Triangle Nearest-Neighbor Searching in 3-Space

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

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

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

Abstract

We study various nearest-neighbor searching problems involving points, lines, segments and triangles in $\mathbb{R}^3$. Among many results, we present a linear-size data structure for answering nearest-neighbor queries with lines amid n points in $\mathbb{R}^3$, in $O^*(n^{1/2})$ time per query (where the $O^*(\cdot)$ notation hides subpolynomial factors). Our solution is based on parametric search, where the problem is reduced to range emptiness queries amid points in $\mathbb{R}^3$ with cylindrical queries. For the latter problem we show that reporting all k points lying inside a cylinder query costs an additional term of $O(k)$. We also study setups where both data and query objects are lines, segments, or triangles, and obtain improved solutions for the two extreme regimes of (near-)linear storage and of fast query time. These results also yield tradeoff bounds, where the cost of a query depends on the storage allocated to the structure. This work is a continuation of a recent work by the authors [7].

2012 ACM Subject Classification Theory of computation; Theory of computation → Computational geometry

Keywords and phrases line-point nearest neighbors, range searching, vertical decomposition in 3-space, test sets

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.34

Funding *Pankaj K. Agarwal*: Partially supported by NSF grants CCF-20-07556 and CCF-22-23870 and by a US-Israel Binational Science Foundation Grant 2022131.
Esther Ezra: Partially supported by Israel Science Foundation Grant 800/22 and US-Israel Binational Science Foundation Grant 2022131.
Micha Sharir: Partially supported by Israel Science Foundation Grant 495/23.

1 Introduction

The general *nearest-neighboring (NN) searching* problem can be formulated as follows: Given a set $\mathcal{S}$ of input objects, often referred to as *sites*, in $\mathbb{R}^d$, and a distance function ρ, the goal is to preprocess $\mathcal{S}$ into a data structure that can efficiently find the object of $\mathcal{S}$ that lies closest to a query object q under the distance function ρ. In its most classical form, referred to as the *post-office problem* [22], both the input sites and the queries are points in $\mathbb{R}^2$ and ρ is the Euclidean metric. Because of wide-ranging applications in numerous disciplines, nearest-neighbor searching has been extensively studied, for about half a century, not only in computational geometry but in many fields such as database systems, information retrieval, GIS, machine learning, computer vision, and natural language processing, and the problem has been extended in numerous ways, to cases where the sites and/or the queries may come from families of more general geometric objects, and ρ may be any general metric. Since NN searching is hard (i.e., inefficient) in high dimensions, much of the work has focused on computing an approximate nearest neighbor for d large. We refer to a few comprehensive surveys [11, 13, 27, 28] on nearest-neighbor searching and other proximity problems and their solutions.

In this paper, we focus on answering *exact* NN-queries where both input sites and query objects are either points, lines, segments, or triangles in $\mathbb{R}^3$, under the Euclidean metric $\|\cdot\|$. For a closed subset $X \subset \mathbb{R}^3$ and a point $p \in \mathbb{R}^3$, we define $\mathrm{dist}(p, X) = \min_{x \in X} \|p - x\|$, and for two closed subsets $X, Y \subset \mathbb{R}^3$, define

$$\mathrm{dist}(X, Y) = \min_{x \in X} \mathrm{dist}(x, Y) = \min_{x \in X,\ y \in Y} \mathrm{dist}(x, y).$$

For a set $\mathcal{S}$ of geometric objects and for a query object q, we define

$$\mathsf{NN}(q, \mathcal{S}) = \arg\min_{\sigma \in \mathcal{S}} \mathrm{dist}(q, \sigma),$$

that is, the nearest neighbor of q in $\mathcal{S}$. Our goal is to preprocess $\mathcal{S}$ into a data structure so that $\mathsf{NN}(q, \mathcal{S})$, for a query object q, can be computed efficiently. We refer to this task as *nearest neighbor (NN)-searching*. This general goal bifurcates into many specific variants, depending on the types of the query and of the input objects, as well as on the storage allocated for the structure; see below.

Related work. As mentioned above, NN-searching, especially when the input sites and query objects are points, has been studied extensively. It is beyond the scope of this paper to review all the related work on NN-searching, so we focus on studies relevant to this paper, involving NN-searching amid points, and also amid lines, segments, or triangles, in three (and higher) dimensions.

Consider the problem of answering NN-queries under the Euclidean metric with query lines amid points in $\mathbb{R}^3$ (which is the starting topic of our study), as well as in higher dimensions. There are several known results concerning this setting. In a recent development [7], it was shown that a set P of n points in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O^*(n^4)$, so that the nearest neighbor in P of a query line ℓ in $\mathbb{R}^3$ can be computed in $O(\log n)$ time. In fact, the analysis in [7] presents similar performance bounds when, instead of a point set P, the input consists of a set L of n lines in $\mathbb{R}^3$. Concerning linear-size data structures, we note that line-queries can be answered in $O^*(n^{2/3})$ time, using known results on 3D semi-algebraic range searching [2, 4, 26] (see also [5]), along with the parametric-search technique (see, e.g., [3, 8]). This is in fact a general approach to solving problems involving NN-searching amid geometric objects, by mapping each input object to a point in some high-dimensional parametric space, and by performing a small number of *range-emptiness* queries with appropriate semi-algebraic ranges (that is, detecting whether the query range contains any point in this transformed setting). However, such an approach often results in suboptimal bounds (more precisely, in bounds that are strongly suspected not to be optimal), mainly due to the high dimension of the parametric space.

We note that a restricted version of this problem was recently studied by Agarwal and Ezra [6], where they developed a linear-size data structure for answering line-intersection queries amid unit balls in $\mathbb{R}^3$. However, their mechanism does not fit into the parametric search framework in [8], since the data structure presented there is tailored to a prespecified distance Δ (as the radius of the balls), and does not extend to arbitrary values of Δ (that are encountered in the search). In this work we present a different approach, which is applicable to any distance parameter Δ, and eventually leads to an efficient NN-searching mechanism – see Section 3.

We also mention the converse problem, where the input objects are lines in $\mathbb{R}^3$ and the queries are points. For this setting, it was shown in [7] that the input lines can be preprocessed into a linear-size data structure, which answers NN-queries in $O^*(n^{2/3})$ time. This is achieved by a careful construction of a "test set" – see below for a discussion concerning this approach, which we also use in this work.

Our contributions. We present efficient data structures and algorithms for various proximity problems involving points and lines (or segments, or triangles) in $\mathbb{R}^3$. Furthermore, our data structures formulate NN queries in terms of semi-algebraic range emptiness queries amid a set of points, or of point-location queries in a family of semi-algebraic sets. We thus essentially develop more efficient data structures for certain instances of semi-algebraic range searching, as well as for point-location, which in turn rely on combinatorial and algorithmic results on arrangements of surfaces in $\mathbb{R}^3$ and $\mathbb{R}^4$. Of particular interest will be the recent results by the authors [7], on vertical decompositions of substructures in 3- and 4-dimensional arrangements – see Section 2 for an overview. While the work in [7] developed the underlying structural theorems, our contribution is to extend this framework to nearest-neighbor search. Many of our data structures can also answer *fixed-radius-neighbor* queries, where, for a query object q and a parameter $\Delta > 0$, we want to report all input sites that lie within distance Δ from q. Among our results, we have:

(i) Nearest point-neighbor to a query line / segment (Section 3). Let P be a set of n points in $\mathbb{R}^3$. We first show that P can be preprocessed, in $O(n \log n)$ expected time, into a linear-size data structure so that, for a query line ℓ and a radius Δ, it can determine, in $O^*(n^{1/2})$ time, whether any point of P lies within distance Δ from ℓ, or equivalently whether the cylinder of radius Δ with ℓ as its axis contains any point of P; if the answer is yes, the data structure also returns such a point. The main challenge in developing this data structure is the construction of a so-called *test set*, namely, a small set of representative semi-algebraic queries, not necessarily cylinders, so that if the data structure can answer those queries efficiently then it can answer efficiently any cylindrical range (emptiness or reporting) query in $\mathbb{R}^3$. We develop a test-set construction algorithm by following the approach in [30] and our recent results on vertical decomposition of substructures in arrangements of surfaces in $\mathbb{R}^3$ and $\mathbb{R}^4$ [7]; see Sections 2.2 and 3.2 for details. Our data structure can also be used to report all k points of P lying in a query cylinder, in $O^*(n^{1/2} + k)$ time (Theorem 4). We also present an extension of this data structure to query segments (but, unfortunately, not for query triangles), with the same performance bounds (Theorem 12).

Our approach generalizes the results in [6], which develop NN-searching machinery using parametric search. However, our data structure is significantly simpler than that presented in [6], due to the recent development involving vertical decompositions in $\mathbb{R}^3$ [7].

(ii) Nearest triangle-neighbor to a query triangle (Section 4). We consider the setup of answering an NN query with a triangle amid a set of n triangles in $\mathbb{R}^3$, which is the most general instance of the type of problems considered here. We break the problem into multiple cases, each interacting some feature of the query object with some feature of the input objects. The hardest case is answering an NN query with a segment amid a set of n line segments. We show that the decision problem in this case can be formulated as a semi-algebraic range searching in $\mathbb{R}^5$, with queries that have *reduced parametric dimension* 5 (namely, each inequality in the predicate that defines the ranges uses only at most five parameters of the query; see [5]), even though the *parametric dimension* of a segment in $\mathbb{R}^3$ (the number of real parameters needed to specify the segment) is six. We further observe that each polynomial inequality in the semi-algebraic predicate that defines the queries only involves either four parameters of the input segments or four parameters of the query segment. Using this additional observation, we obtain a slightly better performance than what one would get from a 5-dimensional semi-algebraic range-searching data structure [5]. In particular, for a storage parameter $s \in [n, n^5]$, we present a data structure of size and expected preprocessing cost $O^*(s)$, so that an NN query with a triangle can be answered in $O^*((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15})$ time (Theorem 15). Due to lack of space, some of the details concerning this result are delegated to Appendix B.

