure with $O^*(s)$ storage and expected preprocessing, which answers an intersection query in $O^*(n/s^{1/6})$ time (here and in what follows, the $O^*(\cdot)$ notation hides subpolynomial factors). For problems (ii) and (iii), using similar arguments, we present a solution that has the same asymptotic performance bounds. We then improve the solution for problem (i), and present a more intricate data structure that uses $O^*(n^2)$ storage and expected preprocessing, and answers a segment-tetrahedron intersection query in $O^*(n^{1/2})$ time, improving the $O^*(n^{2/3})$ query time obtained by the standard solution. Using the parametric

Editor in Charge: Kenneth Clarkson

Work by Esther Ezra was partially supported by Grants 824/17, 800/22 from the Israel Science Foundation and US-Israel Binational Science Foundation Grant 2022131. Work by Micha Sharir was partially supported by Grants 260/18 and 495/23 from the Israel Science Foundation. A preliminary version of the paper has appeared in [17].

Esther Ezra
ezraest@cs.biu.ac.il

Micha Sharir
michas@tauex.tau.ac.il

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

2 School of Computer Science, Tel Aviv University, Tel Aviv, Israel

Published online: 03 June 2024

search technique of Agarwal and Matoušek (SIAM J Comput 22:794–806, 1993), we can obtain data structures with similar performance bounds for the *ray-shooting* problem amid tetrahedra in $\mathbb{R}^4$. Unfortunately, so far we do not know how to obtain a similar improvement for problems (ii) and (iii). Our algorithms are based on a primal-dual technique for range searching with semi-algebraic sets, based on recent advances in this area (Agarwal et al. in SIAM J Comput 50:760–787, 2021. Also in Proceedings of Symposium on Computational Geometry (SoCG) 5:1–5:14, 2019. Also in arXiv:1812.10269; Matoušek and Patáková in Discrete Comput Geom 54:22–41, 2015). As this is a result of independent interest, we spell out the details of this technique. We present several applications of our techniques, including continuous collision detection amid moving tetrahedra in 3-space, an output-sensitive algorithm for constructing the arrangement of n tetrahedra in $\mathbb{R}^4$, and an output-sensitive algorithm for constructing the intersection or union of two or several nonconvex polyhedra in $\mathbb{R}^4$.

Keywords Computational geometry · Ray shooting · Tetrahedra in $\mathbb{R}^4$ · Intersection queries in $\mathbb{R}^4$ · Polynomial partitioning · Range searching · Semi-algebraic sets · Tradeoff

Mathematics Subject Classification 52C45 · 68Q25 · 68U05

1 Introduction

In this paper we consider various intersection problems involving segments, triangles and tetrahedra in $\mathbb{R}^4$. In four dimensions, the interesting setups involve (i) intersections between (one-dimensional) query segments and (three-dimensional) input tetrahedra, (ii) intersections between (two-dimensional) query triangles and (two-dimensional) input triangles, and (iii) intersections between (three-dimensional) query tetrahedra and (one-dimensional) input segments. We study all three problems, and derive efficient solutions to each of them.

As an interesting application, we consider the *continuous collision detection* problem, where the input consists of n tetrahedra in $\mathbb{R}^3$, each of which is moving at some constant velocity of its own, and the goal is to detect whether any pair of them collide. Adding the time as a fourth coordinate, this becomes a batched version of intersection detection in $\mathbb{R}^4$, involving both setups (i) (or (iii)) and (ii). Specifically, a collision can occur when a vertex v of one tetrahedron hits a face f of another, or when an edge e of one tetrahedron hits an edge e' of another. In the four-dimensional space-time, the first event corresponds to an intersection between the ray traced by v and the three-dimensional prism traced by f (setups (i) and (iii)). The second event corresponds to an intersection between the two-dimensional strip traced by e and the strip traced by e' (setup (ii)).

Other applications include output-sensitive construction of the arrangement of n tetrahedra in $\mathbb{R}^4$, and an output-sensitive algorithm for computing the intersection or the union of two or several not necessarily convex polyhedra in $\mathbb{R}^4$. In the three-dimensional versions of these problems, which were recently studied by the authors,

the only setups that needed to be considered were segment intersection amid triangles [18] (or triangle intersection amid segments [4]). In four dimensions, though, we also face the triangle-triangle intersection problem, since we also need to find intersections between pairs of 2-faces of the input objects.

Before proceeding, we note that our results are stated and proved under the assumption that the input and query objects are in *general position*. Informally, this means that the intersection between any query and input objects occurs at the "right" dimensionality. For example, the intersection between a line and a tetrahedron should be at a single point that lies in the relative interior of the tetrahedron, and the intersection between two triangles should also occur at a single point that lies in the relative interiors of both triangles. We will later be more precise about this assumption, but comment right now that in most cases this assumption can be removed using standard perturbation techniques. In one of our applications we will need more ad-hoc techniques to address thus issue.

Setup (i): Segment-tetrahedron intersection queries. Consider first the case of query segments vs. input tetrahedra. In the setup considered here, the input objects are n (not necessarily disjoint) tetrahedra in $\mathbb{R}^4$ and the query objects are segments, and the goal is to detect, count, or report intersections between the query segment and the input tetrahedra.

As far as we can tell, this problem has not been explicitly studied so far. We first present, in Sect. 3, a "traditional" (albeit novel) solution, in which the problem is reduced to a range searching problem in a suitable parametric space, which, in the case of (lines supporting) segments in $\mathbb{R}^4$, is six-dimensional. We carefully adapt and combine recent techniques, developed by Agarwal et al. [5] and Matoušek and Patáková [26], which provide algorithmic constructions of intricate space decompositions based on polynomial partitioning. Using this machinery, we solve the problem so that, with a so-called prespecified storage parameter s, a segment intersection query can be answered in[1] $O^*(n/s^{1/6})$ time, for any $n \leq s \leq n^6$, and the storage and preprocessing cost are both $O^*(s)$.

A special case of this setup is an extension to four dimensions of the classical *ray shooting* problem, which has mostly been studied in two and three dimensions. In a general setting, we are given a collection S of n simply-shaped objects, and the goal is to preprocess S into a data structure that supports efficient ray shooting queries, where each query specifies a ray ρ and asks for the first object of S hit by ρ, if such an object exists. In this work we only consider the (already challenging) case of input tetrahedra. Using the parametric search technique of Agarwal and Matoušek [7], ray shooting queries can be reduced to segment-intersection detection queries, up to a polylogarithmic factor in the query cost. By the above discussion, we obtain the following result:

Theorem 1.1 *Given a collection $\mathcal{T}$ of n tetrahedra in $\mathbb{R}^4$, and any storage parameter $n \leq s \leq n^6$, we can preprocess $\mathcal{T}$ into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any segment-intersection or ray-shooting query in $\mathcal{T}$ in $O^*(n/s^{1/6})$ time. The query time bound applies to segment-intersection*

[1] As in the abstract, the $O^*(\cdot)$ notation hides subpolynomial factors, typically of the form n^ε, for any $\varepsilon > 0$, and their coefficients which depend on ε.

detection and counting queries (and to ray shooting queries). The cost is $O^*(n/s^{1/6}) + O(k)$ *for reporting queries, where* k *is the output size.*

The recent work of Afshani and Cheng [1] shows, for the setting of Theorem 1.1 under the pointer machine model, that any data structure that reports all intersections in $\mathcal{T}$ with a query line in $O^*(1)$ time must use space close to $\Omega(n^6)$. This suggests that our tradeoff bound is likely to be nearly tight[2] for $s = n^6$. Nevertheless, we manage to obtain an improvement when the storage parameter is quadratic. Specifically, we show, in Sect. 5:

Theorem 1.2 *A collection* $\mathcal{T}$ *of* n *arbitrary tetrahedra in* $\mathbb{R}^4$ *can be preprocessed into a data structure of size* $O^*(n^2)$*, in expected time* $O^*(n^2)$*, which supports segment-intersection detection and ray-shooting queries in time* $O^*(n^{1/2})$ *per query. A segment-intersection reporting query takes* $O^*(n^{1/2}) + O(k)$ *time, where* k *is the output size.*

This indeed improves the bound stated in Theorem 1.1, which, with $s = O^*(n^2)$ storage, has query time $O^*(n^{2/3})$. Furthermore, with the storage bound specified in Theorem 1.2, the query bound is similar to that obtained for segment-intersection (and ray-shooting) amid hyperplanes (rather than tetrahedra) in $\mathbb{R}^4$ [7]. We comment, however, that this improvement does not hold for counting queries.

We then go on to extend the result of Theorem 1.2 to obtain a tradeoff between storage (and expected preprocessing time) and query time. We obtain the following result.

Theorem 1.3 *Let* $\mathcal{T}$ *be a set of* n *tetrahedra in* $\mathbb{R}^4$*. With storage parameter* s*, which can vary between* n *and* n^6*, we can answer a segment intersection or a ray shooting query amid the tetrahedra of* $\mathcal{T}$ *in time*

$$Q(n,s) = \begin{cases} O^*\left(\frac{n^{7/6}}{s^{1/3}}\right) & \text{for } s = O(n^2), \\ O^*\left(\frac{n^{3/4}}{s^{1/8}}\right) & \text{for } s = \Omega(n^2). \end{cases} \tag{1}$$

Again, this bound pertains to detection queries, and incurs an additive term of $O(k)$ *for reporting queries, where* k *is the output size, namely the number of intersections between the query segment and the input tetrahedra.*

See Fig. 1 for an illustration. This implies the following corollary.

Corollary 1.4 *One can answer* m *segment intersection detection or ray-shooting queries, on* n *tetrahedra in* $\mathbb{R}^4$ *in*

$$\max\left\{O^*(m^{3/4}n^{7/8} + n),\ \ O^*(m^{8/9}n^{2/3} + m)\right\} \tag{2}$$

expected time and storage. The first (resp., second) bound dominates when $m \le n^{3/2}$ *(resp.,* $m \ge n^{3/2}$*). For reporting queries, the bound incurs an additive term of* $O(k)$*, where* k *is the output size.*

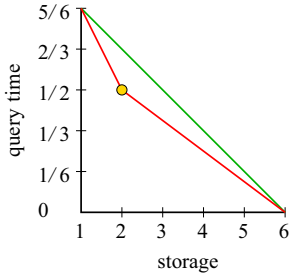

Fig. 1 The tradeoff between storage and query time. The breakpoint in the graph represents the case studied in Theorem 1.2. Both axes are drawn on a logarithmic scale. The straight green graph is the tradeoff given in Theorem 1.1, and the bent red graph is the improved tradeoff given in Theorem 1.3

Setup (ii): Triangle-triangle intersection queries. We next consider the second setup of intersection queries, where both input and query objects are triangles in $\mathbb{R}^4$. We show that this setup can also be reduced, similar to setup (i), to a multi-level range searching problem in $\mathbb{R}^6$ involving semi-algebraic ranges. This allows us to obtain the same performance bounds here too. Namely we have:

Theorem 1.5 *Given a collection $\mathbf{\Delta}$ of n triangles in $\mathbb{R}^4$ and any storage parameter $n \leq s \leq n^6$, we can preprocess $\mathbf{\Delta}$ into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any triangle-intersection query in $\mathbf{\Delta}$ in $O^*(n/s^{1/6})$ time.*

We comment that, based on the recent work of Afshani and Cheng [1], the tradeoff bound in Theorem 1.5 is nearly tight for $s = n^6$ in the pointer machine model. That is, $\Omega(n^6)$ storage is needed to ensure fast query time.

Since both input and query objects are triangles, it is also interesting to consider the bichromatic batched version of the problem. Namely we have:

Theorem 1.6 *Given two collections R and B of triangles in $\mathbb{R}^4$, of respective sizes m and n, we can detect an intersection between some triangle of R and some triangle of B, or count all such intersections, in expected time (and storage) $O^*(m^{6/7}n^{6/7} + m + n)$. We can also report all these intersections in expected time $O^*(m^{6/7}n^{6/7} + m + n) + O(k)$, where k is the output size.*

As a consequence, integrating this bound with the one obtained in Theorem 1.1 (in which we need to set $s = n^{12/7}$ to match the performance with that stated above, as is easily verified), we obtain an overall $O^*\left(n^{12/7}\right)$ expected-time solution for the continuous collision detection problem, that is:[3]

Theorem 1.7 *Given n tetrahedra in $\mathbb{R}^3$, each of which is moving at some constant velocity of its own, one can detect a collision between any pair of moving tetrahedra in $O^*\left(n^{12/7}\right)$ expected time (and storage).*

[2] See, e.g., [3], for the comparison between the RAM and the pointer machine models, as well as the justification to use the latter for range reporting computation.

[3] Here we use an obvious divide-and-conquer approach in order to reduce the general (non-bichromatic) problem to the bichromatic version.

Indeed, in the four-dimensional space-time, a tetrahedron Δ moving at some fixed velocity, traces a prism-like polytope Δ^*, of constant complexity. A collision between two moving tetrahedra Δ_1, Δ_2 in three dimensions occurs if and only if Δ_1^* and Δ_2^* have a nonempty intersection. We can therefore triangulate the boundary of each such prism-like polytope Δ^* into $O(1)$ tetrahedra, and then obtain the settings in Theorems 1.1 and 1.6 (where in the first we consider the tetrahedra and their edges, and in the latter we consider the 2-faces of these tetrahedra). We recall that we assume that both settings are in general position. For the first, we solve the bichromatic version for the tetrahedra and their edges, since the tetrahedra and the tetrahedron-edges (with which we query) lie in general position (see our comment above about standard perturbation techniques and the general position assumptions stated in Sect. 3) we can apply Theorem 1.1. In order to enforce the general position assumption for the 2-faces of the resulting tetrahedra (the input of Theorem 1.6), we apply the following steps. First, after triangulating each prism-like polytope Δ^*, we color each of the resulting 2-faces in a distinct color. By this step we partition the entire set of the 2-faces, over all Δ^*, into $O(1)$ subcollections. We next apply Theorem 1.6 for each pair of such subcollections in turn (overall, we have $O(1)$ such pairs), and report that a collision between any pair of moving tetrahedra has been detected if we detected an intersection between a pair of triangles from two different collections. As above, we use standard perturbation techniques (see also Sect. 3 and 4), in order to conclude that the resulting $O(1)$ pairs of triangle collections are in general position.

Collision detection has been widely studied—see Lin, Manocha and Kim [24] for a recent comprehensive survey, and the references therein. We are not aware of any work that addresses the exact algorithmic approach for the specific setup considered here, although there are some works, such as Canny [12] or Schömer and Thiel [29], that address similar contexts.

We then consider the applications of our techniques to the problems of output-sensitive construction of an arrangement of tetrahedra in $\mathbb{R}^4$, and of constructing the intersection or union of two or several (nonconvex and bounded) polyhedra in $\mathbb{R}^4$. Using the bounds for setups (i) and (ii), we obtain, in Sect. 7:

Theorem 1.8 (i) *Let $\mathcal{T}$ be a collection of n tetrahedra in general position*[4] *in $\mathbb{R}^4$. We can construct the arrangement $\mathcal{A}(\mathcal{T})$ of $\mathcal{T}$ in $O^*(n^{12/7} + n^{1/2}k_2) + O(k_4 \log k_4)$ randomized expected time, where k_2 is the number of intersecting pairs of tetrahedra in $\mathcal{T}$, and k_4 is the number of vertices of $\mathcal{A}(\mathcal{T})$.*
(ii) *Given two arbitrary (bounded) polyhedra R and B in $\mathbb{R}^4$, each of complexity $O(n)$ (where the complexity is the number of faces of all dimensions on their boundary), that lie in general position with respect to one another,*[5] *the intersection $R \cap B$ can be computed in expected time $O^*(n^{12/7} + n^{1/2}k_2) + O(k_4 \log k_4)$, where k_2 is the number of 2-faces of $\mathcal{A}(R \cup B)$, and k_4 is the number of vertices of $\mathcal{A}(R \cup B)$.*

As another application of our technique we present an efficient algorithm for detecting, counting or reporting intersections between n 2-flats and n lines in $\mathbb{R}^4$. We show

[4] Here general position implies that a pair of intersecting tetrahedra intersect in a two-dimensional convex polygon of constant complexity.

[5] In the context of our analysis this implies that, for any pair of intersecting tetrahedra $\Delta_1 \in \partial R$ and $\Delta_2 \in \partial B$, $\Delta_1 \cap \Delta_2$ is a a two-dimensional convex polygon of constant complexity.

that, given n lines and n 2-flats in $\mathbb{R}^4$, one can detect whether some line intersects some 2-flat in $O^*(n^{13/8})$ expected time, or count the number of such intersections. One can also report all k intersections in $O^*(n^{13/8}) + O(k)$ expected time. This result is a degenerate special case of the triangle-triangle intersection setup (ii), and admits a faster solution. (Note that in general position 2-flats and lines are not expected to meet in $\mathbb{R}^4$, which makes this special case interesting. One can also regard this problem as a variant, in four dimensions, of *Hopcroft's problem*, seeking to detect an incidence between n points and n lines in the plane.)