2 Preliminaries

2.1 Arrangements and vertical decompositions

Let $\mathcal{S}$ be a family of n semi-algebraic sets[1] of constant complexity in $\mathbb{R}^d$. The *arrangement* of $\mathcal{S}$, denoted by $\mathcal{A}(\mathcal{S})$, is the decomposition of $\mathbb{R}^d$ into maximal connected relatively open cells of all dimensions, so that all points within a cell lie in the relative interior of objects from a fixed subfamily of sets of $\mathcal{S}$, and on the boundary of objects from another such subfamily. The *vertical decomposition* of $\mathcal{A}(\mathcal{S})$ is a popular (and the only known general-purpose) technique for decomposing each cell of $\mathcal{A}(\mathcal{S})$ into constant-complexity subcells, each of which is homeomorphic to a ball of the appropriate dimension. These cells, referred to as "vertical pseudo-prisms" (prisms for short), are obtained by recursing on the dimension. Details on their structure can be found, e.g., in [7, 17]. Vertical decompositions for geometric arrangements and *substructures* of arrangements are useful in obtaining efficient divide-and-conquer mechanisms for solving a variety of combinatorial and algorithmic problems, as well as for constructing data structures for geometric searching problems, see, e.g., [1] and the references therein. This is also the case in the present work.

Union of semi-algebraic regions. Specifically, consider the setting where $\mathcal{S}$ is a set of n constant-complexity (closed) semi-algebraic regions in $\mathbb{R}^3$. Let $\mathcal{U}(\mathcal{S})$ denote their union and $\mathcal{C}(\mathcal{S})$ denote the (closure) of the complement of their union. For any parameter $m \leq n$, let $\psi(m)$ denote the maximum combinatorial complexity of the (undecomposed) union of a subset of $\mathcal{S}$ of size at most m. In many natural applications, $\psi(m)$ is $O^*(m^2)$ (as opposed to the general case, where $\psi(m) = O(m^3)$), such as the cases of cylinders, cubes, and "fat" objects in 3-space [10, 12, 18, 19]. The next theorem, which is crucial for the analysis in Section 3, states bounds on the complexity of the vertical decomposition of $\mathcal{C}(\mathcal{S})$ and on the overall (expected) running time to construct it:

▶ **Proposition 1** ([7]). *Let $\mathcal{S}$ and $\psi(\cdot)$ be as above. Then the total complexity of the vertical decomposition of the cells in $\mathcal{C}(\mathcal{S})$ is $O^*(n^2 + \psi(n))$. Moreover, the vertical decomposition of $\mathcal{C}(\mathcal{S})$ can be constructed in $O^*(n^2 + \psi(n))$ randomized expected time.*

Line-point proximity in 3-space. The analysis in this paper will also rely on the following proximity result reported in [7]:

▶ **Proposition 2** ([7]). *Let P be a set of n points in $\mathbb{R}^3$. Then P can be preprocessed, in $O^*(n^4)$ expected time, into a data structure of size $O^*(n^4)$, so that, for any query line $\ell \in \mathbb{R}^3$, $\mathsf{NN}(\ell, P)$ can be computed in $O^*(1)$ expected time. The same performance bounds still apply when we replace P with a set L of n lines in $\mathbb{R}^3$.*

2.2 NN-searching and test sets

Let P be a set of n points in $\mathbb{R}^d$, and let Γ be a family of semi-algebraic ranges of constant complexity. Using the parametric-search technique [8], an NN-query $\mathsf{NN}(\gamma, P)$, for a range $\gamma \in \Gamma$, can be reduced to answering $O^*(1)$ semi-algebraic range-emptiness queries with query objects of the form $\mathcal{B}(\gamma, \rho) = \{x \in \mathbb{R}^d \mid \mathrm{dist}(x, \gamma) \leq \rho\}$ for some parameter $\rho > 0$. We will thus be designing faster data structures for answering such range-emptiness queries (see, e.g., Section 3), using the ideas in [25, 30].

[1] Roughly speaking, a semi-algebraic set in $\mathbb{R}^d$ is the set of points in $\mathbb{R}^d$ that satisfy a Boolean formula over a set of polynomial inequalities; the complexity of a semi-algebraic set is measured in terms of the number of polynomials defining the set and their maximum degree.

We briefly describe the approach taken in Sharir and Shaul [30]. Roughly speaking, the technique in [30] builds a linear-size partition tree recursively using a hierarchical clustering of the input points. At each node v of the tree, we have a subset $P_v \subseteq P$ of n_v points. For a parameter $r > 1$, one constructs an *elementary-cell partition* $\Pi = \{(P_1, \tau_1), \ldots, (P_r, \tau_r)\}$ where (i) $n_v/r \leq |P_i| \leq 2n_v/r$ for each i, (ii) $P_1, \ldots, P_r$ is a partition of P_v, and (iii) each τ_i is an "elementary cell" (a semi-algebraic set of constant complexity homeomorphic to a ball) containing P_i (the cells τ_i do not have to be pairwise disjoint). The main property of Π is that any range in Γ crosses (intersects but does not fully contain) only few elementary cells of Π. A major ingredient to accomplish this is to construct a so-called *test set* of a small number of semi-algebraic sets (not necessarily ranges of Γ), which represent well all ranges of Γ (see below, and also [25]). Formally, for a parameter $r > 1$, we call a semi-algebraic set $\gamma \subset \mathbb{R}^d$ (n/r)-*shallow* if $|P \cap \gamma| \leq n/r$, and a finite collection Φ of semi-algebraic sets of constant complexity is called a *test set* for (n/r)-shallow ranges in Γ if it satisfies the following properties:

(P1) Every set in Φ is (n/r)-shallow with respect to P.

(P2) The complement of the union of any m sets of Φ can be decomposed into at most $\zeta(m)$ elementary cells for any $m \geq 1$, where $\zeta(m)$ is a suitable monotone increasing super-linear function of m.

(P3) Any (n/r)-shallow set $\gamma \in \Gamma$ can be covered by the union of at most δ ranges of Φ, where δ is a constant (independent of r).

The following lemma summarizes the main result of [30].

▶ **Lemma 3.** *Let P and Γ be as above. Suppose that a test set of size $r^{O(1)}$ for (n/r)-shallow ranges of Γ can be constructed in expected $O^*(nr^{O(1)})$ time. Then a linear-size partition tree on P can be constructed in $O^*(n)$ expected time, so that it supports Γ-emptiness queries in $O^*(n/\zeta^{-1}(n))$ time per query; all k points lying in a Γ-range can be reported in $O(k)$ additional time.*

3 Line NN Queries amid Points in $\mathbb{R}^3$

In this section we consider the setting in which the input is a set P of n points in $\mathbb{R}^3$, and the query objects are lines in $\mathbb{R}^3$. We will later extend this data structure to handle segments and triangles as query objects (see Section 3.3 and Appendix A.2). We next describe a linear-size data structure for query lines.

3.1 Line NN queries

As discussed above, using parametric search [1, 8], an NN query on P can be reduced to answering $O^*(1)$ *cylinder-emptiness* queries on P. That is, we want to preprocess P into a data structure of near-linear size that can efficiently determine whether a query cylinder C (of arbitrary radius) contains a point of P, and if the answer is Yes, return such a point. For simplicity, we assume that the axis of the query cylinder C is not parallel to the yz-plane. When the axis is parallel to the yz-plane, we need one fewer degree of freedom to specify C, and the solution becomes simpler; for the sake of brevity, we omit the straightforward details of adapting the forthcoming analysis to this case.

Data structure. Inspired by the construction in [6], we construct a two-level partition tree Ψ on P, as follows. For a point $p \in P$, let p^* be its xy-projection, and let $P^* = \{p^* \mid p \in P\}$. Without loss of generality, we assume that no two points in P project to the same point.

Our top-level tree is a two-dimensional partition tree on P^* (with respect to lines in the xy-plane), based on simplicial partitions [24], and each of its nodes stores a second-level partition tree.

Concretely, let S be a set of n points in $\mathbb{R}^2$, and let $r > 0$ be a parameter. A *simplicial* $(1/r)$*-partition* for S is an elementary-cell partition (as defined above), where the elementary cells are triangles. We denote this partition by the collection $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$, where $m \leq r$ is an integer. The *crossing number* of Π for a line ℓ in $\mathbb{R}^2$ is the number of its cells that are crossed by ℓ. The crossing number of Π is defined as the maximum of the crossing numbers over all lines ℓ. Matoušek [24] described an algorithm for constructing a simplicial partition whose crossing number is $O(\sqrt{r})$. If $r = O(1)$, the running time of his algorithm is $O(n)$; see also [16]. (In Matoušek's construction, the cells Δ_i do not have to be disjoint (although the sets S_i are). The later construction of Chan [16] yields cells that are pairwise disjoint.)

We choose r to be a sufficiently large constant. By constructing simplicial partitions recursively and stopping the recursion as soon as the number of points becomes smaller than some sufficiently large constant threshold n_0, we construct a two-dimensional partition tree on P^*, which is the primary tree Ψ of our structure. Each node $v \in \Psi$ is associated with a triangle Δ_v and a subset $P_v^* \subseteq P^* \cap \Delta_v$. If v is the root then $\Delta_v = \mathbb{R}^2$ and $P_v^* = P^*$. Let $P_v = \{p \in P \mid p^* \in P_v^*\}$ be the subset of P corresponding to P_v^*. Set $n_v = |P_v| = |P_v^*|$.

For a node $v \in \Psi$, we build a second-level partition tree (now in $\mathbb{R}^3$) Σ_v on P_v for answering range queries with cylinders whose xy-projections contain Δ_v (we refer to such cylinders as *wide* at v; see below), using the algorithm described below. By Lemma 5 below, P_v can be preprocessed, in $O(n_v \log n_v)$ time, into a linear-size data structure so that an emptiness query for a cylinder whose xy-projection contains Δ_v can be answered in $O^*(n_v^{1/2})$ time.

Query procedure. Let C be a query cylinder whose axis is not parallel to the yz-plane, and let C^* be its xy-projection, which is a strip bounded by two parallel lines. We visit Ψ recursively in a top-down manner, starting from its root. Suppose we are at a node $v \in \Psi$. If $C^* \cap \Delta_v = \emptyset$, then we simply return to the parent recursive step. If v is a leaf, then we check all points of P_v. If any one of them lies in C, we return YES and also such a point; otherwise, we return NO. So assume that v is an internal node and $\Delta_v \cap C^* \neq \emptyset$. If $\Delta_v \subset C$, then we use the secondary data structure Σ_v stored at v to test whether $P_v \cap C \neq \emptyset$, as described below. If $\partial C^* \cap \Delta_v \neq \emptyset$ (i.e., one of the two lines bounding C^* intersects Δ_v), then we recursively visit the children of v. If any of the children returns a point of $P_v \cap C$, we return that point with a YES outcome, and otherwise we return NO. For an emptiness (rather than reporting) query, the procedure can terminate as soon as a point of P inside C is found.