Setup (iii): Tetrahedron-segment intersection queries. This is a symmetric version of setup (i), where the input consists of n segments in $\mathbb{R}^4$ and the query is with a tetrahedron T, where the goal is to detect, count or report intersections between T and the input segments. Using a similar machinery, we obtain the same asymptotic performance bounds, as in the previous standard solutions, for this setup too.

Theorem 1.9 *Given a collection S of n segments in $\mathbb{R}^4$, and any storage parameter $n \leq s \leq n^6$, we can preprocess S into a data structure of size $O^*(s)$, in randomized $O^*(s)$ expected time, so that we can answer any tetrahedron-intersection query in S in $O^*(n/s^{1/6})$ time. The query time bound applies to tetrahedron-intersection detection and counting queries. The cost is $O^*(n/s^{1/6}) + O(k)$ for reporting queries, where k is the output size.*

The paper is organized as follows. We begin with a short preliminary section (Sect. 2), where we briefly review some basic notions used in the paper, such as range searching and semi-algebraic sets. We then present, in Sect. 3, the standard (but novel) technique for setup (i). A simple modification of the algorithm, also presented in Sect. 3, yields an algorithm for setup (iii). The algorithm for setup (ii) is then presented in Sect. 4. The improved algorithm for setup (i) is presented in Sect. 5. This improved solution can be extended to yield an improved tradeoff between storage and query time, the details of which are given in Sect. 6. Our applications, for constructing arrangements of tetrahedra in $\mathbb{R}^4$, constructing the intersection or union of nonconvex polyhedra in $\mathbb{R}^4$, and continuous collision detection, are presented in Sect. 7. Finally, in Sect. 8, we study the special case of intersections between 2-flats and lines in $\mathbb{R}^4$.

2 Preliminaries

In this section we briefly review some basic concepts and tools that we will be using in our analysis.

The problems studied in this paper are solved via a reduction to *semi-algebraic range searching*. Specifically, we are given a set P of n points in $\mathbb{R}^d$, which we want to preprocess into a data structure that supports *range searching* queries. Each such query specifies a range K, which is a region of some shape in $\mathbb{R}^d$, and the goal is either to detect whether K contains a point of P (referred to as *range emptiness*), or to count the number of these points (*range counting*), or to report all of them (*range reporting*). Range searching has been thoroughly reviewed in several surveys, such as the recent survey of Agarwal [3], and we refer the reader to this survey for details.

Range searching has been extensively studied for more than three decades, but most of these studies have focused on the case where K is a halfspace, bounded by a hyperplane, or, more generally, where K is a simplex. Only recently, focus has shifted to the case where the ranges are semi-algebraic. A set is called semi-algebraic if it is defined as a Boolean combination of polynomial equalities and inequalities. It is said to have constant complexity if the number of polynomials and their maximum degree are both bounded by a constant. In what follows we assume that the ranges under consideration have *e degrees of freedom*, meaning that each of them can be specified in terms of e real parameters.

We assume that the model of computation is the *real RAM* model, where algebraic manipulation of a constant number of polynomials of constant maximum degree can be performed exactly in constant time. Such manipulations include computing the roots of a polynomial equation, in the sense that any polynomial sign test involving such a root can be performed exactly (in constant time), and all kinds of algebraic operations of a similar nature. See the book of Basu, Pollack and Roy [11], for details of such computations, as well as the studies in [8, 10] where such a model of computation was used.

The ability to perform range searching with semi-algebraic sets is due to recent algorithmic advances in the theory of *polynomial partitioning*. The technique has been introduced by Guth and Katz [21] in 2010, and has later been extended by Guth [20] in 2015. In a nutshell, these works show that, given a collection S of n k-dimensional varieties in $\mathbb{R}^d$, for any $0 \leq k \leq d-1$, of constant maximum degree, and for a specified parameter D, one can construct a polynomial $F \in \mathbb{R}[x_1, \ldots, x_d]$ of degree $O(D)$, so that its zero set $Z(F)$ partitions $\mathbb{R}^d$ into $O(D^d)$ cells, each being a connected component of $\mathbb{R}^d \setminus Z(F)$, so that each cell τ is crossed by at most n/D^{d-k} varieties of S. This yields a powerful divide-and-conquer mechanism, that has been used effectively to solve many combinatorial problems, involving incidences, distinct distances, and many other topics; see the book of Sheffer [32] for more details.

The technique has been initially combinatorial in nature, as it lacked efficient algorithms for the actual construction of partitioning polynomials. It has also suffered from the problem that the partition guarantees good bounds within each of its cells, but not necessarily on the zero set $Z(F)$ itself. Many works have addressed this latter issue, but a full satisfactory solution was hard to come by.

Only recently, the missing algorithmic part has been supplied, in a couple of fundamental works, by Matoušek and Patáková [26] and by Agarwal, Aronov, Ezra and Zahl [5] (see also somewhat earlier works [8, 10]).

The crucial technical tool in [5], on which their technique is based, is the following result. We give here a restricted specialized version that suffices for our purposes, as we apply it in $d = 6$ dimensions. (When applying this tool in d dimensions, the parameter 6 has to be replaced by d.)

Theorem 2.1 (A specialized version of Agarwal et al. [5, Corr. 4.8]) *Given a set Ψ of N constant-degree algebraic surfaces in $\mathbb{R}^6$, and a sufficiently small parameter $0 < \delta < 1/6$, there are finite collections $\Omega_0, \ldots, \Omega_6$ of semi-algebraic sets in $\mathbb{R}^6$ with the following properties.*

- *For each index i, each cell $\omega \in \Omega_i$ is a connected semi-algebraic set of constant complexity. The size $|\Omega_i|$ of Ω_i (the number of its sets) is a constant that depends on δ.* [6]
- *For each index i and each $\omega \in \Omega_i$, at most $\frac{N}{4|\Omega_i|^{1/6-\delta}}$ surfaces from Ψ cross ω (intersect ω without fully containing it).*
- *The cells partition $\mathbb{R}^6$, in the sense that $\mathbb{R}^6 = \bigsqcup_{i=0}^{6} \bigsqcup_{\omega\in\Omega_i} \omega$, where $\bigsqcup$ denotes disjoint union.*

The sets in $\Omega_0, \ldots, \Omega_6$ can be computed in $O(N)$ expected time, where the constant of proportionality depends on δ, by a randomized algorithm. For each i and for every set $\omega \in \Omega_i$, the algorithm returns a semi-algebraic representation of ω, a reference point inside ω, and the subset of surfaces of Ψ that cross ω.

Here is a brief overview of the range searching technique that we use. Suppose, for specificity, that each range in the family S under consideration is defined as the conjunction of t inequalities $f_i(\mathbf{x}, \mathbf{y}) \leq 0$, for $i = 1, \ldots, t$, where f_i are constant-degree polynomials in $\mathbf{x} \in \mathbb{R}^d$, the coordinates of the points of P, and in $\mathbf{y} \in \mathbb{R}^e$, which is a vector representing the at most e real parameters specifying a range in S.

There are two ways to represent the problem, which we refer to as the *primal* and *dual* settings. In the primal, standard setting, we work in the d-dimensional *object space*, where the points of P are represented as points and the ranges as regions. In the dual, we work in the e-dimensional *query space*, where the ranges are represented as points and the points of P as ranges, where the range σ_p associated with a point p is the set of all points representing regions of S that contain p. The problem that we face in the dual is also referred to as *point enclosure* searching; that is, the input is a set of regions, the query is a point q, and the goal is to detect whether q is contained in some input region, or to count or report all such regions.

In the primal, we handle the conjunction of the t inequalities using a *multi-level* search tree, where each level of the tree caters to one of the inequalities. At each level, we construct a partition of space (that is of $\mathbb{R}^d$) into $O(1)$ cells, and associate with each cell τ the set $P_\tau = P \cap \tau$, to which we refer as a *canonical set*. Assuming we are not yet at the final level, each canonical set is passed to the next level, and the points in it are partitioned using a similar scheme.

When we query with a range K, we find, at each level i that we process, in $O(1)$ time, all the cells that are fully contained in the 'halfspace' $f_i(\mathbf{x}, \mathbf{y}_K) \leq 0$, and the cells that this halfspace crosses (intersects but does not fully contain), where $\mathbf{y}_K$ are the parameters that specify K (as above). In fact, due to our general position assumption, we actually consider the open halfspace $f_i(\mathbf{x}, \mathbf{y}_K) < 0$. We then continue the query recursively, at the same level, in each cell of the second type. For each cell τ of the first type, we continue the query at the next-level structure associated with P_τ, which we query with the inequality involving f_{i+1}. When we reach the last level, the points in the first kind of canonical sets that the query has reached can be trivially detected,

[6] This latter property is not explicitly mentioned in [5]. However, it follows from the proof details there, where it is shown that $|\Omega_i| \leq D^{i+O(\delta)}$, for some constant D representing the degree of a partitioning polynomial at "level i".

counted, or reported, using the fact that these sets are pairwise disjoint, by construction. Canonical sets of the second kind will be processed recursively, within the same level, until we reach subproblems of constant size, and their points are then handled by brute force.

The construction is different in the dual, for point-enclosure queries. We also use here a multi-level structure, one level for each inequality. Here we have a set S of constant-complexity semi-algebraic regions (namely, at the i-th level, the region associated with a point p is $\{\mathbf{y} \in \mathbb{R}^e \mid f_i(\mathbf{x}_p, \mathbf{y}) < 0\}$, where $\mathbf{x}_p$ are the coordinates of p), and we query with a point. Using Theorem 2.1, we partition space (this time $\mathbb{R}^e$), at each level, into a constant number of cells, as prescribed there, and for each cell τ of the partition, we construct the set S^0_τ of the regions that fully contain τ, and the set S_τ of the regions that cross τ. We recursively process each of these sets, the sets S^0_τ at the next level, and the sets S_τ at the same level. (More precisely, when handling a conjunction of inequalities, the sets S^0_τ are replaced by similarly defined sets using f_{i+1} instead of f_i.)

A query with a point q is easy to answer. We locate, in $O(1)$ time, the cell τ containing q, and continue the query recursively, with S^0_τ at the next level, and with S_τ at the same level.

In the primal, the storage used by the structure is near-linear. The query cost depends on the number of cells crossed by the query region K. Skipping the details in this overview (for which see [3, 5, 26]), which are based on properties of polynomial partitions, as given, e.g., in [26], this cost is shown to be $O^*(n^{1-1/d})$.

In the dual, the cost of a query is easily seen to be $O(\log n)$. Using Theorem 2.1 and the analysis around it given in [5], one can show that the storage is $O^*(n^e)$. Again, we omit the details in this overview.

The review of multi-level structures given above assumes that each inequality $f_i(\mathbf{x}, \mathbf{y}) < 0$ uses the same number d of the coordinates of $\mathbf{x}$ and the same number e of the parameters of the ranges in $\mathbf{y}$. In practice, and in some of the applications given in this paper, this does not have to be the case. Nevertheless, the performance bounds (the query cost in the primal and the storage in the dual) continue to be the same. More precisely, as can be shown, the parameter d in the former bound is the maximum number of coordinates of the points of P used at any of the polynomials f_i, and the parameter e in the latter bound is the maximum number of parameters of the ranges that are used at any polynomial.

The primal and dual techniques can be combined, in order to obtain a *tradeoff* between the storage and the cost of a query. This is done by constructing the primal structure up to a certain depth, in which the subproblems have a prescribed 'intermediate' size, and then by handling each subproblem in the dual. When $d = e$, as will be the case in most of the applications of these techniques in this paper, one obtains a structure that uses $O^*(s)$ storage and answers a query in $O^*(n/s^{1/d})$ time, for any so-called 'storage parameter' $n \le s \le n^d$.

These tradeoff bounds are useful in the offline, or batched mode, where we have m queries given in advance. As shown in several recent works (such as the appendix in [4]), the overall cost of processing these queries, for the case $d = e$, is $O^*(m^{d/(d+1)}n^{d/(d+1)} + m + n)$.

3 Segment-Intersection amid Tetrahedra: An Initial Algorithm

In this section we prove Theorem 1.1. Specifically, we present an initial solution to the problem of segment-intersection detection amid tetrahedra in four dimensions, which is based on a careful combination of the recent semi-algebraic range searching machinery of [5, 26]. As far as we can tell, such a solution has not yet been presented in the literature. Also, the adaptation of the available techniques to this problem is not simple, requires nontrivial and careful enhancements, and is sufficiently novel, in our opinion, to be of independent interest. Moreover, this gives a yardstick for appreciating the improvement obtained by our improved algorithm, presented in Sect. 5. The machinery developed here will also be used, with some appropriate modifications, in the algorithms for handling setups (ii) and (iii).

The parametric search technique of Agarwal and Matoušek [7] reduces ray shooting queries to segment-intersection detection queries, so it suffices to consider the latter problem. The reporting and counting variants are simple extensions of the same technique, as will be discussed as we go.

To obtain a tradeoff between the storage of the structure (and its preprocessing cost) and the query time, our algorithm uses a primal-dual approach. However, both the primal and dual setups suffer from the fact that, in four dimensions, segments and tetrahedra require too many parameters to specify. Specifically, a segment requires eight parameters (e.g., by specifying its two endpoints), while a tetrahedron requires 16 parameters (e.g., by specifying the coordinates of its four vertices).

To address this issue, we use a multi-level data structure, where each level caters to one aspect of the condition that a segment crosses a tetrahedron. This is done so that, at each of these levels, the number of parameters that a segment or a tetrahedron requires is at most six.

Specifically, the condition that a segment e, that lies on a line ℓ, intersects a tetrahedron Δ, supported by a hyperplane h_Δ, is the conjunction of the following conditions:

(i) The two endpoints of e lie on different sides of h_Δ.
(ii) With a suitable choice of an orientation of Δ, ℓ has the same orientation with respect to each of the 2-planes that support the four 2-faces of Δ, where each 2-plane is oriented consistently with Δ.

We remind the reader that we assume general position. In particular, this means that the sidedness in condition (i) is with respect to the open halfspaces bounded by h_Δ, and that the orientations in condition (ii) are positive or negative but not zero.

Informally, concerning the notion of orientation in (ii), a line ℓ is oriented by specifying the order of some pair of points $p_{\ell,1}$, $p_{\ell,2}$ on ℓ. A 2-plane π is oriented by specifying the circular order of three noncollinear points $q_{\pi,1}$, $q_{\pi,2}$, $q_{\pi,3}$ on π. The relative orientation of ℓ with respect to π is the orientation of the 5-tuple

$$\left(p_{\ell,1},\ p_{\ell,2},\ q_{\pi,1},\ q_{\pi,2},\ q_{\pi,3}\right).$$

See (3) and the discussion around it for more precise details.

Condition (i) is the conjunction of two sub-conditions, each testing the position of some endpoint of e with respect to the hyperplanes h_Δ. Condition (ii) is the disjunction

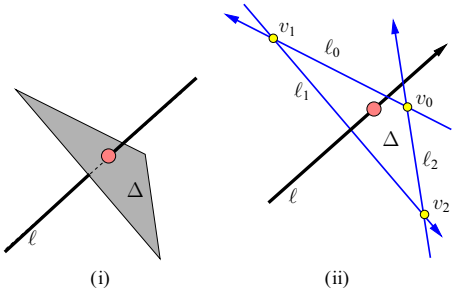

Fig. 2 An illustration of condition (ii) in three dimensions. (i) The line ℓ intersects the triangle Δ. (ii) We assign appropriate directions to ℓ and to the lines supporting the edges of Δ, so that the directed ℓ intersects Δ if and only if ℓ is positively oriented with respect to each of the three directed edge-supporting lines of Δ. These three conditions are enforced by three respective orientation tests

of two conjunctions of four sub-conditions each, where the first (resp., second) disjunction involves tests that check that the orientation of ℓ with respect to the 2-planes supporting specific 2-faces of the tetrahedra are all positive (resp., negative). Thus, the dual structure is the union of two substructures, each of which has six levels, two for testing the sub-conditions of condition (i) and four for testing the sub-conditions of condition (ii). For simplicity, we consider only one substructure, where the orientations have to be positive. See Fig. 2 for a three-dimensional illustration of this condition.

More precisely, each but the last level is a collection of structures, each of which operates on some canonical subset of the input tetrahedra, produced at the previous levels, and collects all the tetrahedra Δ of that subset that satisfy the corresponding sub-condition for the query segment (that a specific endpoint of e lies in a specific side of h_Δ for the first two levels, or that the oriented 2-plane supporting a specific 2-face of Δ is positively oriented with respect to the directed line ℓ for the last four levels), as the disjoint union of precomputed canonical sets of tetrahedra. The last level just tests whether the last sub-condition is satisfied for any tetrahedron in the current canonical set.