The query procedure can be adapted to report all points of $C \cap P$: for a leaf node v, we report all points of $P_v \cap C$. For an internal node v, if $\Delta_v \subset C^*$, the secondary structure returns all points of $P_v \cap C$; and if $\partial C^* \cap \Delta_v \neq \emptyset$, the recursive processing of v will report all points of $P_v \cap C$.

Analysis. The height of Ψ is $O(\log n)$, and each of its internal nodes v stores a linear-size (linear in n_v) secondary structure, while each leaf uses $O(1)$ space. This is easily seen to imply that the overall size of Ψ is $O(n \log n)$. A similar argument shows that the preprocessing time of constructing Ψ is $O(n \log^2 n)$.

Concerning the query time, we present the analysis for emptiness queries. Denote by $Q(n)$ the maximum emptiness query time in Ψ, constructed on a set of at most n points. For $n \leq n_0$, $Q(n) = O(n)$. If $\Delta_v \subset C^*$ then we use the secondary data structure stored at v, and

answer an emptiness query in $O^*(n_v^{1/2})$ time; see Lemma 5. Since the crossing number of our simplicial partition is $O(\sqrt{r})$, the procedure visits recursively $O(\sqrt{r})$ children of v. We therefore obtain the following recurrence for n.

$$Q(n) = \begin{cases} O(n) & n \le n_0 \\ c_1\sqrt{r}Q(2n/r) + c_2 r n^{1+\varepsilon} & n > n_0, \end{cases}$$

where $c_1, c_2 > 0$ are constants (independent of r). Choosing r to be a sufficiently large constant, the solution is $Q(n) = O^*\left(n^{1/2}\right)$. The overall time for a reporting query, by Lemma 5, is $O^*\left(n^{1/2}\right) + O(k)$.

Returning to the original problem of finding a nearest neighbor for a line query amid points, we have now all the ingredients (modulo those in Lemma 5, described below) for concluding the main result of this section:

▶ **Theorem 4.** *A set P of n points in $\mathbb{R}^3$ can be preprocessed, in $O(n \log n)$ expected time, into a data structure of size $O(n)$, so that (i) for any query line ℓ in $\mathbb{R}^3$, the point of P nearest to ℓ can be computed in $O^*(n^{1/2})$ time; (ii) for a query line ℓ and a distance parameter ρ, all k points of P within distance ρ from ℓ can be reported in $O^*\left(n^{1/2}\right) + O(k)$ time.*

We thus devote the remainder of this section to the proof of Lemma 5, which is the main (still missing) ingredient of the analysis.

Range queries for wide cylinders. Let P be a set of n points in $\mathbb{R}^3$, and let Δ be a triangle in $\mathbb{R}^2$ (xy-plane) that contains P^*, the xy-projection of P. We describe a data structure that, for a query cylinder C whose xy-projection C^* contains Δ, detects whether $C \cap P \neq \emptyset$ (or reports all points of $C \cap P$). If $\Delta \subset C^*$, then the width of C^* is at least that of Δ. Without loss of generality, we can assume that (i) the width of Δ is $\le w_0$ for some suitable sufficiently small but fixed constant $w_0 < 1$, whose value will be set below (see Section 3.2), and (ii) the radius of the query cylinder is at least 1. We refer to cylinders of radius at least 1 as *wide* cylinders (with respect to Δ). We thus assume that P lies in a vertical slab σ of width w_0 bounded by two vertical parallel planes H and H', and that the normals of H and H' are $(\pm 1, 0, 0)$. As is easily verified, these assumptions do not lose generality. Specifically, for larger values of w_0 (recall that $w_0 < 1$ always), we can split the respective vertical slabs into $O(1)$ parallel subslabs of smaller width. As a result, P is split into $O(1)$ subsets, and we can then query each subset with C separately.

By Lemma 3, to answer a range query amid P with a wide cylinder, it suffices to describe an algorithm for constructing a test set for wide cylinders. In the remainder of the analysis we describe an algorithm that, for a given parameter $r > 1$, constructs a test set of size $O^*(r^6)$, in time $nr^{O(1)}$, with $\zeta(m) = O^*(m^2)$ (cf. Lemma 10) and $\delta = 1$. Plugging these bounds into Lemma 3, we obtain the following:

▶ **Lemma 5.** *Let P be a set of n points in $\mathbb{R}^3$, and let Δ be a triangle in $\mathbb{R}^2$ containing the xy-projection of P. Then P can be preprocessed, in $O(n \log n)$ expected time, into a data structure of size $O(n)$, so that for a cylinder C whose xy-projection contains Δ, an emptiness query with C can be answered in $O^*\left(n^{1/2}\right)$ time, and all k points of $C \cap P$ can be reported in $O^*\left(n^{1/2}\right) + O(k)$ time.*

3.2 Test-set construction

As above, let $P \subset \mathbb{R}^3$ be a set of n points lying in a vertical slab σ of width $w_0 = 2\sin^2(\pi/\kappa)$, for some sufficiently large constant integer κ (whose value will be fixed later), bounded by two vertical parallel planes H and H' whose normals are $(\pm 1, 0, 0)$. We now describe an algorithm for constructing a test set for answering range queries with wide cylinders, i.e., cylinders of radius at least 1.

Let $\mathbf{C}$ be the family of cylinders of radius at least 1 whose axes are not parallel to the yz-plane. (Cylinders whose axis is parallel to that plane are easier to handle, and we omit the details here.) A cylinder $C \in \mathbf{C}$ can be represented by a point $\mathbf{p}_C = (y_{C,1}, z_{C,1}, y_{C,2}, z_{C,2}, \rho_C) \in \mathbb{R}^5$, where $\rho_C \geq 1$ is the radius of C, and $(0, y_{C,1}, z_{C,1})$ and $(1, y_{C,2}, z_{C,2})$ are the intersection points of the axis of C with the planes $x = 0$ and $x = 1$. We thus identify C with the point $\mathbf{p}_C$ in the halfspace $\rho \geq 1$ in $\mathbb{R}^5$. (This is the only place where we need the assumption of the axis of the query cylinder not being parallel to the yz-plane, thus allowing us to use a real, rather than projective, parametric 5-space.)

For a cylinder $C_{\mathbf{y}}$ represented by a point $\mathbf{y} = (y_1, z_1, y_2, z_2, \rho)$ in $\mathbb{R}^5$ with $\rho \geq 1$ and for a point $\mathbf{x} \in \mathbb{R}^3$, we can define the predicate $\Pi(\mathbf{x}, \mathbf{y})$ that is 1 iff $\mathbf{x} \in C_{\mathbf{y}}$ (or equivalently, $\mathbf{x}$ is within distance ρ from the axis of $C_{\mathbf{y}}$) by the 8-variate polynomial inequality

$$\Pi(\mathbf{x}, \mathbf{y}) : \|\mathbf{x} - \mathbf{p}_1\|^2 - \left\langle \mathbf{x} - \mathbf{p}_1, \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \right\rangle^2 \leq \rho^2, \tag{1}$$

where $\mathbf{p}_1 = (0, y_1, z_1)$ and $\mathbf{p}_2 = (1, y_2, z_2)$. For a point $q \in \mathbb{R}^3$, we can define the region

$$K_q := \{\mathbf{y} = (y_1, z_1, y_2, z_2, \rho) \in \mathbb{R}^5 \mid \rho \geq 1, \Pi(q, \mathbf{y}) = 1\} \tag{2}$$

to be the set of all points representing wide cylinders that contain q. K_q is clearly a semi-algebraic set of constant complexity.

Algorithm. Let $r \geq 1$ be a parameter. We construct a test set $\mathbf{Q}$ for (n/r)-shallow ranges in $\mathbf{C}$, as follows. We choose a random sample N of $O(r \log r)$ points of P, with an appropriate constant of proportionality. We then form the set of regions $N^* = \{K_p \mid p \in N\}$ and construct their arrangement $\mathcal{A}(N^*)$ in $\mathbb{R}^5$. Let $k = \lceil c \log r \rceil$ be an integer parameter, where $c > 0$ is an appropriate constant of proportionality. Let $\mathcal{A}_{\leq k}(N^*)$ be the set of all points of $\mathcal{A}(N^*)$ at *level* at most k, that is, these points represent all wide cylinders that contain at most k points of N. We compute the vertical decomposition $\mathcal{A}^{\parallel}_{\leq k}(N^*)$ of $\mathcal{A}_{\leq k}(N^*)$, whose size is $O^*(r^6)$ [23]. By a standard random-sampling argument [21], a sufficiently large choice of c implies that, with probability at least $1/2$, each cell of $\mathcal{A}^{\parallel}_{\leq k}(N^*)$ is crossed by at most n/r surfaces of N^*. If this is not the case, we discard N and choose another random subset, until we find, in expected $O(1)$ trials, one with the desired property.

Let τ be a cell in $\mathcal{A}^{\parallel}_{\leq k}(N^*)$. We associate with τ a *generalized cylinder* $\gamma_\tau = \bigcup\{C \in \mathbf{C} \mid \mathbf{p}_C \in \tau\}$, namely the volume swept by a continuous motion of a wide cylinder, as its representative point varies over τ. We set $\mathbf{Q} = \{\gamma_\tau \mid \tau \in \mathcal{A}^{\parallel}_{\leq k}(N^*)\}$; note that $|\mathbf{Q}| = O^*(r^6)$. We state a few key properties of $\mathbf{Q}$, which are similar to those stated in [30] (proofs of some of them are missing in [30], and we provide them for the sake of completeness – see Appendix A.1).

▶ **Lemma 6.** *The set* $\mathbf{Q}$ *has the following properties:*

(i) *Each generalized cylinder* $\gamma_\tau \in \mathbf{Q}$ *is a semi-algebraic set of constant complexity, and it is* (n/r)*-shallow with respect to* P *with probability at least* $1/2$*, for a sufficiently large constant of proportionality.*

(ii) *For any* (n/r)*-shallow wide cylinder* $C \in \mathbf{C}$*, there is a generalized cylinder* $\gamma_\tau \in \mathbf{Q}$ *that contains* C*.*

Union complexity of the test-set. We next prove that for any subset $\mathsf{Q} \subseteq \mathbf{Q}$ of m generalized cylinders, the complexity of $\mathcal{C}(\mathsf{Q}) \cap \sigma$ is $O^*(m^2)$. By Proposition 1, we can then conclude that $\mathcal{C}(\mathsf{Q}) \cap \sigma$ can be decomposed into $O^*(m^2)$ elementary cells. It is known [7, 29] that the complexity of $\mathcal{C}(\mathsf{Q}) \cap \sigma$ is $O(m^2 + \chi)$, where χ is the number of vertices on $\partial\mathcal{C}(\mathsf{Q}) \cap \sigma$, so it suffices to show that $\chi = O^*(m^2)$.

Let κ be a sufficiently large constant. For each generalized cylinder $\gamma \in \mathsf{Q}$, we partition its boundary into *canonical surface patches*, such that each patch is smooth and the outer normals (as points on the unit sphere of directions $\mathbb{S}^2$) of the points on a single patch do not vary by more than $2\pi/\kappa$. Since γ is a semi-algebraic set of constant complexity, $\partial\gamma$ can be partitioned into $O(1)$ canonical surface patches with this property, where the constant of proportionality depends on κ and on the complexity of γ.[2] (As a concrete example demonstrating this decomposition, if γ is just a standard cylinder, the canonical surface patches are portions of the boundary of γ, each of which is bounded by a pair of generator lines, parallel to the axis of γ, with an angular span of $2\pi/\kappa$ of their normals.) Let $\mathcal{P}$ be the set of canonical patches created over all generalized cylinders in Q, clipped to within the slab σ.

As above, let $\mathbb{S}^2$ be the unit sphere of directions in $\mathbb{R}^3$. A direction $\rho \in \mathbb{S}^2$ is *good* for a canonical patch τ if the following two conditions hold:

(i) The angle between ρ and the (outer) normal $\mathbf{n}$ of either of the planes H, H' is smaller than $\pi/2 - \pi/\kappa$. (That is, ρ is not "nearly parallel" to H, H'.)

(ii) Each line tangent to (the relative interior of) τ at any point p forms an angle of at least π/κ with ρ. (That is, ρ is not "too tangent" to τ.)

We say that a direction ρ is *good for a vertex* v of $\mathcal{A}(\mathsf{Q})$ if it is good for each of the at most three patches containing v (each of which lies on the boundary of some generalized cylinder in Q). When ρ is good, $-\rho$ is also good, and we treat ρ and $-\rho$ as a common direction.

Following the analysis in [10, 18], we obtain the following lemma whose proof, omitted here, can be found in these papers.

▶ **Lemma 7.** *There exists a set* $\mathcal{Z}$ *of* $O(1/\kappa^2)$ *directions in* $\mathbb{S}^2$ *so that any vertex of* $\mathcal{A}(\mathsf{Q})$ *has at least one good direction in* $\mathcal{Z}$*.*

We now prove the main property of $\mathcal{C}(\mathsf{Q})$ (we defer the proofs of Lemma 8 and Lemma 9 to Appendix A.1):

[2] Since γ is a semi-algebraic set of constant complexity, its Gauss map $N_\gamma : \gamma \to \mathbb{S}^2$ is also a semi-algebraic set of constant complexity (with a suitable parameterization of $\mathbb{S}^2$). We can thus decompose N_γ into $O(1)$ pieces, each of which is monotone in the radial direction, i.e., any ray emanating from the origin intersects a patch in at most one point. We can then partition each patch into $O(\kappa^2)$ pieces so that each piece lies in a spherical cap of radius π/κ. Each patch of γ corresponds to one of the resulting pieces.

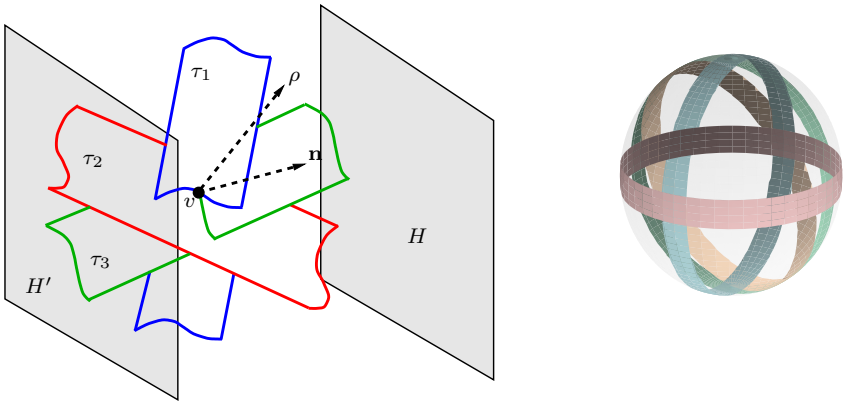

Figure 1 (left) $\mathbf{n}$ is a good direction of a vertex v. (right) The set of bad directions for a vertex.

▶ **Lemma 8** (Good directions; see [10, 18]). *Let σ be a vertical slab of width at most $w_0 = 2\sin^2(\pi/\kappa)$, where κ is a sufficiently large constant. Let τ be a surface patch in $\mathcal{P}$ that intersects σ, let γ be the generalized cylinder whose boundary contains τ, and let $\rho \in \mathcal{Z}$ be a good direction for τ. For any point $u \in \tau \cap \sigma$, if a line ℓ in direction ρ (or $-\rho$) enters γ at u, then ℓ intersects $\partial\sigma$ before exiting γ.*

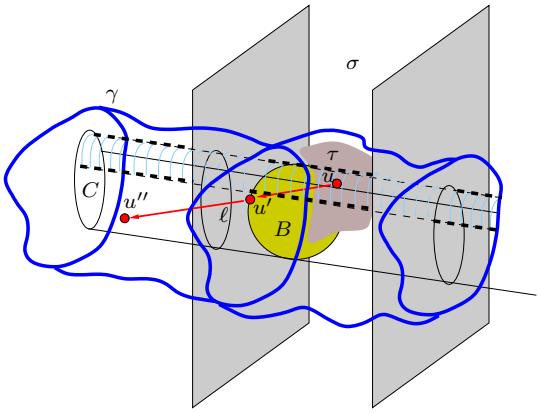

Figure 2 Proof of Lemma 8.

For a direction $\rho \in \mathcal{Z}$, let $\mathcal{P}_\rho$ the set of canonical patches for which ρ is a good direction. A (clipped) patch τ in $\mathcal{P}_\rho$ is called *ρ-upper* (resp., *ρ-lower*) if, for any point $u \in \tau$, the point $u + \alpha\rho$ lies in the exterior (resp., interior) of the generalized cylinder γ associated with τ, for any sufficiently small parameter $\alpha > 0$. Let $\mathcal{P}_\rho^+$ (resp., $\mathcal{P}_\rho^-$) be the set of the ρ-upper (resp., ρ-lower) patches of $\mathcal{P}_\rho$. The *ρ-upper envelope* E_ρ^+ of $\mathcal{P}_\rho^+$ is the set of points u on the patches of $\mathcal{P}_\rho^+$, such that a ray from u in the $(+\rho)$-direction does not meet any other patch in $\mathcal{P}_\rho^+$. The *ρ-lower envelope* E_ρ^- of $\mathcal{P}_\rho^-$ is defined analogously.

▶ **Lemma 9.** *Let v be a vertex of $\mathcal{C}(\mathsf{Q}) \cap \sigma$. Then there exists a direction $\rho \in \mathcal{Z}$ such that v is a vertex of the sandwich region enclosed between E_ρ^+ and E_ρ^-.*

The number of vertices on the sandwich region is $O^*(m^2)$ [9]. Hence, for any direction $\rho \in \mathcal{Z}$, there are $O^*(m^2)$ vertices on $\mathcal{C}(\mathsf{Q}) \cap \sigma$ for which ρ is a good direction. Summing over all $O(1)$ directions of $\mathcal{Z}$, we obtain that the complexity of $\mathcal{C}(\mathsf{Q}) \cap \sigma$ (the complement of the union of Q clipped within σ) is $O^*(m^2)$. Combining this property with Proposition 1, we obtain that $\mathcal{C}(\mathsf{Q}) \cap \sigma$ can be partitioned into $O^*(m^2)$ elementary cells.

Putting everything together, we obtain the main result:

▶ **Lemma 10.** *Let $P \subset \mathbb{R}^3$ be a set of n points in $\mathbb{R}^3$ lying in a vertical slab of width at most w_0, for w_0 as defined above, and let $r \geq 1$ be a parameter. A test set $\mathbf{Q}$ of size $O^*(r^6)$ for all cylinders of radius at least 1 that are (n/r)-shallow with respect to P, with $\zeta(m) = O^*(m^2)$ and $\delta = 1$, can be computed in $nr^{O(1)}$ expected time.*

▶ Remark. The proof of the property that $\zeta(m) = O^*(m^2)$ in Lemma 10 is a variant of the analysis in [18], where standard cylinders were considered. However, a major difference between the two settings, of standard cylinders and generalized cylinders, is that whereas the overall complexity of the union of standard cylinders is $O^*(m^2)$ [10,18], this is not necessarily the case for generalized cylinders. Indeed, we can form a set of m generalized cylinders by taking the Minkowski sums of planes in $\mathbb{R}^3$ with some sufficiently small ball (it is easily checked that such a sum can be a generalized cylinder). Given such a collection, we can easily form a three-dimensional grid-like construction, which results in $\Omega(m^3)$ union complexity. The crucial property, however, is that we only consider generalized cylinders that are *wide* in σ, in which case Lemma 10 implies that the complexity of the union is relatively small (near quadratic) if we remain inside σ.

Tradeoff and the offline setup. Using standard techniques (see, e.g., [1] and [5, Appendix]), one can obtain a tradeoff between storage and query time, essentially interpolating between the two extreme cases, that of near-linear storage and that of fast query time (shown in Theorem 4 and Proposition 2). We therefore obtain:

▶ **Theorem 11.** *Given a set P of n points in $\mathbb{R}^3$, and a parameter $s \in [n, n^4]$, one can preprocess P into a data structure of size $O^*(s)$, in expected time $O^*(s)$, so that, for any query line $\ell \in \mathbb{R}^3$, one can compute $\mathsf{NN}(\ell, P)$ in $O^*((n/s^{1/4})^{2/3})$ expected time.*

3.3 Segment NN queries

We now extend the linear-size data structure described in Section 3.1 for answering fixed-radius neighbor queries with respect to segments. That is, for a query segment e in $\mathbb{R}^3$ and a distance parameter $\rho > 0$, we want to determine whether there is a point of P within distance ρ from e, and, if so, to return such a point. Combining this data structure with parametric search, we can also answer NN queries. For a segment e and a real value $t > 0$, let $\Lambda(e, t) := e \oplus B_t = \{x + y \mid x \in e, y \in B_t\}$, where B_t is the ball of radius t centered at the origin. Borrowing the terminology from [10], we refer to $\Lambda(e, t)$ as a *cigar*. We build a data structure for answering cigar-emptiness queries, as follows.