We use the fact that lines in $\mathbb{R}^4$ require six real parameters to specify. The space of lines in $\mathbb{R}^4$ is actually projective, but for simplicity of presentation we regard it as a real space, and ignore the special cases in which the real representation fails. Handling these cases follows the same approach, and is in fact simpler, as it uses fewer parameters.

One simple way to represent a line ℓ in $\mathbb{R}^4$ is by the points $u_\ell^0 = (x_0, y_0, z_0, 0)$ and $u_\ell^1 = (x_1, y_1, z_1, 1)$ at which ℓ crosses the hyperplanes $w = 0$ and $w = 1$, respectively (ignoring lines that are orthogonal to the w-axis), so the line ℓ can be represented as the point $p_\ell = (x_0, y_0, z_0, x_1, y_1, z_1)$ in $\mathbb{R}^6$, as desired.

Similarly, 2-planes in $\mathbb{R}^4$ also require six parameters to specify. This is simply because the duality in $\mathbb{R}^4$ maps lines to 2-planes and vice versa, but a concrete way to represent 2-planes by six parameters, which we will use, is to specify three points on a 2-plane π that are intersections of π with three fixed 2-planes, such as, say, $x = y = 0$, $x = 0$ and $y = 1$, and $x = y = 1$ (again ignoring special directions of π).

Each of the intersection points has two degrees of freedom (as two of its coordinates are fixed), for a total of six. Denote these points as $v_\pi^{(00)}$, $v_\pi^{(01)}$, and $v_\pi^{(11)}$, and put $q_\pi = \left(v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)}\right)$, listing only the w- and z-coordinates of each point, so q_π is indeed a point in $\mathbb{R}^6$.

These observations are meaningful only for the last four levels of the structure. The first two levels are simpler, as they deal with points (the endpoints of e) and hyperplanes (those supporting the tetrahedra of $\mathcal{T}$) in $\mathbb{R}^4$. Thus each of the first two levels is a halfspace range searching structure for points and halfspaces in $\mathbb{R}^4$. (Actually, this is the case when we pass to the dual 4-space; in the primal we have a point-enclosure problem, where the query is a point and the input consists of halfspaces bounded by the relevant hyperplanes.) Using standard techniques (see, e.g., [3]), this can be done, for N halfspaces in the current canonical subset and using $O^*(N)$ storage, so that a query costs $O^*(N^{3/4})$ time.[7] This cost will be subsumed by the query time bounds for the last four levels. The cost of a query includes the cost of reporting its output, as a list of canonical sets.

We next consider the (more involved) situation in the last four levels of the structure. Here the query segment is replaced by its supporting oriented line ℓ, and each tetrahedron Δ is replaced by the oriented 2-plane supporting a specific 2-face of Δ, one 2-face for each level. In the primal setup, the line ℓ is represented as a point p_ℓ in (projective) 6-space, in the manner just described, and a tetrahedron Δ, represented by a suitable oriented 2-plane π supporting its 2-faces (due to the multi-level approach, we consider only one 2-face of Δ a time), is represented as a semi-algebraic region K_π, consisting of all points that represent (directed) lines that are positively oriented with respect to π. The problem that we face is a point-enclosure query, in which we want to collect all the regions K_π that contain p_ℓ. In the dual setup, the 2-planes π are represented as points in $\mathbb{R}^6$, and the (directed) query line ℓ is represented as a semi-algebraic region Q_ℓ that consists of all (oriented) 2-planes that are positively oriented with respect to ℓ. The problem here is a semi-algebraic range searching query, where we want to collect all the input points in Q_ℓ.

The orientation test of ℓ with respect to π amounts to computing the sign of the 5×5 determinant

$$\begin{vmatrix} u_\ell^0 & 1 \\ u_\ell^1 & 1 \\ v_\pi^{(00)} & 1 \\ v_\pi^{(01)} & 1 \\ v_\pi^{(11)} & 1 \end{vmatrix}, \tag{3}$$

with a suitable orientation of the pair of points u_ℓ^0, u_ℓ^1 on ℓ (dictating the direction of ℓ), and of the triple of points $v_\pi^{(00)}$, $v_\pi^{(01)}$, $v_\pi^{(11)}$ on π (dictating the orientation of π). Again, our general position assumption requires that the sign in (3) is either positive or negative but not zero.

[7] A tradeoff between storage and query time is also available, but we do not need it here.

To compute these signs, at each of the four latter levels of the structure, we use a primal-dual approach, where the top part of the structure is in the primal, and at each of its leaf nodes (i.e., intermediate nodes at some suitable level) we pass to the dual.

The dual setup. The dual setup is simpler, so we begin with its description. In the dual setup, each tetrahedron Δ of the current canonical subset of $\mathcal{T}$ is mapped to the point $q_\pi = \left(v_\pi^{(00)},\ v_\pi^{(01)},\ v_\pi^{(11)}\right)$ in $\mathbb{R}^6$, where π is the 2-plane supporting the 2-face of Δ that corresponds to the present level. More precisely, the coordinates of q_π are suitably permuted to represent the correct orientation of π (with respect to Δ). As just mentioned, the query line ℓ is mapped to a semi-algebraic region Q_ℓ of constant complexity in $\mathbb{R}^6$, consisting of all points q_π that represent (oriented) 2-planes that have positive orientation with respect to ℓ, that is, the corresponding determinant in (3) is positive. The case of negative orientation needs to be handled too, in a fully symmetric manner, but for specificity we stick with the case of positive orientation. (Q_ℓ is in fact defined by a single polynomial inequality, where the polynomial is cubic in q_π.)

As already mentioned, the task at hand, at each but the last level, is to collect the points q_π that lie in Q_ℓ, as the disjoint union of a small number of precomputed canonical sets of tetrahedra, and the task at the last level, for detection queries, is to determine whether Q_ℓ contains any point q_π, for π corresponding to the last 2-faces of the tetrahedra in the present canonical subset of $\mathcal{T}$. For counting queries, we add the size of each output canonical set to a global counter, and for reporting queries we output the elements of each output set. (For counting and reporting queries we use the fact that the canonical sets produced by the structure for a specific query are pairwise disjoint, as easily follows from our construction, see also Sect. 2.) In other words, we have, at each of these levels, a problem involving range searching with semi-algebraic ranges in $\mathbb{R}^6$. Using the algorithm of Matoušek and Patáková [26], which is a simplified version of the algorithm of Agarwal et al. [8], this can be done, for N tetrahedra, with $O^*(N)$ storage and expected preprocessing time, so that a detection or counting query takes $O^*(N^{5/6})$ time (including the cost of reporting, without enumerating, the output canonical sets). Reporting queries are handled and analyzed in a suitably modified manner; see below. See [3, Thm. 6.1] for more details.

The primal setup. With this procedure at hand, we go back to the primal structure, at each of the last four levels. As noted, the problem that we face there is a point enclosure problem, where the input consists of some N constant-complexity semi-algebraic regions in $\mathbb{R}^6$ of the form K_π, and the query is the point p_ℓ that represents ℓ, as defined earlier, and the task is to collect all the regions K_π that contain p_ℓ, as the disjoint union of a small number of precomputed canonical sets, or, at the last level, to determine whether p_ℓ is contained in any such region (or output the number of such regions, or report all of them). Here K_π is given by a single polynomial inequality, and the polynomial is quadratic in p_ℓ.

This problem has recently been studied in Agarwal et al. [5], using a multi-level polynomial partitioning technique, for the case where we allow maximum storage for the structure (that is, $O^*(N^6)$ in our case) and want the query time to be logarithmic. We next show that the structure can be modified so that its preprocessing stops

'prematurely' when its overall storage attains some prescribed value, and each of the subproblems at the new leaves can be handled via the dual algorithm presented above.

We remark that a more recent study of similar problems [4] contains a similar detailed analysis of this technique. We also remark that, with some care, we could have switched the roles of the top and bottom parts of the structure, so that the top part deals with the dual setup and the bottom part with the primal. In that case the technique that one would have to adapt is the complementary one of Matoušek and Patáková [26] rather than that of [5].

The crucial technical tool in [5], on which their technique is based, is Theorem 2.1, as stated in Sect. 2.

In our case, the surfaces of Ψ are the boundaries of the regions K_π (each defined by a single quadratic polynomial equation; see (3)). A straightforward enhancement of the algorithm of [5] also yields, for each i and each $\omega \in \Omega_i$, the set of regions K_π that fully contain ω, within the same asymptotic time bound.

We compute the partition of Theorem 2.1 and find, for each $\psi = \partial K_\pi \in \Psi$, the sets $\omega \in \Omega_i$, over all $i = 0, \ldots, 6$, that ψ crosses, and those that are fully contained in K_π. For each i and $\omega \in \Omega_i$, let $\mathcal{K}_{i,\omega}$ (resp., $\mathcal{K}^0_{i,\omega}$) denote the set of tetrahedra $\Delta \in \mathcal{T}$ for which ∂K_π crosses ω (resp., K_π fully contains ω).

The overall size of the sets $\mathcal{K}^0_{i,\omega}$, over all i and $\omega \in \Omega_i$, is $O(N)$, with a constant that depends on δ (that is, on the sizes $|\Omega_i|$, which depend on δ).

For each i and ω we also have a recursive subproblem that involves the subset $\mathcal{K}_{i,\omega}$ of the tetrahedra Δ for which ∂K_π crosses ω. Putting $r_i := |\Omega_i|$, for $i = 0, \ldots, 6$, we have, for each i and ω, $|\mathcal{K}_{i,\omega}| \le \dfrac{N}{4r_i^{1/6-\delta}}$. We run the recursion, but not all the way through, as in [5]. Instead, we use the following storage allocation rule. We fix the storage that we are willing to allocate to the structure, and distribute it among the nodes of the recursion, as follows. To simplify the analysis, we distinguish between the storage itself, and the so-called *storage parameter* s, which is what we actually manage, but we have the property that the actual storage will always be $O^*(s)$, as will be the preprocessing cost.

Let s be the storage parameter that we allocate at the root of the structure. For each i and each set $\omega \in \Omega_i$, we allocate the storage parameter $s/(4|\Omega_i|)$ for ω. Hence, when we reach some set ω at a deeper level of recursion, say level j, the storage parameter allocated to ω is $\dfrac{s}{4^j|\Omega^{(1)}_{i_1}| \cdot |\Omega^{(2)}_{i_2}| \cdots |\Omega^{(j)}_{i_j}|}$, where $\Omega^{(1)}_{i_1}, \Omega^{(2)}_{i_2}, \ldots, \Omega^{(j)}_{i_j}$, for indices $0 \le i_1, i_2, \ldots, i_j \le 6$, are the partition families at the ancestors of ω in the recursion.

We stop the recursion when we reach nodes for which the allocated storage parameter is (roughly) equal to the number of tetrahedra at the node; a more precise statement of the termination rule is given shortly.

Put, for each set ω, $r_\omega := |\Omega^{(1)}_{i_1}| \cdot |\Omega^{(2)}_{i_2}| \cdots |\Omega^{(j)}_{i_j}|$, using the above notation for ω. The storage parameter allocated to ω is thus $s/(4^j r_\omega)$. Also, by Theorem 2.1, the number of tetrahedra Δ that participate in the subproblem at ω is at most

$$\frac{n}{4^j|\Omega^{(1)}_{i_1}|^{1/6-\delta} \cdot |\Omega^{(2)}_{i_2}|^{1/6-\delta} \cdots |\Omega^{(j)}_{i_j}|^{1/6-\delta}} = \frac{n}{4^j r_\omega^{1/6-\delta}},$$

and the stopping condition that we use is that

$$\frac{s}{4^j r_\omega} = \frac{n}{4^j r_\omega^{1/6-\delta}}, \quad or \quad r_\omega = (s/n)^{(6/5)/(1+6\delta/5)}.$$

The size of a subproblem at a leaf is (using the $O^*(\cdot)$ notation to hide exponents that are proportional to δ and constants of proportionality that depend on δ)

$$n_\omega = \frac{n}{4^j r_\omega^{1/6-\delta}} = \frac{1}{4^j} O^*\left(\frac{n^{6/5}}{s^{1/5}}\right) = O^*\left(\frac{n^{6/5}}{s^{1/5}}\right).$$

At each leaf ω we pass to the dual structure reviewed above. It uses $O^*(n_\omega)$ storage and answers a query in time $O^*(n_\omega^{5/6}) = O^*(n/s^{1/6})$. To answer a query with a line ℓ in the combined structure, we first begin with the primal structure (recall that it supports point-enclosure queries), where we query with the point p_ℓ representing ℓ, and we locate the leaf cell ω that contains p_ℓ. From the analysis in [5] this search costs $O(\log n)$ time, with a constant of proportionality that depends on δ (see below). We then search with Q_ℓ in the dual structure at ω, which takes, as just noted, $O^*(n/s^{1/6})$ time. The overall cost of the query is therefore $O^*(n/s^{1/6})$.

As to the actual storage used by the structure, the allocation mechanism ensures that each level of the recursion uses storage that is at most $7/4$ times larger than the storage used in the previous level, because each node has seven child collections $\Omega_0, \ldots, \Omega_6$, each of which is allocated an amount of storage $s/4$. Hence the overall storage used is $O((7/4)^j s)$, where j is the recursion depth. Arguing as in the query time analysis, we can make the factor $(7/4)^j$ to be $O(s^\delta)$, for any small $\delta > 0$. That is, the overall storage used is $O(s^{1+\delta})$, or, in our notation, $O^*(s)$.

The above description of the structure applies to any single level among the four latter levels of the structure. The first two levels are considerably simpler and more efficient. The primal-dual approach is straightforward for halfspace range searching, and the parametric dimension is only four for the first two levels. The standard machinery (reviewed, e.g., in [3]) implies that, with s storage and N input tetrahedra, the cost of a query at each of these levels is $O^*(N/s^{1/4})$.

Putting everything together, and using standard arguments in the analysis of multi-level structures (see [3, Thm. 6.1] for details, and see also the appendix in [4]), the overall size of the six-level structure is $O^*(s)$, for any prescribed storage parameter s between n and n^6, and a query takes $O^*(n/s^{1/6})$ time. That is, this finally concludes the proof of Theorem 1.1 for the case of intersection detection queries. Counting and reporting queries are handled similarly, with a similar analysis, exploiting the fact that the decomposition in Theorem 2.1 is into disjoint subsets, as is a similar decomposition used in the machinery of [26]. For reporting queries, their cost involves an additional term $O(k)$, where k is the output size, which is obtained simply by reporting all the tetrahedra in each canonical set that the search reaches. □

Remark Our mechanism is in fact a special instantiation of the following general result, which is of independent interest, and which yields a trade-off bound for semi-algebraic

range searching in any dimension d. That is, consider a general problem of this kind, that involves n points in $\mathbb{R}^d$, and aims to answer semi-algebraic range queries, where the ranges have constant complexity, and each range has d degrees of freedom (so the problem has a symmetric dual version). One can then show, using the suitable general forms of the constructions in [5, 26], that our analysis can be applied almost verbatim in order to solve such a problem in time $O^*(n/s^{1/d})$ per query, using $O^*(s)$ space and preprocessing, where s is any parameter between n and n^d; see Sect. 2 and [3]. These queries include detecting whether a query range contains any input point, counting the number of such points, or reporting them (with an additional term $O(k)$ in the query cost, where k is the output size). Using duality, we obtain the same performance bounds for point-enclosure queries, where the input consists on n constant-complexity semi-algebraic regions in $\mathbb{R}^d$, and the query is with a point p, where the goal is to detect, count or report containments of p in the input regions. The same asymptotic bound is obtained for simplex range searching [3], but our analysis shows that this bound corresponds to a much more general family of query ranges. The two extreme cases $s = n$ and $s = n^d$ have been treated in [26] and [5], respectively, but the tradeoff between these extreme cases has not been treated explicitly (for $d > 4$), as far as we can tell. As evidenced in the preceding analysis, this tradeoff is not as routine as one might think, because of the complicated nature of the partitioning used in Theorem 2.1 (as well as in [26, Thm. 1.1]). We summarize this result in the following corollary:

Theorem 3.1 *Let P be a set of n points in $\mathbb{R}^d$, for any dimension d, and let Γ be a family of semi-algebraic ranges of constant complexity in $\mathbb{R}^d$, each of which has d degrees of freedom. Let $n \leq s \leq n^d$ be a prespecified storage parameter. Then one can preprocess P into a data structure of storage and preprocessing $O^*(s)$, such that a range-query, with a range $\gamma \in \Gamma$, can be answered in $O^*(n/s^{1/d})$ time. Such queries include detecting whether γ contains any point of P, counting the number of such points, and reporting them (with an additional $O(k)$ term in the latter case, where k is the number of these points). The same performance bounds apply to the dual point-enclosure case, where the input consists of n regions from Γ and the query is with a point $p \in \mathbb{R}^d$.*

Remarks **(a)** Theorem 3.1 can be extended to the case where the number of degrees of freedom of the ranges is different from d, but the resulting performance bound has a more complicated expression, which is not spelled out in this work. See, e.g., the appendix in [4] for a recent study that handles the asymmetric setup. See also the recent studies of Afshani and Cheng [1, 2] for larger lower bounds on semi-algebraic range searching, which arise when the ranges have more degrees of freedom.
(b) We note that our technique can be extended to segment intersection detection queries amid a collection of n $(d-1)$-simplices in any dimension d. In that case the structure has $d+2$ levels. The first two levels ensure that the endpoints of the query segment e lie on different sides of the hyperplane containing the input simplex Δ, and are implemented by halfspace range searching structures in $\mathbb{R}^d$. The last d levels ensure that the line containing e has positive orientation with respect to each of the $(d-2)$-flats containing the facets of Δ, with suitable orientations of the line and the flats. Since lines and $(d-2)$-flats in $\mathbb{R}^d$ have $2d-2$ degrees of freedom, these levels are implemented using semi-algebraic range searching structures, where both primal

and dual parts are in $\mathbb{R}^{2d-2}$. Hence the cost of the query at each of the last d levels dominates the overall cost, which is thus $O^*(n/s^{1/(2d-2)})$. The parameter s can vary between n and n^{2d-2}.