Data structure. As for the case of lines, we build a two-level data structure, where the top level is a linear-size partition tree T on P^*, the xy-projection of P, for 2D halfplane range searching. The only difference is that at each node v of the top-level tree, we now build a secondary data structure for answering emptiness queries with respect to wide cigars. More precisely, P_v is assumed to lie in a vertical slab σ of width at most $w_0 < 1$, where w_0 is as above, and the radius of a query cigar is assumed to be at least 1. This requires adapting the test-set construction to cigars, as described below.

Query procedure. Let $\Lambda := \Lambda(e, \rho)$ be a query cigar. Let Λ^* (resp. e^*) be the xy-projection of Λ (resp. e), let ℓ_e be the line containing e, and let $W_e \subset \mathbb{R}^2$ be the strip of width 2ρ containing Λ^*.

We visit T recursively in a top-down manner. Suppose we are at a node v. If $\Lambda^* \cap \triangle_v = \emptyset$, we return to the parent of v, so assume that $\Lambda^* \cap \triangle_v \neq \emptyset$. If v is a leaf, we test whether $\Lambda_v \cap P_v \neq \emptyset$ in a brute-force manner and return such a point if the answer is YES. If v is an internal node, we proceed as follows. If $\triangle_v \subset W_e$, we query the secondary structure with Λ to test whether $P_v \cap \Lambda \neq \emptyset$. Finally, if $\triangle_v$ intersects a boundary line of W_e, we recursively visit the children of v. If any of them returns a point of $P_v \cap \Lambda$, we return that point with a YES outcome; otherwise we return NO.

Assuming that the secondary data structure answers a query correctly (as will be argued below), the correctness of the overall query procedure follows immediately. The same analysis as in Section 3.1 implies that the overall query time is $O^*\left(n^{1/2}\right)$ (for emptiness testing), or $O^*\left(n^{1/2}\right) + O(k)$ (for reporting queries).

Test-set construction. Recall that we query the secondary structure with Λ only if $\triangle_v \subset W_e$, implying that the width of $\triangle_v$ is at most 2ρ. We can therefore assume that we build the data structure for a point set P that lies in a vertical slab of width at most w_0, for a sufficiently small constant $w_0 < 1$, and that it is queried with a cigar of radius at least 1. The test-set construction follows the same approach as in Section 3.2. A cigar $\Lambda := \Lambda(e, \rho)$ is represented using 7 real parameters $(a, b, \rho) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}_{\geq 0}$, where a, b are the endpoints of the segment e and ρ is the radius of Λ. For a point $p \in P$, we define $K_p \subset \mathbb{R}^7$ to be the set of points corresponding to cigars that contain p and whose radii are at least 1. With these definitions at hand, the construction of the test set $\mathbf{Q}$, a set of *generalized cigars*, is exactly the same as in Section 3.2 except that we work in a seven-dimensional parametric space. Lemma 8 still holds because for any generalized cigar γ and for any point $u \in \partial\gamma \cap \sigma$, the ball of radius 1 tangent to γ at u lies inside γ, and thus the proof follows verbatim, as can easily be verified. Hence, the complexity of $\mathcal{C}(\mathsf{Q}) \cap \sigma$, for any subset $\mathsf{Q} \subset \mathbf{Q}$ of m generalized cigars, is $O^*(m^2)$.

Omitting further details, we conclude that P, a point set lying in a vertical slab of width w_0, can be preprocessed into a linear-size data structure, so that for a cigar Λ of radius at least 1, one can detect, in $O^*\left(n^{1/2}\right)$ time, whether $\Lambda \cap P \neq \emptyset$. Putting everything together, we obtain:

▶ **Theorem 12.** *A set P of n points in $\mathbb{R}^3$ can be preprocessed, in $O(n \log n)$ expected time, into a data structure of size $O(n)$, so that (i) for any query segment e in $\mathbb{R}^3$, the point of P nearest to e can be computed in $O^*(n^{1/2})$ time; (ii) for a query segment e and a distance parameter ρ, all k points of P within distance ρ from e can be reported in $O^*(n^{1/2}) + O(k)$ time.*

Concerning the fast-query regime, we need to extend Proposition 2 to the case of query segments. Such an extension is done in the recent work [7]. As before, combining both structures, we conclude that Theorem 11 also holds for query segments.

▶ Remark. A natural question is whether the nearest-neighbor of a query triangle in P can also be computed in $O^*\left(n^{1/2}\right)$ time using a linear-size data structure. Unfortunately, it is unlikely that such a data structure exists, because computing the NN of a query plane is as hard as answering slab-emptiness queries in $\mathbb{R}^3$, and the known lower bounds for simplex range searching also hold for slabs [1]. A linear-size data structure with $O^*(n^{2/3})$ query time follows easily from the known results on semi-algebraic range searching in $\mathbb{R}^3$ [4, 26].

4 Nearest-Neighbor Queries with Segments or Triangles in $\mathbb{R}^3$

In this section we extend the data structures developed in Section 3 to the general case where the input is a set $\boldsymbol{\Delta}$ of n triangles in $\mathbb{R}^3$ and the query object is also a triangle τ in $\mathbb{R}^3$, and the goal is to compute $\mathsf{NN}(\tau, \boldsymbol{\Delta})$. We begin with a simpler case in which the input is a set of segments in $\mathbb{R}^3$ and the query object is also a segment in $\mathbb{R}^3$. This will be one of the main ingredients for handling the general case, and will turn out to be the most expensive (and involved) part of the full algorithm for the case of triangles.

The case of segments

Let $\mathcal{E} = \{e_1, \ldots, e_n\}$ be a set of n segments in $\mathbb{R}^3$, and let P be the set of the endpoints of these segments. We wish to preprocess $\mathcal{E}$ into a data structure so that, for a query segment g, $\mathsf{NN}(g, \mathcal{E})$ can be computed quickly. Our data structure is based on the following simple observation.

▶ **Lemma 13.** *Let e and g be two segments in $\mathbb{R}^3$. Let $(p, q) \in e \times g$ be the closest pair of points on these segments. Then either (i) p is an endpoint of e, or (ii) q is an endpoint of g, or (iii) p and q lie in the relative interior of e and g, respectively, in which case they are also the closest pair of points in ℓ_e and ℓ_g, the lines containing these two segments.*

Using Lemma 13, we construct three separate data structures:

Ψ_1: For a parameter $s \in [n, n^3]$, we preprocess $\mathcal{E}$ into a data structure of size $O^*(s)$, in $O^*(s)$ expected time, using Corollary 19 (in Appendix B.1), so that, for a query point $p \in \mathbb{R}^3$, $\mathsf{NN}(p, \mathcal{E})$ can be computed in $O^*\left(n/s^{1/3}\right)$ expected time.

Ψ_2: For a parameter $s \in [n, n^4]$, we preprocess P into a data structure of size $O^*(s)$, in $O^*(s)$ expected time, using Corollary 17 (in Appendix A.2), so that, for a query segment g in $\mathbb{R}^3$, $\mathsf{NN}(g, P)$ can be computed in $O^*\left((n/s^{1/4})^{2/3}\right)$ expected time.

Ψ_3: For a parameter $s \in [n, n^5]$, we preprocess $\mathcal{E}$ into a data structure of size $O^*(s)$, in $O^*(s)$ expected time, as described below (see Theorem 14), so that, for a query segment g in $\mathbb{R}^3$, the structure first identifies the subset $\mathcal{E}_g \subseteq \mathcal{E}$ of segments e such that the closest pair of points in $e \times g$ lie in the relative interiors of e and g, and then it computes $\mathsf{NN}(\ell_g, L_g)$, where $L_g = \{\ell_e \mid e \in \mathcal{E}_g\}$. The query time is $O^*\left((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15}\right)$.

As referenced above, the constructions of Ψ_1 and Ψ_2 are presented in the appendices, so we focus here on Ψ_3. We first derive the condition for the closest pair of points on two segments to lie in their respective relative interiors, and then describe the data structure Ψ_3.

Closest pair. Let e and g be two segments in $\mathbb{R}^3$, and let ℓ_e (resp., ℓ_g) be the line containing e (resp., g). We parameterize a line ℓ in $\mathbb{R}^3$ by the pair (a, b), so that $\ell = \{a + b\lambda \mid \lambda \in \mathbb{R}\}$, where $a = \ell \cap \{z = 0\}$ and b is a unit vector in the direction of ℓ (this is a variant of the representation that we have used earlier in the paper, and it ignores cases where ℓ is horizontal, cases that can indeed be ignored using a generic coordinate frame, or treating them separately, using a simpler structure). Thus each of a, b has two degrees of freedom. Let (a_e, b_e) (resp., (a_g, b_g)) be the parameters describing ℓ_e (resp., ℓ_g). Let $\alpha_e^-, \alpha_e^+ \in \mathbb{R}$ be the two parameters specifying the endpoints of e, namely $e : a_e + b_e\lambda$ for $\lambda \in [\alpha_e^-, \alpha_e^+]$. Similarly, let α_g^-, α_g^+ specify the endpoints of g, so $g : a_g + b_g\mu$ for $\mu \in [\alpha_g^-, \alpha_g^+]$. Thus a segment e can be represented as a point $e^* = (a_e, b_e, \alpha_e^-, \alpha_e^+)$ in $\mathbb{R}^6$. We thus seek parameters λ, $\mu \in \mathbb{R}$ that minimize

$$\begin{aligned}\operatorname{dist}^2(\ell_e, \ell_g) &= |a_e + \lambda b_e - a_g - \mu b_g|^2 \\ &= |a_e - a_g|^2 + \lambda^2 + \mu^2 - 2\lambda\mu(b_g \cdot b_e) + 2\lambda(a_e - a_g) \cdot b_e - 2\mu(a_e - a_g) \cdot b_g,\end{aligned}$$

and then require that $\alpha_e^- \le \lambda \le \alpha_e^+$ and $\alpha_g^- \le \mu \le \alpha_g^+$. The desired values λ, μ then satisfy the equations (of the λ- and μ-derivatives of the right-hand side of the above expression being 0)

$$\begin{aligned}&2\lambda - 2(b_g \cdot b_e)\mu + 2(a_e - a_g) \cdot b_e = 0 \\ &2\mu - 2(b_g \cdot b_e)\lambda - 2(a_e - a_g) \cdot b_g = 0, \qquad \text{or}\end{aligned}$$

$$\lambda = \frac{(a_e - a_g) \cdot [(b_g \cdot b_e)b_g - b_e]}{1 - (b_g \cdot b_e)^2} \qquad \text{and} \qquad \mu = \frac{(a_e - a_g) \cdot [b_g - (b_g \cdot b_e)b_e]}{1 - (b_g \cdot b_e)^2}.$$

Then the condition for the closest pair to lie in the relative interiors of e and g is

$$\alpha_e^- \le \frac{(a_e - a_g) \cdot [(b_g \cdot b_e)b_g - b_e]}{1 - (b_g \cdot b_e)^2} \le \alpha_e^+ \qquad \text{and} \qquad \alpha_g^- \le \frac{(a_e - a_g) \cdot [b_g - (b_g \cdot b_e)b_e]}{1 - (b_g \cdot b_e)^2} \le \alpha_g^+.$$

The denominator is always nonnegative. It is 0 when ℓ_e is parallel to ℓ_g. In such a case $\operatorname{dist}(\ell_g, \ell_e)$ is attained at any point on ℓ_g (and a corresponding point on ℓ_e), and in particular at an endpoint of g. Therefore distances to these parallel lines will be considered when we handle the endpoints of g, using the structure Ψ_1. We therefore only consider input segments for which the denominator is positive, and then the above inequalities are the same as the conjunction of

$$\alpha_e^- \left(1 - (b_e \cdot b_g)^2\right) \le (a_e - a_g) \cdot [(b_e \cdot b_g)b_g - b_e] \tag{3}$$
$$\alpha_e^+ \left(1 - (b_e \cdot b_g)^2\right) \ge (a_e - a_g) \cdot [(b_e \cdot b_g)b_g - b_e] \tag{4}$$
$$\alpha_g^- \left(1 - (b_e \cdot b_g)^2\right) \le (a_e - a_g) \cdot [b_g - (b_g \cdot b_e)b_e] \tag{5}$$
$$\alpha_g^+ \left(1 - (b_e \cdot b_g)^2\right) \ge (a_e - a_g) \cdot [b_g - (b_g \cdot b_e)b_e]\,. \tag{6}$$

As mentioned above, the parametric dimension of a segment in $\mathbb{R}^3$ is 6. However, each inequality in (3)–(6) involves only at most 5 of them (using only one of the parameters specifying the endpoints). Therefore, the reduced parametric dimension [5, Appendix] of both input and query segments in $\mathbb{R}^3$ for our predicate is only 5. Moreover, the reduced parametric dimensions of the input and the query objects are 5 and 4, respectively, in (3) and (4), and they are 4 and 5, respectively, in (5) and (6).

The data structure Ψ_3. We construct a five-level data structure. The first four levels of Ψ_3 are partition trees that, for a query segment g, identify the subset $\mathcal{E}_g$ of segments e for which (3)–(6) hold, namely, those for which the closest pair of points on e and g lie in their respective relative interiors. Each node of the first four levels is associated with a canonical subset of $\mathcal{E}$, and $\mathcal{E}_g$ is returned as the union of a few canonical subsets associated with the nodes of the fourth-level partition tree that the query reaches.

The first level constructs a partition tree $\mathcal{T}^{(1)}$ on $\mathcal{E}$ using a 5-dimensional semi-algebraic range-searching data structure (see [2, 26]), such that for a query segment g, it filters out the subset $\mathcal{E}_g^{(1)}$ of $\mathcal{E}$, as the union of a few canonical subsets, associated with the nodes of $\mathcal{T}^{(1)}$ that the query reaches, for which (3) holds. For each node v of $\mathcal{T}^{(1)}$, we construct a second-level partition tree $\mathcal{T}_v^{(2)}$ on the associated subset $\mathcal{E}_v$, again using a 5-dimensional semi-algebraic range-searching data structure, that, for a query segment g, filters out the

segments of $\mathcal{E}_v$ for which (4) holds. Similarly, the next two levels of partition trees are constructed to filter out the input segments e from the appropriate canonical subset from the preceding level, for which (5), and then (6), holds. We stress again that in (3) and (4) the input segments have reduced parametric dimension 5 and the query segments have dimension 4, and in (5) and (6) it is the other way around. This property is crucial for the analysis of the performance of the query procedure, presented next.

Query procedure for segment nearest neighbor amid segments. For a query segment g, we query the top-level partition tree $\mathcal{T}^{(1)}$ and compute the subset $\mathcal{E}_g^{(1)}$ of segments that satisfy (3) with g, as the union of a family of canonical subsets associated with the elements of the set $\mathcal{F}^{(1)}$ of the nodes in $\mathcal{T}^{(1)}$ that the query reaches. For each node $v \in \mathcal{F}^{(1)}$, we query the second-level partition tree $\mathcal{T}_v^{(2)}$ to compute the subset $\mathcal{E}_{g,v}^{(2)}$ of segments that satisfy (4) with g, again as the union of a small-size family of canonical subsets associated with the elements of the set $\mathcal{F}_v^{(2)}$ of nodes in $\mathcal{T}_v^{(2)}$. Note that $\bigcup_{v\in\mathcal{F}^{(1)}}\bigcup_{w\in\mathcal{F}_v^{(2)}}\mathcal{E}_w$ is the subset of segments of $\mathcal{E}$ that satisfy both (3) and (4), i.e., their closest points to g lie in their relative interiors. By continuing to query in the third and fourth-level partition trees, in a similar manner, we compute $\mathcal{E}_g$ as a family of canonical subsets associated with a suitable subset of the nodes of the fourth-level partition trees. For each such canonical subset $\mathcal{E}_w$, we query the fifth-level data structure, on the set L_w of the lines containing the segments of $\mathcal{E}_w$, compute $\mathsf{NN}(\ell_g, L_w)$, where ℓ_g is the line containing g, and return the segment corresponding to the nearest among the lines produced throughout the process.

A more naïve analysis of the performance of the structure proceeds as follows. Since the reduced parametric dimension of both query and input segments is (at most) 5 at each of the first four levels of Ψ_3, and it is 4 for the fifth level, Theorem A.2 in [5] implies that, for a storage parameter $s \in [n, n^5]$, we can construct Ψ_3 with size and expected preprocessing cost $O^*(s)$, so that the overall query time is $O^*\left(n/s^{1/5}\right)$.

However, a more careful analysis of the runtime of the query procedure leads to a slightly improved bound on the query time. Specifically, for each $i \le 5$, let t_o^i (resp., t_q^i) denote the reduced parametric dimension of the input (resp., query) segments of the polynomial inequality used in the i-th level of the data structure, and let $t_o = \max_i t_o^i$ and $t_q = \max_i t_q^i$, both of which are 5. Again, proceeding naïvely, the analysis in [5, Appendix] shows that, for a storage parameter $s \in [n, n^{t_q}]$, the query time is $O^*\left((n/s^{1/t_q})^{\frac{1-1/t_o}{1-1/t_q}}\right)$, which leads to the query time of $O^*(n/s^{1/5})$ as mentioned above. But a more refined solution of the recurrence for the query time, based on the technique of [5] (see eq. (A11) in Appendix A.2 of that paper) can be written in our context as

$$Q(n,s) = O^*\left(\sum_{i=1}^{5}\left(\frac{n}{s^{1/t_q^i}}\right)^{\frac{1-1/t_o^i}{1-1/t_q^i}}\right).$$

Note that in the fifth level we have $t_o = t_q = 4$, so its contribution to the overall query bound is dominated by the cost of the preceding levels, and we obtain that the query time is $O^*\left((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15}\right)$. For $s \in [n, n^{31/19}]$, the second term dominates and thus the query time for these values of s is $O^*((n/s^{1/4})^{16/15})$. For $s \in [n^{31/19}, n^5]$, the first term dominates and thus the query time becomes $O^*((n/s^{1/5})^{15/16})$. (At the breakpoint value $s = n^{31/19}$, the query time is $O^*(n^{12/19})$.)

If the input is a set of lines instead of segments, then the reduced parametric dimension of the input objects is 4. Moreover, we do not have to construct the first two levels of the partition trees, i.e., Ψ_3 is only a three-level data structure. In this case, the query

time improves to $O^*((n/s^{1/5})^{15/16})$ for $s \in [n, n^5]$. (The improvement is for the range $s \in [n, n^{31/19}]$.) Analogously, if the input is a set of segments but the query objects are lines, then the reduced parametric dimension of query objects reduces to 4 and we do not have to construct the third and fourth levels of partition trees in Ψ_3. Again, by Theorem A.2 in [5], the query time becomes $O^*((n/s^{1/4})^{16/15})$ for $s \in [n, n^4]$. (Here the improvement is for the range $s \in [n^{31/19}, n^4]$; note the reduced range of s.) Altogether, we obtain:

▶ **Theorem 14.**

(i) *Let $\mathcal{E}$ be a set of n segments in $\mathbb{R}^3$, and let $s \in [n, n^5]$ be a storage parameter. $\mathcal{E}$ can be preprocessed, in randomized expected time $O^*(s)$, into a data structure of size $O^*(s)$, so that for a query segment g, $\mathsf{NN}(g, \mathcal{E})$ can be computed in $O^*((n/s^{1/5})^{15/16}+(n/s^{1/4})^{16/15})$ time. If $\mathcal{E}$ is a set of n lines, then the query time can be improved to $O^*((n/s^{1/5})^{15/16})$.*

(ii) *If the query objects are lines but the input is a set $\mathcal{E}$ of segments, then, for a storage parameter $s \in [n, n^4]$, $\mathcal{E}$ can be preprocessed, in randomized expected time $O^*(s)$, into a data structure of size $O^*(s)$, so that for a query line ℓ, $\mathsf{NN}(\ell, \mathcal{E})$ can be computed in $O^*((n/s^{1/4})^{16/15})$ time.*