Setup (iii). A very similar mechanism, with the same performance bounds, handles the reverse situation of setup (iii), in which the input is a set of n segments in $\mathbb{R}^4$, and the query is with a tetrahedron T, and the goal is to detect, count, or report intersections between T and the input segments. The algorithm and its analysis are very similar to those given above (see once again conditions (i)–(ii), as well as (3) and the discussion around that part), except that we have to flip the roles of points and hyperplanes (in the first two levels of the structure) and of lines and 2-planes (in the last four levels of the structure).

The resulting algorithm is what is asserted in Theorem 1.9.

4 Triangle-Triangle Intersection Queries in $\mathbb{R}^4$

Let $\mathbf{\Delta}$ be a set of n triangles in $\mathbb{R}^4$. We consider various triangle-triangle intersection problems, the simplest of which is just to detect whether a query triangle intersects any triangle of $\mathbf{\Delta}$. Alternatively, we may want to count or to report all such intersections. For concreteness we focus on the detection problem in what follows, but, as in the previous section, the algorithm can easily be extended to also handle the other kinds of problems.

Similar to the preceding section, we use a multi-level data structure, where each level caters to one aspect of the condition that a triangle crosses another triangle. Specifically, let Δ_1 and Δ_2 be two triangles, and let π_1, π_2 be the respective 2-planes that contain them. Our general position assumption allows us to assume that π_1 and π_2 always intersect at a single point ξ, and Δ_1 intersects Δ_2 if and only if ξ belongs to both triangles. Note that Δ_1 and Δ_2 intersect if and only if π_1 intersects Δ_2 and π_2 intersects Δ_1. As is easily verified, this latter pair of conditions is equivalent, with suitable orientations of π_1, π_2, and of the lines supporting the edges of both triangles, as defined earlier in this paper, to the conjunction of the following conditions:

(i) π_1 is positively oriented with respect to each of the lines that support the edges of Δ_2.
(ii) π_2 is positively oriented with respect to each of the lines that support the edges of Δ_1.

As in the preceding section, we focus here only on the case of positive orientations, as stated in the above conditions. Handling the case of negative orientations is done in a fully symmetric manner.

Conditions (i) and (ii) are the conjunction of a total of six sub-conditions, each of which tests the orientation of, say, the 2-plane π_1 with respect to the line supporting some specific edge of Δ_2, or vice versa.

We can therefore apply a suitable variant of the same primal-dual machinery of the preceding section, where at each of level of the structure we have a problem involving range searching with semi-algebraic ranges in $\mathbb{R}^6$. Note that, unlike the problem studied in Sect. 3, here all levels of the structure involves semi-algebraic

range searching, with quadratic or cubic polynomial inequalities, in six dimensions; see (3) and the discussion around it for the degrees of these polynomials. This yields a proof of Theorem 1.5.

The batched bichromatic version. We next apply Theorem 1.5 for the batched version of the triangle-triangle intersection problem. That is, we have m red triangles and n blue triangles (see Theorem 1.6), and we choose the storage parameter s to be such that the cost of m queries with the red triangles is asymptotically roughly the same as the cost of preprocessing the blue triangles. That is, we set $s = mn/s^{1/6}$ (where the right-hand side is roughly the cost of m queries, each taking $O^*(n/s^{1/6})$ time), or $s = m^{6/7}n^{6/7}$. For this choice to make sense, we need to ensure that $n \leq s \leq n^6$, or that $n^{1/6} \leq m \leq n^6$. When $m > n^6$ we only use the data structure of [5] and obtain the running time $O^*(m + n^6) = O^*(m)$, and when $m < n^{1/6}$ we only use the data structure of [26] and obtain the running time $O^*(mn^{5/6} + n) = O^*(n)$ (refer once again to Sect. 3 for more details about these data structures). Altogether we obtain the bound in Theorem 1.6.

5 Segment-Intersection amid Tetrahedra: An Improved Solution

In this section we present an improved algorithm for setup (i) of the paper, for a data structure of roughly quadratic size. This improvement applies for segment-intersection detection and reporting, but is not guaranteed for counting, because a tetrahedron intersected by the query segment may arise more than once in the output. Let $\mathcal{T}$ be a collection of n tetrahedra in $\mathbb{R}^4$. Our improved solution constructs a data structure that uses $O^*(n^2)$ storage (and expected preprocessing time), and answers a query in $O^*(n^{1/2})$ time. This is indeed a significant improvement over the standard algorithm in Sect. 3, in which, with storage $O^*(n^2)$, the query cost is $O^*(n^{2/3})$. With a suitable tradeoff, presented in Sect. 6, the improvement can be extended for any storage parameter between n and n^6, although it is most substantial when the storage is nearly quadratic; see Fig. 1.

Assume, without loss of generality, that the query segment is bounded (i.e., not a ray or a full line). The algorithm constructs a partitioning polynomial F in $\mathbb{R}^4$ of degree $O(D)$, for some large but constant parameter D, so that each of the $O(D^4)$ cells of the partition is crossed by at most n/D^2 2-faces of the tetrahedra in $\mathcal{T}$ and by a total of at most n/D tetrahedra. The existence of such a polynomial follows from Guth [20], and an expected linear-time algorithm for its construction (for constant D) is given in [5]. We classify each tetrahedron $\Delta \in \mathcal{T}$ with respect to a partition cell τ that it intersects, as being either *narrow* in τ, if a 2-face of Δ crosses τ, or *wide* otherwise (that is, Δ crosses τ but none of its 2-faces crosses τ). Let $\mathcal{N}_\tau$ (resp., $\mathcal{W}_\tau$) denote the set of narrow (resp., wide) tetrahedra at τ.

There are two cases to consider in our analysis, depending on whether the query segment ρ is contained or not contained in the zero set $Z(F)$ of F. Each of these cases requires its own data structure. The latter case is an extension of the analysis in [18] (given there for the three-dimensional version of the problem), and the case where $\rho \subset Z(F)$ requires a different approach than that taken in [18] for handling queries on the zero set. See below for full details.

5.1 A Sketch of the Analysis

A query segment ρ that is not contained in $Z(F)$ crosses at most $O(D)$ cells of the partition. For each partition cell τ (an open connected component of $\mathbb{R}^4 \setminus Z(F)$) we construct an auxiliary data structure on the wide tetrahedra at τ, and preprocess the narrow tetrahedra at τ recursively. As we show in Sect. 5.2, the structure for the wide tetrahedra uses $S_0(n) = O^*(n^2)$ storage, and a query amid them takes $Q_0(n) = O^*(n^{1/2})$ time. We then query the auxiliary structure at τ, in order to detect if such a segment-tetrahedron intersection exists. Otherwise, we detect such intersections recursively. If no intersection with a wide or a narrow tetrahedron has been found, we proceed to the next cell[8] τ' crossed by ρ, repeat the whole procedure at τ', and keep doing this till we either find a tetrahedron hit by ρ or run out of cells, and then conclude that ρ does not hit any tetrahedron of $\mathcal{T}$. The correctness of this procedure is clear (modulo that of the procedure for handling wide tetrahedra).

When the query segment ρ is contained in $Z(F)$, we apply a secondary partition over $Z(F)$, where the underlying regions are the intersections of the input tetrahedra with $Z(F)$. In this case, we apply a recursive mechanism, with a similar framework as described above. That is, we construct an auxiliary data structure for the wide tetrahedra, and recurse with the narrow tetrehedra. However, the analysis in this case requires special handling, which exploits some further algebraic properties of zero sets—see below.

Denote by $S(n)$ (resp., $Q(n)$) the maximum storage (resp., query time) required by the overall structure for n tetrahedra. Also denote by $S_1(n)$ (resp., $Q_1(n)$) the maximum storage (resp., query time) required for processing the input tetrahedra for intersection queries with segments contained in $Z(F)$, for any set of n tetrahedra in $\mathbb{R}^4$. We then have, for a suitable absolute constant $c > 0$ (where the constant hidden in the $O_D(\cdot)$ notation depends on D),

$$\begin{aligned} S(n) &= O_D(S_0(n/D)) + S_1(n) + cD^4 S(n/D^2) \\ Q(n) &= \max\left\{ O_D(Q_0(n/D)) + cDQ(n/D^2),\ Q_1(n) \right\}. \end{aligned}$$

We show, in Sect. 5.3, that $S_1(n) = O_D^*(n^2)$ and $Q_1(n) = O_D^*(n^{1/2})$. Substituting these bounds, as well as the bounds for $S_0(n)$ and $Q_0(n)$, the solutions of these recurrences are easily seen to be (for D a constant) $S(n) = O^*(n^2)$ and $Q(n) = O^*(n^{1/2})$. Modulo the missing details, to be provided in the following two subsections, this establishes Theorem 1.2.

5.2 Handling the Wide Tetrahedra

Handling the wide tetrahedra at a partition cell τ resembles, and extends to four dimensions, a similar machinery recently developed by the authors in [18]. It is done via the following secondary recursion. We first assume, without loss of generality, that

[8] The order of processing the cells during a query is important for ray-shooting queries, but is immaterial for segment intersection queries.

x_d is a *good direction* in the sense that, for any fixed $a \in \mathbb{R}^3$, we have that $F(a, x_d)$, viewed as a polynomial in x_d, has finitely many roots. We next choose some large constant parameter $r_0 \gg D$, and partition $\partial\tau$ into $O_D(1)$ $x_1x_2x_3$-*monotone strata*, that is, this is a decomposition of $\partial\tau$ into strata (also referred to as "pseudo-prisms"), where each such portion is crossed at most once by any x_4-parallel line. This is fairly standard to do, using the *cylindrical algebraic decomposition* [14, 30], or CAD for short, of F, and the resulting strata are of dimension three or lower (see [11, 14, 30] for details concerning this decomposition).

We construct, for each stratum σ, a $(1/r_0)$-cutting for the set of (constant-degree algebraic) 2-surfaces of intersection of σ with the wide tetrahedra in $\mathcal{W}_\tau$ (since the tetrahedra are wide, these are portions of hyperplanar cross-sections of $Z(F)$). The cutting is constructed by projecting σ and the 2-surfaces that it contains onto the $x_1x_2x_3$-subspace, constructing a $(1/r_0)$-cutting, within that subspace, on the projected surfaces, and then lifting the resulting cutting back to σ. Using standard results on vertical decomposition in three dimensions (see, e.g., [31]) and the theory of cuttings [22], we obtain $O^*(r_0^3)$ cells of the cutting (referred to as (pseudo-)prisms, in accordance with the way in which the vertical-decomposition–based cutting is constructed), each of which is crossed by (intersects but not contained in) at most n/r_0 wide tetrahedra. The prisms can be of any dimension ≤ 3 and are assumed to be relatively open, and are thus pairwise disjoint.

For each pair ψ_1, ψ_2 of prisms, over all possible pairs of strata, we define S_{ψ_1,ψ_2} to be the set of all segments e so that e has an endpoint in ψ_1 and an endpoint in ψ_2, and the relative interior of e is fully contained in τ. Clearly, S_{ψ_1,ψ_2} is a semi-algebraic set of constant complexity in a 6-dimensional parametric space,[9] and we decompose it into its $O(1)$ connected components.

For each segment $e \in S_{\psi_1,\psi_2}$, let $\mathcal{T}(e)$ denote the set of all wide tetrahedra Δ of $\mathcal{W}_\tau$ that e crosses. We have the following crucial technical lemma, akin to Lemma 2.2 in [18]:

Lemma 5.1 *Each connected component C of S_{ψ_1,ψ_2} can be associated with a fixed set $\mathcal{T}_C$ of wide tetrahedra Δ of $\mathcal{W}_\tau$, none of which crosses $\psi_1 \cup \psi_2$, so that, for each segment $e \in C$, $\mathcal{T}_C \subseteq \mathcal{T}(e)$, and each tetrahedron Δ in $\mathcal{T}(e) \setminus \mathcal{T}_C$ crosses $\psi_1 \cup \psi_2$.*

Proof Pick an arbitrary but fixed segment e_0 in C, and define $\mathcal{T}_C$ to consist of all the tetrahedra in $\mathcal{T}(e_0)$ that do not cross $\psi_1 \cup \psi_2$. See Fig. 3 for an illustration.

Let e be another segment in C. Since C is connected, as a set in the six-dimensional parametric space $\mathcal{F}$ of segments connecting a point on ψ_1 with a point on ψ_2, there exists a continuous path π in C that connects e_0 and e. That is, each point on π represents a segment with one endpoint on ψ_1 and the other on ψ_2, and π represents a continuous variation of such a segment (in the Hausdorff metric sense) from e_0 to e. Let Δ be a tetrahedron in $\mathcal{T}(e_0)$ that does not cross $\psi_1 \cup \psi_2$ (that is, $\Delta \in \mathcal{T}_C$). For a segment $e' \in \pi$, define the point $q_\Delta(e')$ to be the unique point $e' \cap \Delta$. ($q_\Delta(e')$ is indeed unique, if it exists, unless e' gets to be contained in or partially overlap Δ, a situation that we will shortly rule out.) As e' starts traversing π from e_0 to e, the point

[9] Each segment is specified by its two endpoints; since they lie on $\partial\tau$, each has three degrees of freedom.

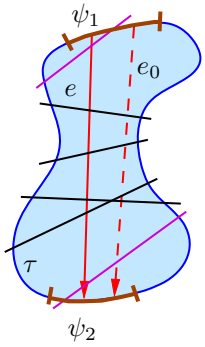

Fig. 3 The set $\mathcal{T}_C$ (consisting of the tetrahedra depicted as black segments), and an illustration of the proof of Lemma 5.1: The tetrahedra that cross some fixed segment e_0 between ψ_1 and ψ_2 are the same tetrahedra that cross any other such segment e, except for those that cross ψ_1 or ψ_2 (like those depicted as magenta segments)

$q_\Delta(e')$ is well defined and varies continuously in $\tau \cap \Delta$, until we reach an instance at which either (i) the relative interior of e' touches $\partial\Delta$, or (ii) an endpoint of e' touches Δ, or (iii) e' comes to overlap Δ in an interval with a nonempty interior.

Case (i) cannot arise because the relative interior of e' is fully contained in τ and Δ is wide at τ. Case (ii) also cannot arise because then Δ would have to intersect either ψ_1 or ψ_2 (since, by assumption, the endpoints of e' lie in $\psi_1 \cup \psi_2$), which we have assumed not to be the case. Case (iii) is also impossible, because it implies that either Case (i) or Case (ii) must also arise, which cannot happen as just argued.

To recap, as e' varies along π, it keeps intersecting Δ for every tetrahedron $\Delta \in \mathcal{T}_C$. Thus the endpoint e of π is also a segment that crosses Δ, and this establishes the first assertion of the lemma.

We next need to show that each tetrahedron in $\mathcal{T}(e) \setminus \mathcal{T}_C$ must cross either ψ_1 or ψ_2 (or both), which is our second assertion. Let Δ be a tetrahedron in $\mathcal{T}(e) \setminus \mathcal{T}_C$, and assume to the contrary that Δ does not cross $\psi_1 \cup \psi_2$. We run the preceding argument in reverse (moving from e to e_0), and observe that, by assumption and by the same argument (and notations) as above, $q_\Delta(e')$ remains well defined and inside e', for all intermediate segments e' along the connecting path π, and does not reach $\partial(\Delta \cap \tau)$, so $\Delta \in \mathcal{T}(e_0)$ and thus we have $\Delta \in \mathcal{T}_C$ (by definition of $\mathcal{T}_C$), contradicting our assumption. This establishes the second assertion, and thereby completes the proof. □

Remark We comment that the closure of ψ_1, ψ_2 may share a boundary, in this case any wide tetrahedron of $\mathcal{T}_C$ must intersect their common boundary but avoid their interiors.

The analysis for wide tetrahedra. For each prism ψ, the *conflict list* K_ψ of ψ is the set of all wide tetrahedra that cross ψ. By construction, $|K_\psi| \leq n/r_0$. The same bound for crossing tetrahedra holds when ψ is lower-dimensional. If a lower-dimensional prism is contained in some tetrahedron there is no need to process ψ further, since any segment that meets ψ hits all these tetrahedra.

Lemma 5.1 and its proof show that, for each connected component C of S_{ψ_1,ψ_2}, the set $\mathcal{T}_C$ is unique and is independent of the choice of the defining segment e.

For each pair of prisms ψ_1, ψ_2, we compute S_{ψ_1,ψ_2} and decompose it into its connected components. For each component C we compute the set $\mathcal{T}_C$ of the wide tetrahedra, as in Lemma 5.1. For this, we pick an arbitrary segment e_0 in C, compute the set $\mathcal{T}(e_0)$ as defined above, and remove from it all the tetrahedra that cross $\psi_1 \cup \psi_2$. All these operations can be implemented in $O_D(1)$, for a fixed pair ψ_1, ψ_2, in the algebraic model that we assume (see [11]), for a total of $O_D^*(r_0^6) \cdot n = O_D(n)$ storage and computation time.

Let s be the storage parameter associated with the problem; we require (and will ensure) that $n \le s \le n^3$. For each canonical set $\mathcal{T}_C$, we replace its (wide) tetrahedra by their supporting hyperplanes (recall our comment above that this set is unique), and preprocess the resulting collection of hyperplanes for efficient segment intersection queries amid hyperplanes in $\mathbb{R}^4$. Using the technique of Agarwal and Matoušek [7], this problem can be solved using $O^*(s)$ storage (and preprocessing), and a query takes $O(n \text{ polylog}(n)/s^{1/4}) = O^*(n/s^{1/4})$ time (see also [3]). Lemma 5.1 guarantees the correctness of this procedure, namely, that replacing each tetrahedron in $\mathcal{T}_C$ by its supporting hyperplane does not cause any "false positive" answer. This is because, with the exception of the tetrahedra that cross $\psi_1 \cup \psi_2$, a segment in S_{ψ_1,ψ_2} crosses all the tetrahedra in $\mathcal{T}_C$. Hence a subsegment crosses a tetrahedron in $\mathcal{T}_C$ if an only if it crosses its supporting hyperplane.

We now process recursively each conflict list K_ψ, over all prisms ψ of the partition of $\partial\tau$. Each recursive subproblem uses the same parameter r_0, but the allocated storage parameter is now set to s/r_0^3. Since the number of subproblems is $O^*(r_0^3)$, this allocation guarantees that the overall storage, over all recursive steps, remains $O^*(s)$. We keep recursing until we reach conflict lists of size close to $n^{3/2}/s^{1/2}$. More precisely, using the explicit bound $O(r_0^{3+\varepsilon})$ for the number of cells in the $(1/r_0)$-cutting (as described above), after j levels of recursion, we get a total of at most $(c_0 r_0^{3+\varepsilon})^j = c_0^j r_0^{(3+\varepsilon)j}$ subproblems, each involving at most n/r_0^j wide tetrahedra, for an arbitrarily small $\varepsilon > 0$ and a constant c_0 that depends on D and ε (recall that r_0 is taken to be sufficiently large).

We stop the recursion at the first level j^* at which $n/r_0^{j^*} \le n^{3/2}/s^{1/2}$. As a result, we have ${r_0}^{j^*} = O(s^{1/2}/n^{1/2})$, and we get $c_0^{j^*} r_0^{(3+\varepsilon)j^*} = O^*(s^{3/2}/n^{3/2})$ subproblems. Each of these subproblems involves at most $n/r_0^{j^*} = O^*\left(n^{3/2}/s^{1/2}\right)$ tetrahedra. Hence the overall size of the inputs, as well as of the canonical sets, at all the subproblems throughout the recursion, is $O^*\left(\frac{s^{3/2}}{n^{3/2}} \cdot \frac{n^{3/2}}{s^{1/2}}\right) = O^*(s)$. In particular, this is the asymptotic cost at the bottom level of the recursion.

As just described, at the bottom of the recursion, each subproblem contains at most $O^*(n^{3/2}/s^{1/2})$ wide tetrahedra, and we detect intersections with them by brute force. We thus obtain the following recurrence for the overall storage $S_0(N_W, s_W)$ for the structure constructed on N_W wide tetrahedra, where s_W denotes the storage parameter allocated to the structure (at the root $N_W = n$, $s_W = s$). The overhead term in the first inequality is

$$O_D^*(r_0^6 s_W) + O_D^*(r_0^6) N_W = O^*(s_W),$$

due to the cost of processing all $O^*(r_0^6)$ pairs of prisms. This term also depends on D, and we choose r_0 sufficiently large with respect to D, to hide this dependence in the $O^*(\cdot)$ notation. That is, we have

$$S_0(N_W, s_W) = \left\{ \begin{array}{ll} O^*(s_W) + c_0 r_0^{3+\varepsilon} S_0\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^3}\right) & \text{for } N_W \geq \Theta^*(n^{3/2}/s^{1/2}), \\ O(N_W) & \text{for } N_W < \Theta^*(n^{3/2}/s^{1/2}). \end{array} \right\}$$

Unfolding the recurrence up to the terminal level j^*, where $N_W = O^*(n^{3/2}/s^{1/2})$, and recall that r_0 is a large constant ($r_0 \gg D$), the sum of the nonrecursive overhead terms, over all nodes at a fixed level j, is

$$c_0^j r_0^{(3+\varepsilon)j} \cdot O^*\left(\frac{s_W}{r_0^{3j}}\right) = O^*\left(s_W\right).$$

Hence, starting the recurrence at $(N_W, s_w) = (n, s)$, the overall contribution of the overhead terms is $O^*(s)$. We showed above that this is also the asymptotic cost at the bottom of the recurrence. Therefore, the overall storage used by the data structure is $O^*(s)$. Using similar considerations, one can show that the overall expected preprocessing time is $O^*(s)$ as well, since the time obeys a similar asymptotic recurrence.

Answering a query. Given a query segment ρ, which is not contained in $Z(F)$, we find its $O(D)$ intersections with $Z(F)$, which decompose it into $O(D)$ subsegments, each fully contained in some partition cell. Moreover, except for the first and last subsegments, the endpoints of each of the other subsegments lie on the boundary of its cell. We process the subsegments in their order[10] along ρ. Let e be the currently processed subsegment. If e is not the first or last subsegment, we find the prisms ψ_1, ψ_2 that contain its endpoints, and find the component C of S_{ψ_1,ψ_2} that contains e. If e is the first or last subsegment, we extend it backwards or forwards, respectively, till the first time it meets the boundary of its cell, and call the resulting subsegment e'. We now compute for e' the corresponding set S_{ψ_1,ψ_2} and its component C that contains e'. Since D and r_0 are constants, all this takes constant time.

The query, on the wide tetrahedra at the present cell τ, proceeds as follows. If the present subsegment e is not the first or the last subsegment, we know, from Lemma 5.1, that it crosses all the tetrahedra of T_C, so we return a positive answer if this set is nonempty. If e is the first or the last subsegment, we perform a segment intersection detection query with e in the set of hyperplanes containing the tetrahedra of $\mathcal{T}_C$, and, if no intersection is detected, or if T_C is empty in the other cases, we continue recursively with the conflict lists K_{ψ_1} and K_{ψ_2} (at the bottom of recursion we apply a brute-force search). If no tetrahedron is found, in all the r_0-recursive steps, we conclude that (the present subsegment of) ρ does not hit any wide tetrahedron within τ. Once again, the correctness of this procedure follows from Lemma 5.1.

[10] As already mentioned, the order is immaterial for segment intersection detection queries, but is important for ray shooting.

Within a fixed cell τ, that has N_W wide tetrahedra and has storage parameter s_W, the query time $Q_0(N_W, s_W)$ on these wide tetrahedra satisfies the recurrence

$$Q_0(N_W, s_W) = \left\{ \begin{array}{ll} O_D(1) + O^*\left(\frac{N_W}{s_W^{1/4}}\right) + 2Q\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^3}\right) & \text{for } N_W \geq \Theta^*(n^{3/2}/s^{1/2}), \\ O(N_W) & \text{for } N_W < \Theta^*(n^{3/2}/s^{1/2}). \end{array} \right\}$$

Unfolding the recurrence, we see that when we pass from some recursive level to the next one, we get two descendant subproblems from each recursive instance, and the term $\frac{N_W}{s_W^{1/4}}$ is replaced in each of them by the term

$$\frac{N_W/r_0}{(s_W/r_0^3)^{1/4}} = \frac{N_W}{s_W^{1/4} r_0^{1/4}}.$$

Hence the overall bound for the nonrecursive overhead terms in the unfolding, starting from $(N_W, s_W) = (n, s)$, is at most

$$O^*\left(\sum_{j \geq 0} \left(\frac{2}{r_0^{1/4}}\right)^j \cdot \frac{n}{s^{1/4}}\right) = O^*\left(\frac{n}{s^{1/4}}\right).$$

(The sum of this geometric sequence is just a small constant that depends on r_0.) Adding the cost at the (at most) 2^{j^*} subproblems at the bottom level j^* of the recursion that the query reaches, where the cost of each subproblem is at most $O^*(n^{3/2}/s^{1/2})$, we obtain the query time

$$Q_0(n, s) = O^*\left(\frac{n}{s^{1/4}} + \frac{n^{3/2}}{s^{1/2}}\right). \tag{4}$$

Therefore, for $s = n^2$ the query time is $O^*(n^{1/2})$. The bounds $S_0(n) := S_0(n, n^2) = O^*(n^2)$ and $Q_0(n) := Q_0(n, n^2) = O^*(n^{1/2})$ are the bounds promised earlier for the wide tetrahedra at a cell.

5.3 Query Segments on $Z(F)$

Consider next segments ρ that are contained in $Z(F)$. Without loss of generality we may assume that F is irreducible; otherwise we apply the forthcoming machinery separately to each irreducible factor of F. (Decomposing F into its irreducible factors, over the reals, can be done in $O_D(1)$ time in the real RAM algebraic model that we are using; see [11, 15, 19, 23] for the relevant literature.) We may also assume that $Z(F)$ is not a hyperplane. If it is, we simply face an intersection detection problem in three dimensions amid a collection of triangles (each tetrahedron crosses $Z(F)$ in a convex polytope of constant complexity, and we replace it by its triangulated boundary, ignoring the easy-to-handle special case where the query is fully contained inside such a polytope). This latter task has been studied in [18], where a solution with better performance bounds has been given.

We partition $Z(F)$ into $O_D(1)$ $x_1x_2x_3$-monotone strata, as we did in the algorithm for wide tetrahedra. These strata cover $Z(F)$ for a generic choice of the coordinate frame. Each tetrahedron $\Delta \in \mathcal{T}$ intersects $Z(F)$ in a semi-algebraic set Δ_F of constant complexity (that depends on D), and we distribute Δ_F among all the strata that it intersects, where each stratum inherits the portion of Δ_F clipped to that stratum. We project each stratum σ, and the portions of the sets Δ_F that it contains, onto the $x_1x_2x_3$-space. For a stratum σ and a tetrahedron Δ that crosses σ, we denote the $x_1x_2x_3$-projection of $\Delta \cap \sigma$ (i.e., $\Delta_F \cap \sigma$), which is also a semi-algebraic set of constant complexity, as K_Δ. We also denote by B_Δ the $x_1x_2x_3$-projection of the intersection of $\partial\Delta$ with σ; note that B_Δ is the union of up to four subsets, each of which is the intersection of a different 2-face of Δ with $Z(F)$. Excluding degenerate scenarios, which are mentioned later, each K_Δ is at most two-dimensional, and each B_Δ is at most one-dimensional. Indeed, $\Delta \cap Z(F)$ is two-dimensional (unless Δ is fully contained in $Z(F)$), and each of the four subsets of B_Δ is contained in the intersection of a 2-plane with $Z(F)$, which is a constant-degree algebraic curve (unless this 2-plane is fully contained in $Z(F)$). Handling the degenerate cases, where Δ or one of its 2-faces is fully contained in $Z(F)$, is easier. The former situation can arise only when $Z(F)$ is a hyperplane, which we have assumed not to be the case. In the latter case, we simply collect all these facets and add them to the two-dimensional surfaces $Z(F)$, to which we apply the procedure described shortly.

We thus face the problem of segment intersection detection in three dimensions (the query segment projects to a segment in 3-space) amid a collection $\mathcal{K}$ of n two-dimensional semi-algebraic sets of constant complexity. We are not aware of an efficient solution to this problem. (A standard solution that maps the problem to semi-algebraic range searching in a higher-dimensional parametric space, results in a much less efficient solution.) We obtain an efficient procedure by exploiting several special properties of our setting. Specifically, we exploit two constraints on the problem:

(a) The sets in $\mathcal{K}$ have a special structure—each of them is the $x_1x_2x_3$-projection of the intersection of a tetrahedron with $Z(F)$.

(b) The query segments also have a special structure—each such segment is the $x_1x_2x_3$-projection of a segment supported by a line that is fully contained in $Z(F)$.

To exploit property (b), let $\mathcal{L}$ denote the set of all lines that are fully contained in $Z(F)$, and let $\mathcal{L}^*$ denote the set of the $x_1x_2x_3$-projections of these lines. We claim that, since $Z(F)$ is not a hyperplane, $\mathcal{L}^*$ cannot be the set of all lines in the $x_1x_2x_3$-space (this property does not hold when $Z(F)$ is a hyperplane). Indeed, take some generic smooth point $w \in Z(F)$, let w^* denote its $x_1x_2x_3$-projection, and let π_w denote the tangent hyperplane to $Z(F)$ at w. By assumption, all the lines in the $x_1x_2x_3$-space that pass through w^* are in $\mathcal{L}^*$, so each of them has a lifted image in $\mathbb{R}^4$ that is contained in $Z(F)$, and also in π_w. This is easily seen to imply that the entire hyperplane π_w is contained in $Z(F)$, which is impossible since F is irreducible and $Z(F)$ is not a hyperplane.

Since F is of constant degree, standard arguments in real algebraic geometry imply that $\mathcal{L}^*$ is a semi-algebraic set of constant complexity, which is not the entire 3-space; see, e.g., [11].

We tackle our problem by processing each stratum σ in turn. We construct a (second) trivariate partitioning polynomial G, of degree $O(D_1)$, where $D_1 \gg D$ is another constant parameter, so that each cell of $\mathbb{R}^3 \setminus Z(G)$ is crossed by at most n/D_1^2 one-dimensional curves B_Δ, and by at most n/D_1 two-dimensional sets K_Δ.

A query segment ρ contained in $Z(F)$ is projected to a segment ρ^* in $\mathbb{R}^3$ (the $x_1x_2x_3$-space), whose supporting line belongs to $\mathcal{L}^*$. Two cases can arise:

ρ^* is not contained in $Z(G)$. We say that a tetrahedron $\Delta \in \mathcal{T}$ is *narrow* at a cell τ of the partition induced by G if B_Δ crosses τ, and Δ is *wide* at τ if K_Δ crosses τ but B_Δ does not. As in the four-dimensional case, we denote by $\mathcal{W}_\tau$ (resp., by $\mathcal{N}_\tau$) the set of wide (resp., narrow) tetrahedra at τ. We preprocess the wide tetrahedra at τ using a special substructure, and handle the narrow tetrahedra recursively.

Handling the wide tetrahedra. The analysis is similar to that for wide tetrahedra in four dimensions, as presented in Sect. 5.2, but we spell it in detail, risking repetition of some of the arguments, as the actual technical details are different in the current setup.

Let τ be a cell of $\mathbb{R}^3 \setminus Z(G)$. Using properties of planar cuttings [22], and slightly abusing the notation of r_0, we partition $\partial\tau$ into $O^*(r_0^2)$ pseudo-trapezoids (trapezoids for short), for some suitable constant parameter $r_0 \gg D_1$, so that each trapezoid is crossed by at most $|\mathcal{W}_\tau|/r_0$ regions K_Δ, for $\Delta \in \mathcal{W}_\tau$. For each pair ψ_1, ψ_2 of trapezoids, we define S_{ψ_1,ψ_2} to be the set of all segments e so that (a) e has an endpoint in ψ_1 and an endpoint in ψ_2, (b) the relative interior of e is fully contained in τ, and (c) the line supporting e belongs to $\mathcal{L}^*$. Clearly, S_{ψ_1,ψ_2} is a semi-algebraic set of constant complexity (in a 4-dimensional parametric space), as each of the conditions (a)–(c) can be expressed as a semi-algebraic predicate of constant complexity, possibly using quantifiers (which can then be eliminated [15]). We decompose S_{ψ_1,ψ_2} into its $O(1)$ connected components.