In Appendix B.2 we present an extension of this mechanism to the case where both input and query objects are triangles. For this we need to construct data structures for additional subproblems, like querying with a vertex of the query triangle amid the input triangles, or querying with the query triangle amid the vertices of the input triangles. As it turns out, the performance cost of these structures is dominated by that of the segment-segment case reviewed above. We thus conclude:

▶ **Theorem 15.** *Let $\mathbf{\Delta}$ be a set of n triangles in $\mathbb{R}^3$, and let $s \in [n, n^5]$ be a storage parameter. $\mathbf{\Delta}$ can be preprocessed, in randomized expected time $O^*(s)$, into a data structure of size $O^*(s)$, so that, for a query triangle τ, $\mathsf{NN}(\tau, \mathbf{\Delta})$ can be computed in $O^*\left((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15}\right)$ time.*

References

1. P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.
2. P. K. Agarwal, B. Aronov, E. Ezra, and J. Zahl. An efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50:760–787, 2021. Also in *Proc. Sympos. on Computational Geometry (SoCG)*, 2019, 5:1–5:14. Also in arXiv:1812.10269.
3. P. K. Agarwal and J. Matousek. On range searching with semialgebraic sets. *Discret. Comput. Geom.*, 11:393–418, 1994. doi:10.1007/BF02574015.
4. P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets II. *SIAM J. Comput.*, 42:2039–2062, 2013. Also in arXiv:1208.3384.
5. Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Matthew J. Katz, and Micha Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. *ACM Trans. Algorithms*, 21(3):25:1–25:59, 2025. doi:10.1145/3721290.
6. Pankaj K. Agarwal and Esther Ezra. Line intersection searching amid unit balls in 3-space. *Algorithmica*, 87(2):223–241, 2025. doi:10.1007/S00453-024-01284-7.
7. Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Vertical decomposition in 3d and 4d with applications to line nearest-neighbor searching in 3d. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 150–170. SIAM, 2024. doi:10.1137/1.9781611977912.8.
8. Pankaj K. Agarwal and Jirí Matousek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993. doi:10.1137/0222051.

9 Pankaj K. Agarwal, Otfried Schwarzkopf, and Micha Sharir. The overlay of lower envelopes and its applications. *Discret. Comput. Geom.*, 15(1):1–13, 1996. doi:10.1007/BF02716576.

10 Pankaj K. Agarwal and Micha Sharir. Pipes, cigars, and kreplach: the union of minkowski sums in three dimensions. *Discret. Comput. Geom.*, 24(4):645–657, 2000. doi:10.1007/S004540010064.

11 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

12 Boris Aronov, Alon Efrat, Vladlen Koltun, and Micha Sharir. On the union of kappa-round objects in three and four dimensions. *Discret. Comput. Geom.*, 36(4):511–526, 2006. doi:10.1007/S00454-006-1263-X.

13 Sunil Arya and David Mount. *Computational Geometry: Proximity and Location*, chapter 63. CRC Press, LLC, 2005.

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

15 Christian Bär. *Elementary Differential Geometry*. Cambridge University Press, New York, NY, 2010.

16 Timothy M. Chan. Optimal partition trees. *Discret. Comput. Geom.*, 47(4):661–690, 2012. doi:10.1007/S00454-012-9410-Z.

17 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. A singly exponential stratification scheme for real semi-algebraic varieties and its applications. *Theor. Comput. Sci.*, 84(1):77–105, 1991. doi:10.1016/0304-3975(91)90261-Y.

18 Esther Ezra. On the union of cylinders in three dimensions. *Discret. Comput. Geom.*, 45(1):45–64, 2011. doi:10.1007/S00454-010-9312-X.

19 Esther Ezra and Micha Sharir. On the union of fat tetrahedra in three dimensions. *J. ACM*, 57(1):2:1–2:23, 2009. doi:10.1145/1613676.1613678.

20 Esther Ezra and Micha Sharir. On ray shooting for triangles in 3-space and related problems. *SIAM J. Comput.*, 51(4):1065–1095, 2022. doi:10.1137/21M1408245.

21 David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discret. Comput. Geom.*, 2:127–151, 1987. doi:10.1007/BF02187876.

22 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

23 Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004. doi:10.1145/1017460.1017461.

24 Jirí Matousek. Efficient partition trees. *Discret. Comput. Geom.*, 8:315–334, 1992. doi:10.1007/BF02293051.

25 Jirí Matousek. Reporting points in halfspaces. *Comput. Geom.*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

26 J. Matoušek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015. doi:10.1007/S00454-015-9701-2.

27 J. S. B. Mitchell and W. Mulzer. Proximity algorithms. In Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 32, pages 849–874. CRC Press, 3rd edition, 2017.

28 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. Nearest-neighbor methods in learning and vision. *IEEE Trans. Neural Networks*, 19(2):377, 2008. doi:10.1109/TNN.2008.917504.

29 M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge-New York-Melbourne, 1995.

30 M. Sharir and H. Shaul. Ray shooting and stone throwing with near-linear storage. *Comput. Geom. Theory Appls.*, 30:239–252, 2005. doi:10.1016/J.COMGEO.2004.10.001.

A Missing Proofs and Details

A.1 Missing proofs from Section 3.2

Proof of Lemma 6. (i) It is well known that every cell τ is a semi-algebraic set defined as a conjunction of at most 10 polynomial inequalities, each of constant degree. Let $\Phi_\tau : \mathbb{R}^5 \to \{0,1\}$ be the resulting predicate, i.e., $\Phi_\tau(\mathbf{y}) = 1$ iff $\mathbf{y} \in \tau$. By applying (1), the generalized cylinder γ_τ can be expressed as

$$\gamma_\tau = \{\mathbf{x} \in \mathbb{R}^3 \mid \exists \mathbf{y}\ (\Pi(\mathbf{x},\mathbf{y}) \wedge \Phi_\tau(\mathbf{y}))\}.$$

Using a quantifier-elimination procedure [14], γ_τ can be expressed as a quantifier-free formula, simply consisting of Boolean operations on polynomial inequalities in $\mathbf{x}$, implying that γ_τ is a semi-algebraic set. Since Π and Φ_τ are constant-complexity semi-algebraic predicates, so is the formula produced by the quantifier-elimination procedure. Hence, γ_τ is a semi-algebraic set of constant complexity.

Next, suppose γ_τ contains a point $q \in P$. Let C be a cylinder with $p_C \in \tau$ such that $q \in C$. Then, by definition, $p_C \in K_q$. Since $p_C \in \tau$ and τ is contained in a cell of $\mathcal{A}(N^*)$, we conclude that $\tau \subset K_q$ if $q \in N$. But the level of τ in $\mathcal{A}(N^*)$ is at most $k = \lceil c \log r \rceil$, so at most k regions of N^* contain τ. Following the same argument as in [30] (based on so-called shallow nets), one can now argue that with probability at least $1 - 1/r^{O(1)}$, there are at most n/r points q of P such that K_q contains or crosses τ. Hence, γ is (n/r)-shallow with probability at least $1 - 1/r^{O(1)}$. Taking the union bound over all cells of $\mathcal{A}^{\parallel}_{\leq k}(N^*)$, we conclude that all generalized cylinders in $\mathbf{Q}$ are (n/r)-shallow with probability at least $1/2$.

(ii) Since N is a random sample of size $O(r \log r)$ points, it is known that for all (n/r)-shallow cylinders C, $|C \cap N| \leq c \log r$. Hence, the level of the point $\mathbf{p}_C$ in $\mathcal{A}(N^*)$ is at most $c \log r$, implying that there exists a cell $\tau \in \mathcal{A}^{\parallel}_{\leq k}(N^*)$ that contains $\mathbf{p}_C$. By construction γ_τ contains C, as claimed.

Proof of Lemma 8. It follows by the definition of a generalized cylinder that u lies on the boundary of a wide cylinder C fully contained in γ. Let $r \geq 1$ be the radius of C, and let x be the point on the axis of C closest to u, i.e., $\|x - u\| = r$. Let B be the ball of radius r centered at x; we thus have $B \subset C \subseteq \gamma$ and $u \in \partial B$. Let h_u be the tangent plane to B at u. Using techniques from elementary differential geometry [15], it can be verified that h_u is also the tangent plane of γ at u.

Let u' be the other intersection point of ∂B and ℓ, and let u'' be the other intersection point of $\partial\gamma$ and ℓ. Since $B \subset C \subseteq \gamma$ it follows that u' must be contained in γ, so, as we walk along ℓ from u, u'' must lie further than u'. We now observe that $|uu'|$ is minimized when uu' forms an angle of π/κ with h_u, as we recall that by property (ii) of good directions, any such angle is at least π/κ with res[ect to the tangent plane to τ (which is identical to h_u as just noted). It thus follows that $|uu'| \geq 2\sin(\pi/\kappa)$. Indeed, assume, without loss of generality, that h_u is parallel to the xy-plane and that u is the north pole of B. By the above condition, the vector uu' must lie in a vertical cone (its axis is a vertical ray in the negative z-direction) whose apex angle is $\pi/2 - \pi/\kappa$. Furthermore $r \geq 1$, so the lower bound on $|uu'|$ follows.

Finally, uu' forms an angle of at most $\pi/2 - \pi/\kappa$ with the normal $\mathbf{n}$ (to either of the two bounding planes of σ'), so the length of the projection of uu' on $\mathbf{n}$ is at least $2|uu'|\sin(\pi/\kappa) \geq 2\sin^2(\pi/\kappa)$. Since the width of σ is at most $2\sin^2(\pi/\kappa)$, the lemma follows.

Proof of Lemma 9. Suppose that v is incident to the three patches τ_1, τ_2, τ_3. By Lemma 7, there is a direction $\rho \in \mathcal{Z}$ that is good for v, i.e., good for all three patches, so all of them belong to $\mathcal{P}_\rho$. For $i \in \{1, 2, 3\}$, suppose τ_i is an upper patch. If v does not appear on the ρ-upper envelope E^+_ρ, then the ray $v + \alpha\rho$ intersects another upper patch τ' at a point v' before exiting the slab σ. Since τ' is an upper patch, the ray $v' - \alpha\rho$ enters the generalized cylinder γ' containing τ', but the ray exits γ' before leaving σ because $v \in \mathcal{C}(\mathsf{Q})$ and $vv' \subset \sigma$, which contradicts Lemma 8. Hence, we conclude that v lies on E^+_ρ. Similarly if τ_i is a lower patch, v lies on the lower envelope E^-_ρ of $\mathcal{P}^-_\rho$. We thus conclude that v is a vertex of the sandwich region enclosed between E^+_ρ and E^-_ρ.