For each segment $e \in S_{\psi_1,\psi_2}$, let $\mathcal{T}(e)$ denote the set of all wide tetrahedra Δ of $\mathcal{W}_\tau$ such that e crosses their associated sets K_Δ. As in the four-dimensional case, our technique depends on the following crucial technical lemma. (Intuitively, in four dimensions, the intersections of the wide tetrahedra with $Z(F)$ have the crucial property, which is needed in the proof, that any segment on $Z(F)$ meets each of them only once. It is used in tracking down this point as we vary the segment continuously; see the third paragraph of the proof. (We exploited a similar property in the proof of Lemma 5.1. This property, however, does not necessarily hold in the three-dimensional projection, but, as we argue below, this does not hurt the analysis.)

Lemma 5.2 *Each connected component C of S_{ψ_1,ψ_2} can be associated with a fixed set $\mathcal{T}_C$ of wide tetrahedra Δ of $\mathcal{W}_\tau$, none of whose associated sets K_Δ crosses $\psi_1 \cup \psi_2$, so that, for each segment $e \in C$, $\mathcal{T}_C \subseteq \mathcal{T}(e)$, and for each tetrahedron Δ in $\mathcal{T}(e) \setminus \mathcal{T}_C$, K_Δ crosses $\psi_1 \cup \psi_2$.*

Proof Pick an arbitrary but fixed segment e_0 in C, and define $\mathcal{T}_C$ to consist of all the tetrahedra in $\mathcal{T}(e_0)$ that do not cross $\psi_1 \cup \psi_2$. See Fig. 4 for an illustration.

Let e be another segment in C. The set S_{ψ_1,ψ_2} has four degrees of freedom, two for representing the endpoint of a segment e in S_{ψ_1,ψ_2} that lies on ψ_1, and two for the other endpoint (on ψ_2). Since C is connected, as a subset of S_{ψ_1,ψ_2}, there exists a continuous path π in C that connects e_0 and e. (As in the four-dimensional case, each

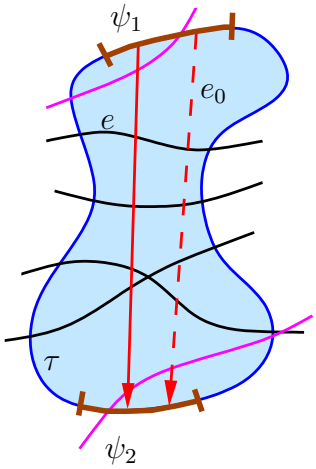

Fig. 4 The set $\mathcal{T}_C$ (consisting of the tetrahedra Δ whose associated sets K_Δ are depicted as black arcs), and an illustration of the proof of Lemma 5.2

point on π represents a segment with one endpoint on ψ_1 and the other on ψ_2, which is contained in a line of $\mathcal{L}^*$ and is fully contained in the interior of τ, and π represents a continuous variation of such a segment from e_0 to e.)

Let Δ be a tetrahedron in $\mathcal{T}(e_0)$ such that K_Δ does not cross $\psi_1 \cup \psi_2$ (that is, $\Delta \in \mathcal{T}_C$). For e_0, the intersection $e_0 \cap K_\Delta$, if nonempty, contains the projection of the single intersection point of the pre-image of e_0 with Δ (and might also contain additional points). We denote this point as $q_\Delta(e_0)$. As e' varies along π from e_0 towards e, the corresponding point $q_\Delta(e')$ is well defined and varies continuously in τ, until we reach an instance at which either (i) the relative interior of e' touches ∂K_Δ, or (ii) e' becomes tangent to K_Δ, or (iii) an endpoint of e' touches K_Δ, or (iv) e' comes to overlap K_Δ in an interval with a nonempty interior.

Case (i) cannot arise because the relative interior of e' is fully contained in τ and K_Δ is wide at τ. Case (iii) cannot arise because then K_Δ would have to intersect either ψ_1 or ψ_2, which we have assumed not to be the case. Case (ii) can occur, but then, assuming that Cases (i) and (iii) do not occur at the same time, the line ℓ in $\mathbb{R}^4$, which is contained in $Z(F)$ and projects to the line supporting e', continues to cross Δ, and its intersection point with Δ continues to project to a point in K_Δ (because $\ell \subset Z(F)$). That is, as we continue to vary e' further towards e, the line ℓ changes continuously and keeps crossing Δ (because Case (iii) does not arise), and thus e' also keeps crossing K_Δ (because Case (i) does not arise). That is, the instantaneous tangency does not cause $q_\Delta(e')$ to disappear, or to experience any jump discontinuity. In an instance of Case (iv), which is not an instance of Case (i), (ii), or (iii), the line ℓ must be fully contained in the hyperplane supporting Δ, so the projection of ℓ cuts K_Δ in a connected segment, from which it easily follows that one of Cases (i), (iii) must arise for e', a contradiction that takes care of this case too.

To recap, as e' varies along π, it keeps intersecting K_Δ for every tetrahedron $\Delta \in \mathcal{T}_C$. Thus the endpoint e of π is also a segment that crosses K_Δ, and this establishes the first assertion of the lemma.

We next need to show that, for each tetrahedron $\Delta \in \mathcal{T}(e) \setminus \mathcal{T}_C$, the associated set K_Δ must cross either ψ_1 or ψ_2 (or both), which is our second assertion. Let Δ be a tetrahedron in $\mathcal{T}(e) \setminus \mathcal{T}_C$, and assume to the contrary that K_Δ does not cross $\psi_1 \cup \psi_2$. We run the preceding argument in reverse (moving from e to e_0), and observe that, by assumption and by the same argument (and notations) as above, $q_\Delta(e')$ remains inside e', for all intermediate segments e' along the connecting path π, and does not reach $\partial K_\Delta \cap \tau$, so $\Delta \in \mathcal{T}(e_0)$ and thus we have $\Delta \in \mathcal{T}_C$ (by definition of $\mathcal{T}_C$), contradicting our assumption. This establishes the second assertion, and thereby completes the proof. □

For each pair of trapezoids ψ_1, ψ_2, and each connected component C of S_{ψ_1,ψ_2}, we take the set $\mathcal{T}_C$ of tetrahedra (back in $\mathbb{R}^4$), replace each $\Delta \in \mathcal{T}_C$ by its supporting hyperplane, and preprocess the resulting collection of hyperplanes for efficient segment intersection detection amid hyperplanes in $\mathbb{R}^4$. Using the technique of [7], this can be done, with $O^*(s)$ storage and preprocessing, with query time $O^*(n/s^{1/4})$. Choosing $s = n^2$, the storage complexity is $O^*(n^2)$ and the query time is $O^*(n^{1/2})$. Lemma 5.2 guarantees the correctness of this procedure (namely, of replacing each tetrahedron by its supporting hyperplane).

We then preprocess recursively each of the sets $\mathcal{T}_\psi$, of the tetrahedra Δ for which K_Δ crosses ψ, over all trapezoids ψ of the partition of $\partial\tau$. A query, with a segment ρ that is contained in $Z(F)$ but its projection ρ^* is not contained in $Z(G)$, is then processed as follows. As in the four-dimensional setup, we need a special treatment for the first and last subsegments of ρ, but we omit here the straightforward details, which are similar to those in the preceding analysis. We first perform a segment intersection detection query in the set of hyperplanes of the tetrahedra in $\mathcal{T}_C$, for the suitable component C that contains the intersection segment of ρ^* and τ, and then continue recursively with $\mathcal{T}_{\psi_1}$ and $\mathcal{T}_{\psi_2}$, where ψ_1 and ψ_2 are the trapezoids that contain the endpoints of the segment. We stop the recursion at nodes ψ for which $|\mathcal{T}_\psi|$ becomes roughly n^2/s. If no intersection with any wide tetrahedron has been detected, we query recursively the set of narrow tetrahedra at τ. If no tetrahedron is found to intersect the present portion of ρ within τ, we proceed to the next cell τ' crossed by the projected segment, and keep doing this until we either find a tetrahedron intersected by ρ, or run out of cells, and then conclude that ρ does not intersect any tetrahedron of $\mathcal{T}$.

The correctness of this procedure is clear. We next present the storage and the query cost for the wide tetrahedra at a cell. We then conclude this discussion with the analysis for the narrow tetrahedra, where we show the bounds on $S_1(n)$ and $Q_1(n)$ introduced in Sect. 5.1.

For the recurrence on the wide tetrahedra, denote by $S'_0(N_W, s_W)$ the maximum storage required by the structure for N_W wide tetrahedra, where s_W is the storage parameter allocated to the structure. Similarly to the analysis of the four-dimensional setup, the allocated storage parameter for each subproblem is set to be s_W/r_0^2 (since the overall number of subproblems is now $O^*(r_0^2)$). Each step of the recurrence requires a cost of $O^*_{D_1}(r_0^4 s_W)$ for the amount of storage allocated for each pair of trapezoids

ψ_1, ψ_2, as described above. We then have

$$S'_0(N_W, s_W) = \left\{ \begin{array}{ll} O^*_{D_1}(r_0^4 s_W) + c_0 r_0^2 S'_0\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^2}\right) & \text{for } N_W \geq \Theta^*(n^2/s), \\ O(N_W) & \text{for } N_W < \Theta^*(n^2/s). \end{array} \right\}$$

The constant c_0 depends on D_1, but is considerably smaller than r_0 (that is, we choose r_0 to be considerably larger). We also comment that throughout this recursion $N_W \leq s_W \leq N_W^2$. The terminal level j^* of the recurrence satisfies ${r_0}^{j^*} \leq s/n$. It is then easily checked that the total contribution of all the overhead terms, as well as the terms at the bottom of the recurrence, is $O^*(s)$, where $n \leq s \leq n^2$. Therefore the overall storage used by the data structure is $O^*(s)$.

Concerning the query time, denote by $Q'_0(N_W, s_W)$ the maximum query time required by the structure for N_W wide tetrahedra. We then have:

$$Q'_0(N_W, s_W) = \left\{ \begin{array}{ll} O_{D_1}(1) + O^*\left(\frac{N_W}{s_W^{1/4}}\right) + 2Q\left(\frac{N_W}{r_0}, \frac{s_W}{r_0^2}\right) & \text{for } N_W \geq \Theta^*(n^2/s), \\ O(N_W) & \text{for } N_W < \Theta^*(n^2/s). \end{array} \right\}$$

We note that, similarly to the four-dimensional setup, when we pass from a recursive level to the following one, we get two descendant subproblems from each recursive instance, one for each of the trapezoids ψ_1, ψ_2, and the term $\frac{N_W}{s_W^{1/4}}$ is replaced in each of them by the term

$$\frac{N_W/r_0}{(s_W/r_0^2)^{1/4}} = \frac{N_W}{s_W^{1/4} r_0^{1/2}}.$$

Hence, unfolding the recurrence, the overall bound for the nonrecursive overhead terms in the unfolding (up to the bottom level j^*), starting from $(N_W, s_W) = (n, s)$, is at most:

$$O^*\left(\sum_{j \geq 0} \left(\frac{2}{r_0^{1/2}}\right)^j \cdot \frac{n}{s^{1/4}}\right) = O^*\left(\frac{n}{s^{1/4}}\right).$$

The cost at the bottom level j^* of the recursion is at most $O^*(n^2/s)$ (by the choice of j^*). This yields an overall bound for the query time of

$$Q'_0(n, s) = O^*\left(\frac{n}{s^{1/4}} + \frac{n^2}{s}\right). \tag{5}$$

We thus obtain $S'_0(n) := S_0(n, n^2) = O^*(n^2)$ and $Q'_0(n) := Q_0(n, n^2) = O^*(n^{1/2})$ for the overall storage and query cost of this subprocedure.

The analysis for narrow tetrahedra. For the recurrence, on narrow tetrahedra, recall our notation introduced in Sect. 5.1, that is, $S_1(n)$ (resp., $Q_1(n)$) is the maximum

storage (resp., query time) required by the structure for n tetrahedra. We then have

$$S_1(n) = O_{D_1}(S_0'(n/D_1)) + S_2(n) + O({D_1}^3)S_1(n/{D_1}^2),$$
$$Q_1(n) = \max\Big\{O_{D_1}(Q_0'(n/D_1)) + (D_1+1)Q_1(n/{D_1}^2),\ \ Q_2(n)\Big\},$$

where $S_2(n)$ (resp., $Q_2(n)$) is the maximum storage (resp., query time) for segments ρ such that $\rho \subset Z(F)$ and $\rho^* \subset Z(G)$. As we show next, these quantities satisfy the bounds $S_2(n) = O^*(n^2)$ and $Q_2(n) = O^*(n^{1/2})$. With these bounds at hand, the solutions of these recurrences are $S_1(n) = O^*(n^2)$ and $Q_1(n) = O^*(n^{1/2})$.

ρ^* is contained in $Z(G)$.

It remains to handle query segments ρ^* that are contained in $Z(G)$. We may assume that $Z(G)$ is irreducible; otherwise we apply the following reasoning within each irreducible component of $Z(G)$. As $Z(G)$ is a two-dimensional algebraic surface of degree $O(D_1)$, it is either ruled (by lines) or not ruled. In the latter case, $Z(G)$ contains only $O({D_1}^2)$ lines, as implied by the Cayley–Salmon theorem [28], and we can prepare the answers to all possible queries along such lines (only for those lines that belong to $\mathcal{L}^*$). Although not a trivial step, all these lines can be computed in $O_{D_1}(1)$ time, which is constant since D_1 is constant, by solving a suitable set of equations that characterize these lines; see [28] and [11].

In the former case, $Z(G)$ is either singly ruled, or doubly ruled (a regulus), or infinitely ruled (a plane). Assume first that $Z(G)$ is singly ruled. Then (see, e.g., [21]), except for at most two exceptional lines, the lines ruling $Z(G)$ form a 1-parameter family of lines. For each tetrahedron Δ, the set of parameters of the lines whose preimages, back in $\mathbb{R}^4$, cross Δ is the union of $O_{D_1}(1)$ intervals, as is easily verified, and they can all be computed in $O_{D_1}(1)$ time. We store the $O_{D_1}(n)$ resulting intervals, obtained over all tetrahedra $\Delta \in \mathcal{T}$, in a segment tree[11]. For each node v of the tree, we take the set $\mathcal{T}_v$ of tetrahedra stored at v and preprocess the set H_v of the hyperplanes supporting these tetrahedra into a segment intersection data structure based on the machinery in [7], as in the previous steps of the algorithm. We allocate a storage parameter s to each level of the segment tree just constructed. At each node v, at any fixed level of the tree, we allocate a storage parameter that is proportional to the number of tetrahedra stored at v. Specifically, we allocate $s \cdot \frac{|H_v|}{n}$ to v. In this manner we obtain a segment-intersection data structure at v that uses $O^*(s|H_v|/n)$ storage and answers a query in time

$$O^*\left(\frac{|H_v|}{\left(s\cdot\frac{|H_v|}{n}\right)^{1/4}}\right) = O^*\left(\frac{|H_v|^{3/4}n^{1/4}}{s^{1/4}}\right) = O^*\left(\frac{n}{s^{1/4}}\right),$$

since $|H_v| \le n$. A query with a segment ρ finds the atomic (leaf) interval of the tree that contains the line supporting ρ, retrieves the $O(\log n)$ nodes on the path to that leaf, performs segment intersection queries with ρ in the sets H_v of these nodes v, and returns a tetrahedron from the output to these queries (if such a tetrahedron exists).

[11] This is a standard data structure, see, e.g., [16, Chap. 10] for details.

It is easily checked, using standard properties of segment trees (i.e., that each tetrahedron appears in at most two sets $\mathcal{T}_v$ at any level of the tree), that the overall storage used by this structure is $O^*(s)$, and that the query cost is $O^*(n/s^{1/4})$.

The case where $Z(G)$ is doubly ruled (a regulus) is handled similarly, applying the above machinery to each of the two families of ruling lines, each of which has a very simple structure.

The case where $Z(G)$ is a plane is easier to handle. In this case we do not lift the scenario back to $\mathbb{R}^4$ but instead remain on the plane $Z(G)$, and face there the problem of segment intersection detection amid a collection of n constant-complexity polygons (the intersections of the tetrahedra with the plane $Z(G)$). This can be done with $O^*(s)$ storage (and preprocessing) and $O^*(n/s^{1/2})$ query time (see, e.g., [3]).

We thus achieve in this case faster query time.

Summarizing all the above cases, we indeed obtain that for $s = n^2$, the resulting storage (and expected preprocessing time) and query time bounds are $S_2(n) = O^*(n^2)$ and $Q_2(n) = O^*(n^{1/2})$. As already noted, this implies that the solutions of the preceding recurrences for $S_1(n)$ and $Q_1(n)$ are $S_1(n) = O^*(n^2)$ and $Q_1(n) = O^*(n^{1/2})$.

Regarding the reporting procedure, since the canonical sets $\mathcal{T}_C$ that we construct are not necessarily pairwise disjoint, we need to apply some processing to the output in order to guarantee that all reported tetrahedra are distinct. Following the approach in [18], this can be done in overall time of $O(k \log k)$, where k is the output size. We comment, however, that since this output is a subset of a set of n fixed elements, we can use fast sorting algorithms in order to speed up the total query time to $O^*(n^{1/2}) + O(k)$. This can be done, e.g., by radix sort of two-digit numbers represented in base $O(\sqrt{n})$.

We thus have finally completed the proof of Theorem 1.2.

Remark Informally, the reason why we have managed to improve the solution only for Setup (i) (for segment queries) is that when the queries are triangles (in Setup (ii)) or tetrahedra (in Setup (iii)), the query object intersects too many cells of the polynomial partition, and the resulting recurrence for the query time does not yield any more efficient solution. It is an interesting open challenge to find improved solutions for these setups too. In particular, setup (iii) seems promising for such an improvement.

6 Tradeoff Between Storage and Query Time

In this section we extend the technique in Sect. 5 to obtain a tradeoff between storage (and expected preprocessing) and query time, which improves the standard tradeoff of Theorem 1.1, for any value $n < s < n^6$ of the storage parameter.

For a quick overview of our approach, consider the segment-intersection structure of Sect. 5, and let s be the storage parameter that we allocate to the structure, which now satisfies $n \leq s \leq n^6$. We modify the procedure for segment intersection inside a cell τ by (i) stopping potentially the r_0-recursion at some earlier 'premature' level, and (ii) modifying the structure at the bottom of recursion so that it uses the segment-intersection technique for hyperplanes, as discussed in Sect. 3, instead of a brute-force scanning of the tetrahedra (the current cost of $O(n^{3/2}/s^{1/2})$, a consequence of this brute-force approach, is too expensive when s is small). A similar adaptation is applied

to the recursion on the narrow triangles, as well as the procedure of segment intersection within the zero set of the partitioning polynomial. With some additional care we obtain the query time bound in (1) and the bound (2) for batched segment intersection queries, as announced in the introduction; refer also to Fig. 1.

We now present the technique in detail. Consider the segment-intersection structure of Sect. 5, and let s be the storage parameter that we allocate to the structure, which satisfies $n \leq s \leq n^6$. As before, we use this notation to indicate that the actual storage (and expected preprocessing) that the structure uses may be $O^*(s)$. We comment that in Sect. 5 s is assumed to be (at most) n^2. Handling larger values of s requires some care, detailed below. For the time being, we continue to assume that $s \leq n^2$, and will later show how to extend the analysis for larger values.

Consider first the subprocedure for handling segment intersection for segments that are not contained in the zero set of the partitioning polynomial. We run the recursive polynomial partitioning procedure described in Sect. 5 up to some 'premature' level k that we will fix later. We obtain $O^*(D^{4k})$ subproblems at the bottom level of recursion, each involving at most n/D^{2k} (narrow) tetrahedra.

Handling wide tetrahedra. Except for the bottom level, we build, at each node τ of the recursion, the same structure on the set $\mathcal{W}_\tau$ of wide tetrahedra in τ, with two (significant) differences. First, since we start the recursion on the partitioning with storage parameter s, we allocate to each subproblem, at any level j, the storage parameter s/D^{4j}, thus ensuring that the storage used by the structure is $O^*(s)$. However, the cost of a query, even at the first level of recursion, given in (4), has the term $O^*(n^{3/2}/s^{1/2})$, which is the cost of a naïve, brute-force processing of the conflict lists at the bottom instances of the r_0-recursion within the partition cells. This is fine for $s = \Omega^*(n^2)$ but kills the efficiency of the procedure when s is smaller. For example, for $s = n$ we get (near) linear query time, much more than what we aim to have. We therefore improve the performance at the bottom-level nodes of the r_0-recurrence (within a partition cell), by constructing, for each respective conflict list, the segment-intersection data structure of Sect. 3 for segment intersection amid hyperplanes in $\mathbb{R}^4$, which, for N tetrahedra and with storage parameter s, answers a query in time $O^*(N/s^{1/6})$. Since at the bottom of the r_0-recursion, both the number of tetrahedra and the storage parameter are $O^*(n^{3/2}/s^{1/2})$, the cost of a query at the bottom of the recursion is

$$O^*((n^{3/2}/s^{1/2})^{5/6}) = O^*(n^{5/4}/s^{5/12}).$$

That is, the modified (improved) cost of a query at such a node is

$$Q(n, s) = O^*\left(\frac{n}{s^{1/4}} + \frac{n^{5/4}}{s^{5/12}}\right), \tag{6}$$

where the bound $O^*(n/s^{1/4})$ is contributed by the recursion from the root, as shown in Sect. 5.

Handling the recursion on the polynomial partitions. At each of the $O^*(D^{4k})$ bottom-level cells τ, we take the set $\mathcal{N}_\tau$ of (narrow) tetrahedra that have reached τ, whose size is now at most n/D^{2k}, allocate to it the storage parameter s/D^{4k}, and

preprocess $\mathcal{N}_\tau$ using the aforementioned technique of Sect. 3, which results in a data structure, with storage parameter s/D^{4k}, which supports segment-intersection queries in time

$$O^*\left(\frac{|\mathcal{N}_\tau|}{(s/D^{4k})^{1/6}}\right) = O^*\left(\frac{n/D^{2k}}{(s/D^{4k})^{1/6}}\right) = O^*\left(\frac{n}{s^{1/6}D^{4k/3}}\right).$$

Multiplying this bound by the number $O^*(D^k)$ of cells that the query segment crosses, the cost of the query at the bottom-level cells is

$$Q_{\text{bot}}(n,s) = O^*\left(\frac{n}{s^{1/6}D^{k/3}}\right). \tag{7}$$

The cost of a query at the inner recursive nodes of some depth $j < k$ is the number, $O^*(D^j)$, of j-level cells that the segment crosses, times the cost of accessing the data structure for the wide tetrahedra at each visited cell. Since we have allocated to each of the $O^*(D^{4j})$ cells at level j the storage parameter s/D^{4j}, the cost of accessing the structure for wide tetrahedra at a j-level cell is, according to (6), at most

$$\begin{aligned} Q_{\text{inner}}(n,s) &= O^*\left(\frac{n/D^{2j}}{(s/D^{4j})^{1/4}} + \left(\frac{(n/D^{2j})^{3/2}}{(s/D^{4j})^{1/2}}\right)^{5/6}\right) \\ &= O\left(\frac{n}{D^j s^{1/4}} + \frac{n^{5/4}}{D^{5j/6}s^{5/12}}\right). \end{aligned}$$

Summing this bound over all j-level cells, for all j, and then adding the bottom-level cost from (7), and the cost of traversing the structure with the query segment (which is proportional to the number of cells intersected by the query segment), the overall cost of a query is (we remind the reader that so far we only consider the case where $s \le n^2$):

$$O^*\left(D^k + \frac{n^{5/4}D^{k/6}}{s^{5/12}} + \frac{n}{s^{1/4}} + \frac{n}{s^{1/6}D^{k/3}}\right). \tag{8}$$

We choose k to (roughly) balance the second and the last terms; specifically, we choose

$$D^k = \sqrt{\frac{s}{n}}.$$

Since D^k should not exceed $O^*(n^{1/2})$, we require for this choice of k that $s = O^*(n^2)$, which is what we are assuming so far. In this case it is easily verified that the second and last terms, which are $O^*(n^{7/6}/s^{1/3})$, dominate both the first and third terms (recall that we assume $s \ge n$), and the query time is therefore

$$O^*(n^{7/6}/s^{1/3}).$$

For larger values of s, that is, when $s = \Omega^*(n^2)$ (but we still assume $s \leq n^3$), we balance the first term with the last term, so we choose

$$D^k = O^*\left(\frac{n^{3/4}}{s^{1/8}}\right).$$

Note that in this range we indeed have that $D^k = O^*(n^{1/2})$. Moreover, in this case the first and last terms dominate the second and third terms, as is easily verified. Therefore the query time is

$$O^*(n^{3/4}/s^{1/8}).$$

As already promised, the case where the query segment lies on the zero set in the current subproblem will be presented later.

Handling the range $n^3 < s \leq n^6$. It remains to handle the range $n^3 < s \leq n^6$. Informally, at each cell τ of the polynomial partition, at any level j of the D-recursion, we have $n_\tau \leq n/D^{2j}$ wide tetrahedra and storage parameter $s_\tau = s/D^{3j}$. Since $s \geq n^3$, we also have $s_\tau \geq n_\tau^3$. With such 'abundance' of storage, we run the r_0-recursion until we reach subproblems of constant size, in which case we simply store the list of wide tetrahedra at each bottom-level node, and the query simply inspects all of them, at a constant cost per subproblem. Hence the cost of a query at τ is $O^*(n_\tau/s_\tau^{1/4})$. To be precise, this is the case as long as $s_\tau \leq n_\tau^4$. If $n^3 \leq s \leq n^4$ there will be some level j of the D-recursion at whose cells τ $s_\tau = s/D^{4j}$ becomes larger than $(n/D^{2j})^4 \geq n_\tau^4$, and then the cost becomes $O^*(1)$. When $n^4 < s \leq n^6$ the cost becomes $O^*(1)$ right away (and stays so). That is, the cost of a query in the structure for wide tetrahedra at a cell τ at level j is

$$O^*\left(\frac{(n/D^{2j})}{(s/D^{4j})^{1/4}}\right) = O^*\left(\frac{n}{s^{1/4}D^j}\right), \quad \text{for } s \leq \frac{n^4}{D^{4j}},$$
$$O^*(1), \quad \text{for } s > \frac{n^4}{D^{4j}}.$$

Since a query visits $O^*(D^j)$ cells τ at level j, the overall cost of searching amid the wide tetrahedra, over all levels, is easily seen to be

$$O^*\left(\frac{n}{s^{1/4}}\right), \quad \text{for } n^3 \leq s \leq n^4,$$
$$O^*\left(D^k\right), \quad \text{for } n^4 < s \leq n^6,$$

where k is the depth of the D-recursion.

Querying amid the narrow tetrahedra is again done as in Sect. 5 (once again, recall that we now consider the case where $s > n^3$, whereas earlier in this section we assumed $s \leq n^3$). At each node τ at the bottom level k of the D-recursion we use the data structure described in Sect. 3, which, with at most n/D^{2k} narrow tetrahedra and

storage parameter s/D^{4k}, answers a query in time

$$O^*\left(\frac{(n/D^{2k})}{(s/D^{4k})^{1/6}}\right) = O^*\left(\frac{n}{D^{4k/3}s^{1/6}}\right).$$

We multiply by the number of cells that the query visits, namely $O^*(D^k)$, and add the cost $O^*(D^k)$ of traversing these cells, for a total of

$$O^*\left(D^k + \frac{n}{s^{1/4}} + \frac{n}{D^{k/3}s^{1/6}}\right).$$

In other words, we get the same asymptotic bound as in (8), except for the second term which is missing now (this term corresponds to querying at the bottom-level nodes of the r_0-recursion on the wide tetrahedra, which is not needed when $s > n^3$, since these bottom-level subproblems now have constant size). Repeating the same analysis as above, we get the same bound $O^*(n^{3/4}/s^{1/8})$ for the query cost.

Handling the zero set. The analysis for the zero set is done similarly to the analysis presented earlier in this paper, and to the one in [18], and is quite straightforward. We do not provide a full description of these details, but only highlight the differences, from which we conclude that the query time bound is subsumed by that obtained when the query segment ρ does not lie on the zero set.

Specifically, let us consider the query time bound obtained for the wide tetrahedra in (5). This bound also subsumes the bounds obtained for the case where ρ^* is contained in the zero set $Z(G)$ of the second partitioning polynomial. The bound holds for $n \le s \le n^2$, and for larger values of s it becomes $O^*(n/s^{1/4})$, as long as $n^2 \le s \le n^4$, and $O^*(1)$ for $n^4 < s \le n^6$. We note that at every level j of the recursion on the narrow tetrahedra we allocate to each subproblem the storage parameter s/D^{3j}, and the bound on the number of (wide and narrow) tetrahedra is still $O(n/D^{2j})$. Therefore at the bottom level k we obtain an overall query time of

$$O^*\left(D^k + \frac{n^{5/3}D^{k/6}}{s^{5/6}} + \frac{n}{s^{1/4}D^{k/4}} + \frac{n}{s^{1/6}D^{3k/2}}\right).$$

This bound is subsumed by the bound in (8), for $s \ge n$, as is easily verified. Therefore adding the query time for segment intersection within $Z(f)$ does not increase the asymptotic bound in (8).

We next analyze the case where the query segment lies on the zero set. In order to obtain the trade-off bounds for segment intersection within $Z(f)$, we recall the multi-level data structure presented in Sect. 5.3. Each level in this data structure is either a one- or a two-dimensional search tree, where the dominating levels are those where we need to apply a planar decomposition over a set of planar regions (or in an arrangement of algebraic arcs) and preprocess it into a structure that supports point-location queries. A standard property of multi-level range searching data structures is that the overall complexity of their storage (resp., query time) is governed by the level with dominating storage (resp., query time) bound, up to a polylogarithmic factor [6]. Recall that in each level of our data structure we form a collection of canonical

sets of the arcs in Γ, which are passed on to the next level for further processing. Our approach is to keep forming these canonical sets, where at the very last level we apply the segment-intersection data structure of Pellegrini [27], as described above. Therefore the overall query cost (resp., storage and preprocessing complexity) is the sum of the query (resp., storage and preprocessing time) bounds over all canonical sets of arcs that the query reaches (resp., all the sets) at the last level.

We now sketch the analysis in more detail. In order to simplify the presentation, we consider one of the dominating levels, and describe the segment-intersection data structure at that level. As stated above, we build this data structure only at the very last level, but the analysis for the dominating level subsumes the bounds for the last level, and thus for the entire multi-level data structure, up to a polylogarithmic factor. In such a scenario we have a set of algebraic arcs (or graphs of functions, or semi-algebraic regions represented by their bounding arcs), which we need to preprocess for planar point location. This is done using the technique of $(1/r)$-cuttings (see [13]), which forms a decomposition of the plane into $O(r^2)$ pseudo-trapezoidal cells, each meeting at most n/r arcs (forming the "conflict list" of the cell). The overall storage complexity is thus $O(nr)$. More precisely, to achieve preprocessing time close to $O(nr)$, one needs to use so-called *hierarchical-cuttings* (see [25]) and also [9]), in which we construct a hierarchy of cuttings using a constant value r_0 as the cutting parameter, instead of the nonconstant r that we will want to use. Using this approach, both storage and preprocessing cost are $O^*(nr)$. Let s be our storage parameter as above, so we want to choose r such that $s = rn$. Thus we obtain that each cell of the cutting meets at most n^2/s arcs. Following our approach above, for each cell of the cutting, the amount of allocated storage is $s/r^2 = n^2/s$. We are now ready to apply Pellegrini's data structure, leading to a query time of $O^*\left(\frac{n^{3/2}}{s^{3/4}}\right)$. Integrating this bound into the query time in (8), we recall that at each level $0 \le j \le k$ the actual storage parameter is $O(s/D^{3j})$, and the number of tetrahedra at hand is $O(n/D^{2j})$. We now need to sum the query bound over all $O(D^j)$ cells reached by the query at the jth level, and over all j. We thus obtain an overall bound of

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

This is exactly the second term in (8). Therefore adding the query time for segment intersection within $Z(f)$ does not increase the asymptotic bound in (8).

We comment that the overall storage and preprocessing time is $O^*(s)$ (see our discussion below). We also comment that the query bound we obtained applies when $n \le s \le n^2$. When s exceeds n^2, every cell of the cutting has a conflict list of $O(1)$ elements, which the query can handle in brute-force. This immediately brings the query time, for queries on the zero set, to $O^*(1)$.

Wrapping up. In summary, our analysis implies that the query bound $Q(n, s)$ satisfies:

$$Q(n,s) = \begin{cases} O^*\left(\frac{n^{7/6}}{s^{1/3}}\right), & s = O^*(n^2), \\ O^*\left(\frac{n^{3/4}}{s^{1/8}}\right), & s = \Omega^*(n^2). \end{cases} \tag{9}$$

The overall storage (and expected preprocessing) is $O^*(s)$. Indeed, we allocate to each subproblem, at any level j, the storage parameter s/D^{4j}, so at each fixed level the total storage (and expected preprocessing) complexity is $O^*(s)$. Since there are only logarithmically many levels, the overall storage (and expected preprocessing) is $O^*(s)$ as well. This completes the proof of Theorem 1.3.

Note that for the threshold $s = n^2$, both bounds yield a query cost of $O^*(n^{1/2})$. Note also that in the extreme cases $s = n^6$, $s = n$ (extreme for the 'six-dimensional' tradeoff mentioned in Sect. 3), we get the respective bounds $O^*(1)$ and $O^*(n^{5/6})$ for the query time. In this case, when either $s = n$ or $s = n^6$ we have $D^k = O(1)$, implying that we handle all the narrow tetrahedra at the root of the recursion tree. That is, we use the technique of Sect. 3 only once. Informally, the bound in (9) 'pinches' the tradeoff curve and pushes it down. The closer s is to $\Theta(n^2)$, the more significant is the improvement. See Fig. 1.

Processing m queries. The improved tradeoff in (9) implies that the overall expected cost of processing m queries with n input tetrahedra, including (expected) preprocessing cost, is

$$O^*(s + mQ(n,s)) = \begin{cases} O^*\left(s + \frac{mn^{7/6}}{s^{1/3}}\right), & s = O^*(n^2), \\ O^*\left(s + \frac{mn^{3/4}}{s^{1/8}}\right), & s = \Omega^*(n^2). \end{cases}$$

To balance the terms in the first case we choose $s = m^{3/4}n^{7/8}$. This choice satisfies $s = O^*(n^2)$ when $m \le n^{3/2}$. To balance the terms in the second case we choose $s = m^{8/9}n^{2/3}$. This choice satisfies $s = \Omega^*(n^2)$ when $m \ge n^{3/2}$. Recall also that s has to be in the range between n and n^6. So in the first case we must have $m^{3/4}n^{7/8} \ge n$, or $m \ge n^{1/6}$. Similarly, in the second case we must have $m^{8/9}n^{2/3} \le n^6$, or $m \le n^6$. We adjust the bounds, allowing also values of m outside this range, by adding the near-linear terms $O^*(n)$ and $O^*(m)$, respectively, which dominate the bound for such off-range values of m. This establishes Corollary 1.4.

7 Output-Sensitive Construction of Arrangements of Tetrahedra and of Intersections of Polyhedra in $\mathbb{R}^4$

The results of Sect. 4 can be applied to construct the arrangement $\mathcal{A}(\mathcal{T})$ of a set $\mathcal{T}$ of n tetrahedra in $\mathbb{R}^4$ in an output-sensitive manner. A complete discrete representation of $\mathcal{A}(\mathcal{T})$ requires, at the least, the collection of all faces, of all dimensions, of the arrangement, and their adjacency structure. Concretely, for each j-dimensional face φ, for $j = 0, 1, 2, 3$, we want the set of all $(j+1)$-dimensional faces that have φ on their boundary. Conversely, for each j-dimensional face φ, for $j = 1, 2, 3, 4$, we want the set of all $(j-1)$-dimensional faces that appear on $\partial\varphi$.

We begin by considering the task of computing all the nonempty intersections of pairs, triples, and quadruples of tetrahedra of $\mathcal{T}$. This will yield the set of vertices, and provide an infrastructure for computing the j-faces, for $j = 1, 2, 3$. Denote the number of these intersections as k_2, k_3, and k_4, respectively.

Recall that we assume that the tetrahedra are in general position, although a suitable adaptation of the following machinery, using well known perturbation techniques, can handle degenerate cases too.

Reporting pairwise intersections. As noted in Sect. 1, two intersecting tetrahedra in general position in $\mathbb{R}^4$ intersect in a two-dimensional convex polygon of constant complexity, and it suffices to report one vertex of each nonempty polygon, in order to detect all intersecting pairs of tetrahedra. As is easily checked, such a vertex is either an intersection of an edge of one tetrahedron with the other tetrahedron, or an intersection of two 2-faces (triangles), one from each tetrahedron.

Reporting vertices of the first kind (edge-tetrahedron intersections) can be done using the machinery in Theorem 1.1, whose details are provided in Sect. 3, which takes $O^*(n^{12/7}) + O(k_2)$ time.[12] We comment that in order to enforce the general position assumption of the tetrahedra w.r.t. the query edges, we follow a similar approach to that described in Sect. 1 around Theorem 1.7. That is, we recursively solve the bichromatic version of this problem. The fact that input tetrahedra are in general position guarantees that when we query with a tetrahedron edge the general position assumptions stated in Sect. 3 are satisfied.

Reporting vertices of the second kind (triangle-triangle intersections) is done using the machinery in Sect. 4, which also takes $O^*(n^{12/7}) + O(k_2)$ time. Here too, the triangular faces of the tetrahedra need to satisfy the general position assumption (see once again Sect. 4). In order to enforce that we use, once again, the approach described around Theorem 1.7. That is, we color each 2-face in a distinct color, and then solve the bichromatic version of the problem (Theorem 1.6) for each pair of distinct colors.

We comment that in practice we do not return the vertices of the intersections but the corresponding pairs of intersecting tetrahedra. The machinery in Sects. 3 and 4 actually yields this data.

Reporting triple and quadruple intersections. We iterate over the input tetrahedra. For each fixed tetrahedron T_0, the previous step provides us with all the other tetrahedra that intersect T_0. Denote their number as k_{T_0}, and observe that $\sum_{T_0} k_{T_0} = 2k_2$. We form the nonempty intersections $T_0 \cap T$, and triangulate each of them. We obtain a collection of $O(k_{T_0})$ triangles, all contained in (T_0 and therefore also in) the hyperplane h_{T_0} supporting T_0.

We have thus reduced our problem to that of reporting all pairwise and triple intersections in a set of $m = O(k_{T_0})$ triangles in $\mathbb{R}^3$. This can be solved using the algorithm in [18, Corr. 5.1], by a procedure that runs in $O^*(m^{3/2}) + O(\ell_{T_0} \log \ell_{T_0})$ time, where ℓ_{T_0} is the number of triple intersections of the triangles. Note that $\sum_{T_0} \ell_{T_0} = O(k_4)$.

Adding up this cost over all tetrahedra T_0, the overall running time is

$$O^*\left(\sum_{T_0} k_{T_0}^{3/2}\right) + O\left(k_4 \log k_4\right) = O^*\left(n^{1/2} \sum_{T_0} k_{T_0}\right) + O\left(k_4 \log k_4\right)$$
$$= O^*(n^{1/2} k_2) + O(k_4 \log k_4).$$

[12] Although this part can be performed faster, as described in Sect. 5, we use the standard solution, since we do not have a similar improvement for the construction of vertices of the second kind.

Constructing the arrangement. For each tetrahedron T_0, it is fairly routine to obtain, from the information collected so far, the full three-dimensional arrangement within T_0 in additional $O(k \log k)$ time, where k is the arrangement complexity; this is done using standard techniques in three dimensions, see, e.g., [27]. This gives us all the j-faces of the four-dimensional arrangement $\mathcal{A}$, for $j = 0, 1, 2, 3$, and their adjacency information. The local adjacency information in $\mathbb{R}^4$ is also available from this data. By local adjacency we mean the adjacency between a j-face and the j'-faces on its boundary, for $j' < j$, over all such pairs of faces. For completion we need to identify disconnected pieces of the boundary of each four-dimensional cell, and record their adjacency to that cell. This can be done by x_4-vertical ray shooting from the x_4-highest point of each connected three-dimensional complex of faces. This calls for performing $O(n)$ x_4-vertical ray shooting queries in a set of n tetrahedra in $\mathbb{R}^4$, which can be done using the machinery presented in Theorem 1.1, or by an even simpler mechanism (since all the rays are vertical).

We have thus established the bound stated in Theorem 1.8.

Output-sensitive construction of the intersection or union of polyhedra in $\mathbb{R}^4$. As another application, consider the problem where we have two not necessarily convex (but bounded) polyhedra R and B in $\mathbb{R}^4$ in general position, whose boundaries consist of, or can be triangulated into $O(n)$ faces of all dimensions, which are segments, triangles, and tetrahedra. The goal is to construct their intersection $R \cap B$ in an output-sensitive manner; a similar application has been shown in [18] for the three-dimensional problem. We note that computing the union $B \cup R$ can be done using a very similar approach, within the same asymptotic time bound.

In order to compute $R \cap B$, we first apply the above algorithm to construct, in an output-sensitive manner, the arrangement $\mathcal{A}(R \cup B)$ of the two polyhedra R and B (specifically, we build the arrangement of the tetrahedra comprising the boundaries of B and R). We then label each cell (of any dimension) of $\mathcal{A}(R \cup B)$ with the appropriate Boolean operation, that is, whether it either lies in $R \setminus B$, $B \setminus R$, $B \cap R$, or in the complement of $B \cup R$. Collecting all the cells of the desired kind (e.g., those in $B \cap R$), and computing the adjacency relation between them, we obtain a suitable representation of the intersection. This establishes the bound stated in Theorem 1.8(ii).

We comment that extending the analysis to the intersection of more than two (albeit, still a constant number of) input polyhedra can also be done, following the same machinery as in the construction of an arrangement of tetrahedra, as presented above. It is easy to verify that in this case we obtain the same asymptotic bound stated in Theorem 1.8(ii).

8 Detecting or Reporting Intersections Between 2-Flats and Lines in $\mathbb{R}^4$

As a final application of our machinery, we consider the problem where we are given a set R of n red 2-flats and a set B of n blue lines in $\mathbb{R}^4$, and the detection problem asks whether there exists a pair of intersecting objects in $R \times B$. In the reporting version

we want to report all such pairs. We only consider the batched version of the problem, but a similar approach can also handle the preprocessing-and-query variant.

We solve the detection problem by regarding the problem as a special degenerate (and much simpler) instance of the segment intersection setup (and also of the triangle-triangle intersection setup), in which we regard the, say, red 2-flats as degenerate tetrahedra (unbounded and of zero volume), construct the data structure of Sect. 5, and query it with each of the blue lines. There exists a red-blue pair of intersecting objects if and only if at least one query has a positive outcome—the corresponding blue query line hits a red 2-flat. Using the bounds and notation given in Corollary 1.4, specialized to the case, under consideration here, where the input tetrahedra degenerate into 2-flats, this can be performed in expected time $\max\{O^*(m^{3/4}n^{7/8}+n),\ O^*(m^{8/9}n^{2/3}+m)\}$, and since $m = n$ in our case this bound is $O^*(m^{13/8})$. This is a clear improvement over the bound $O^*(n^{12/7})$ obtained using the initial approach presented in Sect. 3. Indeed, in this latter approach, with storage parameter s, a query takes $O^*(n/s^{1/6})$ time, and thus n queries cost $O^*(n^2/s^{1/6})$ time. Balancing these costs results in overall expected running time of $O^*(n^{12/7})$. Similar improvements are obtained for other values of m, as long as $n^{1/6} \ll m \ll n^6$ (see the analysis in the preceding section).

Since there are no wide tetrahedra in this special variant, there is no need to construct the auxiliary data structure for wide tetrahedra, as in Sect. 5, and we simply construct the recursive hierarchy of polynomial partitions, where each cell in each subproblem is associated with the set of red 2-flats that cross it. A blue query line ℓ is propagated through the cells that it crosses until it either comes to lie on the zero set of the current partitioning polynomial, or reaches bottom-level cells, and we check, in each such cell, whether ℓ intersects any of the $O(1)$ red 2-flats associated with the cell.

An easy adaptation of our machinery allows us to report all k red-blue intersecting pairs in expected time $O^*(n^{13/8}) + O(k)$.

In summary, we have:

Theorem 8.1 *Given n blue lines and n red 2-flats in $\mathbb{R}^4$, one can detect whether some blue line intersects some red 2-flat in $O^*(n^{13/8})$ expected time. One can also report all k red-blue intersections in $O^*(n^{13/8}) + O(k)$ expected time.*

We remark that this case can also be considered as a special case of the triangle-triangle intersection setup. We also note that, similar to the three-dimensional setup in [18], this technique does not support counting queries, because the subproblems that a query encounters will not in general involve pairwise disjoint sets.

Acknowledgements The authors are grateful to two anonymous referees for their comments and suggestions and for their significant investment in reading this paper.

References

1. Afshani, P., Cheng, P.: Lower bounds for intersection reporting among flat objects. In: 39th International Symposium on Computational Geometry, pp. 3:1–3:16 (2023). https://doi.org/10.4230/LIPIcs.SoCG.2023.3

2. Afshani, P., Cheng, P.: Lower bounds for semialgebraic range searching and stabbing problems. J. ACM **70**(2), 16:1-16:26 (2023). https://doi.org/10.1145/3578574
3. Agarwal, P.K.: Simplex range searching and its variants: a review. In: Journey through Discrete Mathematics: A Tribute to Jiří Matoušek, pp. 1–30. Springer, Berlin (2017)
4. Agarwal, P.K., Aronov, B., Ezra, E., Katz, M., Sharir, M.: Intersection queries for flat semi-algebraic objects in three dimensions and related problems. In: Proceedings of 38th Symposium on Computational Geometry, pp. 4:1–4:14 (2022). Also in arXiv:2203.10241
5. Agarwal, P.K., Aronov, B., Ezra, E., Zahl, J.: An efficient algorithm for generalized polynomial partitioning and its applications. SIAM J. Comput., **50**, 760–787 (2021). Also in Proceedings of Symposium on Computational Geometry (SoCG), 5:1–5:14 (2019). Also in arXiv:1812.10269
6. Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Advances in Discrete and Computational Geometry, volume 223 of Contemporary Mathematics, pp. 1–56. AMS Press, Providence, RI (1999)
7. Agarwal, P.K., Matoušek, J.: Ray shooting and parameric search. SIAM J. Comput. **22**, 794–806 (1993)
8. Agarwal, P.K., Matoušek, J., Sharir, M.: On range searching with semialgebraic sets II. SIAM J. Comput. **42**, 2039–2062 (2013). arXiv:1208.3384
9. Aronov, B., Ezra, E., Sharir, M.: Testing polynomials for vanishing on cartesian products of planar point sets: Collinearity testing and related problems. Discrete Comput. Geom., **68**, 997–1048 (2022). Also in Proceeding of 36th Symposium on Computational Geometry, 8:1–8:14 (2020). Also in arXiv:2003.09533
10. Aronov, B., Ezra, E., Zahl, J.: Constructive polynomial partitioning for algebraic curves in $\mathbb{R}^3$ with applications. SIAM J. Comput. **49**(6), 1109–1127 (2020). https://doi.org/10.1137/19M1257548
11. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry, 2nd edn. Springer, Berlin (2006)
12. Canny, J.: Collision detection for moving polyhedra. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI) **8**, 200–209 (1986)
13. Chazelle, B., Friedman, J.: A deterministic view of random sampling and its use in geometry. Combinatorica **10**(3), 229–249 (1990)
14. Collins, G.E.: Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In: Proceedings of 2nd GI Conference Automata Theory and Formal Languages, volume 33. Springer LNCS (1975)
15. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, Berlin-Heidelberg (2007) to computational algebraic geometry and commutative algebra. Springer, Berlin-Heidelberg (2007)
16. de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer (2008). https://www.worldcat.org/oclc/227584184
17. Ezra, E., Sharir, M.: Intersection searching amid tetrahedra in 4-space and efficient continuous collision detection. In: Proceedings of 30th European Symposium Algorithms, pp. 51:1–51:17 (2022). Also in arXiv:2208.06703
18. Ezra, E., Sharir, M.: On ray shooting for triangles in 3-space and related problems. SIAM J. Comput., **51**, 1065–1095 (2022). Also in Proceedings of 37th Symposium on Computational Geometry, 34:1–34:15 (2021), and in arXiv:2102.07310
19. Galligo, A., Vorobjov, N.: Complexity of finding irreducible components of a semialgebraic set. J. Complex. **11**, 174–193 (1995)
20. Guth, L.: Polynomial partitioning for a set of varieties. Math. Proc. Camb. Philos. Soc. **159**, 459–469 (2015). arXiv:1410.8871
21. Guth, L., Katz, N.H.: On the Erdös distinct distances problem in the plane. Ann. Math. **181**, 155–190 (2015). arXiv:1011.4105
22. Haussler, D., Welzl, E.: Epsilon-nets and simplex range queries. Discrete Comput. Geom. **2**, 127–151 (1987)
23. Kaltofen, E.: Polynomial factorization 1987-1991. In: Proceedings of 1st Latin American Symposium Theoretical Informatics, pp. 294–313. Lecture Notes in Computer Science, vol. 583 (1992)
24. Lin, M.C., Manocha, D., Kim, Y.J.: Collision and proximity queries. In: Handbook on Discrete and Computational Geometry, chapter 39, 3rd edn, pp. 1029–1056. CRC Press, Boca Raton (2017)
25. Matoušek, J.: Range searching with efficient hierarchical cuttings. Discrete Comput. Geom. **10**, 157–182 (1993)

26. Matoušek, J., Patáková, Z.: Multilevel polynomial partitions and simplified range searching. Discrete Comput. Geom. **54**, 22–41 (2015)
27. Pellegrini, M.: Ray shooting on triangles in 3-space. Algorithmica **9**, 471–494 (1993)
28. Salmon, G.: A Treatise on the Analytic Geometry of Three Dimensions, vol. 2, 5th edn. Hodges, Figgis and co. Ltd., Dublin (1915)
29. Schömer, E., Thiel, Ch.: Efficient collision detection for moving polyhedra. In: Proceedings of 11th Symposium on Computational Geometry, pp. 51–60 (1995)
30. Schwartz, J.T., Sharir, M.: On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. Adv. Appl. Math. **4**, 298–351 (1983)
31. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)
32. Sheffer, Adam: Polynomial Methods and Incidence Theory. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge (2022)