A.2 Triangle/Segment NN queries: Fast query time

We can extend the data structure stated in Proposition 2 to the case where the query objects are segments or triangles in $\mathbb{R}^3$.

First, consider a segment $e = ab$ in $\mathbb{R}^3$. Let ℓ_e be the line containing e, let W_e be the slab bounded by the two planes passing through the endpoints of e and orthogonal to e. For a point $p_i \in P$, if $p_i \notin W_e$ then $\mathrm{dist}(p_i, e) = \min\{\|p_i - a\|, \|p_i - b\|\}$, otherwise $\mathrm{dist}(p_i, e) = \mathrm{dist}(p_i, \ell_e)$.

We therefore build two data structures. In the first structure, we preprocess P into an $O^*(n^2)$-size data structure so that for a query point $q \in \mathbb{R}^3$, $\mathsf{NN}(q, P)$ can be computed in $O(\log n)$ time [1]. The second structure is a multi-level partition tree T on P, builds on the one just described. Each node v of T stores a canonical subset $P_v \subseteq P$. The first two levels build halfspace range-searching data structures so that for a query segment $e \in \mathbb{R}^3$, we can extract the subset $P_e = P \cap W_e$ as a union of $O^*(1)$ canonical subsets. At each node w of the second level, we build the data structure stated in Proposition 2, so that for a query line ℓ_e, $\mathsf{NN}(\ell_e, P_w)$ can be computed.

Let e be a query segment. By querying the first data structure with the endpoints of e, we compute the point $p_i \in P$ that is nearest to an endpoint of e. By querying the second data structure with e, we compute $\mathsf{NN}(\ell_e, P \cap W_e)$. The closer of these two points is $\mathsf{NN}(e, P)$.

The total size of the data structure is $O^*(n^4)$ and the query time remains $O^*(1)$. (The cost of a NN query with a point amid points is subsumed by the cost of the other structure.)

Next, consider a triangle $\triangle$ in $\mathbb{R}^3$. Let $\pi_\triangle$ be the plane containing $\triangle$, let $\ell_\triangle$ be the line passing through the origin and normal to $\pi_\triangle$, and let $W_\triangle = \triangle \oplus \ell_\triangle$ be the unbounded prism spanned by $\triangle$ and orthogonal to it. If a point $p_i \notin W_\triangle$, then the point of $\triangle$ closest to p_i lies on one of its edges, and if $p_i \in W_\triangle$ then $\mathrm{dist}(p_i, \triangle) = \mathrm{dist}(p_i, \pi_\triangle)$.

We first construct the data structure just described for computing the NN of a query segment to a point in P. Next, we preprocess P into a simplex range-emptiness data structure of size $O^*(n^3)$. For a query triangle $\triangle$, it computes $\mathsf{NN}(\pi_\triangle, P \cap W_\triangle)$, as follows. For a real value $t > 0$, let $W_{\triangle,t} \subset W_\triangle$ be the set of points within distance t from $\pi_\triangle$; $W_{\triangle,t}$ is a bounded prism with its base being a translate of $\triangle$. Using the above data structure, we can determine in $O(\log n)$ time whether $P \cap W_{\triangle,t} \neq \emptyset$ and return such a point if the answer is yes. By combining this data structure with parametric search, we can compute $\mathsf{NN}(\pi_\triangle, P \cap W_\triangle)$ in $O^*(1)$ time.

For a query triangle $\triangle$, we first compute the point of P that is closest to its edges using the first data structure and then compute $\mathsf{NN}(\pi_\triangle, P \cap W_\triangle)$ using the second data structure. The closer of these two points is $\mathsf{NN}(\triangle, P)$.

Putting everything together, we obtain the following.

▶ **Corollary 16.** *Let P be a set of n points in $\mathbb{R}^3$. Then P can be preprocessed, in $O^*(n^4)$ expected time, into a data structure of size $O^*(n^4)$, so that, for any query triangle (or segment) $\triangle \in \mathbb{R}^3$, $\mathsf{NN}(\triangle, P)$ can be computed in $O^*(1)$ time.*

We can again obtain a space/query-time trade-off. Recall from Theorem 12 that the NN of a query segment can be computed in $O^*(\sqrt{n})$ time using $O(n)$ space, but computing the NN of a query triangle takes $O^*(n^{2/3})$ time using $O(n)$ space. Hence, we obtain the following:

▶ **Corollary 17.** *Let P be a set of n points in $\mathbb{R}^3$ and $s \in [n, n^4]$ a parameter. Then P can be preprocessed, in $O^*(s)$ expected time, into a data structure of size $O^*(s)$, so that, for any query segment e in $\mathbb{R}^3$, $\mathsf{NN}(e, P)$ can be computed in $O^*((n/s^{1/4})^{2/3})$ time. For a query triangle, the query time is $O((n/s^{1/4})^{8/9})$.*

B Nearest-Neighbor Queries with Segments or Triangles in $\mathbb{R}^3$

B.1 Point NN-Queries amid Segments or Triangles

In what follows we will rely on the following proximity result from [7]:

▶ **Proposition 18** ([7]). *Let L be a set of n lines in $\mathbb{R}^3$, and let $s \in [n, n^3]$ be a parameter. Then L can be preprocessed, in $O^*(s)$ expected time, into a data structure of size $O^*(s)$, so that for any query point $q \in \mathbb{R}^3$, $\mathsf{NN}(q, L)$ can be computed in $O^*(n/s^{1/3})$ expected time.*

Our goal is to show that the data structure from Proposition 18 can be extended to answering NN queries when the input objects are segments or triangles. We first consider the case of segments, and focus on the regime of near-linear storage.

Let $\mathcal{E} = \{e_1, \ldots, e_n\}$ be a set of n segments in $\mathbb{R}^3$. For a segment $e_i \in \mathcal{E}$, let ℓ_i be the line containing e_i and W_i the slab bounded by the two planes passing through the endpoints of e_i and orthogonal to e_i. We follow the same idea as in Appendix A.2. We build two data structures: The first one preprocesses P, the set of endpoints of the segments in $\mathcal{E}$, into a data structure for answering $\mathsf{NN}(q, P)$ queries, for a query point $q \in \mathbb{R}^3$. The second structure is a multi-level partition tree T on $\mathcal{E}$, each of whose nodes v stores a canonical subset $\mathcal{E}_v \subseteq \mathcal{E}$. The first two levels build halfspace range-searching data structures so that for a query point $q \in \mathbb{R}^3$, we can extract the subset $\mathcal{E}_q$ of segments $e_i \in \mathcal{E}$ such that $q \in W_i$, as a union of a few canonical subsets (see below for precise bounds). At each node w of the second level, let $\mathcal{L}_w$ be the set of lines containing the segments of the canonical subset $\mathcal{E}_w$. We construct a NN-query data structure for $\mathcal{L}_w$, following Proposition 18.

Let $q \in \mathbb{R}^3$ be a query point. By querying the first data structure with q, we compute the segment e_i of $\mathcal{E}$ whose endpoint is closest to q. Next, using the second data structure, among those segments e_i for which $q \in W_i$, we compute the segment e_i nearest to q. The closer of these two segments is $\mathsf{NN}(q, \mathcal{E})$. The total size of the data structure is $O^*(n)$ and the query time remains $O^*(n^{2/3})$. (The cost of a NN query with a point amid points is subsumed by the cost of the other structure.)

Next, let $\mathcal{T} = \{\triangle_1, \ldots, \triangle_n\}$ be a set of n triangles in $\mathbb{R}^3$. Let $\mathcal{E}$ be the set of edges of the triangles in $\mathcal{T}$. For a triangle $\triangle_i$, let π_i be the plane containing $\triangle_i$, let ℓ_i be the line passing through the origin and normal to π_i, and let $W_i = \triangle_i \oplus \ell_i$ be the prism spanned by $\triangle_i$ and orthogonal to it.

We again build two data structures. We first construct the NN data structure just described on the set $\mathcal{E}$. For a point $q \in \mathbb{R}^3$, let $\mathcal{T}_q = \{\triangle_i \in \mathcal{T} \mid q \in W_i\}$. Next, we build a following four-level partition tree. For a query point $q \in \mathbb{R}^3$, the first three levels extract the

subset $\mathcal{T}_q$ of triangles, as a union of a few canonical subsets. For each canonical subset $\mathcal{T}_z$, let P_z be the set of points dual to the planes containing the triangles of $\mathcal{T}_z$. We preprocess P_z into a linear-size semi-algebraic range-searching data structure [4, 26], where the query ranges are the dual regions of a ball in $\mathbb{R}^3$. More precisely, for a point $q \in \mathbb{R}^3$ and a parameter $\rho \geq 0$, let $\gamma_{q,\rho}$ be the set of points dual to the planes that are within distance ρ from q (namely, intersect the ball of radius ρ centered at q). Therefore, for a query point $q \in \mathbb{R}^3$ and a node z at the third-level tree such that $\mathcal{T}_z \subseteq \mathcal{T}_q$, we combine the fourth-level data structure with parametric search, and, as a consequence, compute $\mathsf{NN}(q, \mathcal{T}_z)$ in time $O^*(|\mathcal{T}_z|^{2/3})$. By querying both data structures with q, we obtain $\mathsf{NN}(q, \mathcal{T})$ as the nearest of these two output triangles.

For the fast query regime, we can construct constant-complexity semi-algebraic trivariate function $f_\triangle$ for each $\triangle \in \mathcal{T}$ that measures the distance of a point $q \in \mathbb{R}^3$ from $\triangle$. By the analysis in [7] concerning vertical decompositions of "lower envelopes" of trivariate functions, we can preprocess these functions, in $O^*(n^3)$ expected time, into a data structure of $O^*(n^3)$ size, so that $\mathsf{NN}(q, \mathcal{T})$ for a query point $q \in \mathbb{R}^3$ can be computed in $O(\log n)$ time. By combining this data structure with the $O^*(n)$-size data structure just presented, we can obtain space/query-time trade-off, with the same performance bound as in Proposition 18. Putting everything together, we obtain the following:

▶ **Corollary 19.** *Let $\mathcal{T}$ be a set of n triangles (or segments) in $\mathbb{R}^3$, and let $s \in [n, n^3]$ be a parameter. $\mathcal{T}$ can be preprocessed, in $O^*(s)$ expected time, into a data structure of size $O^*(s)$, so that for any query point $q \in \mathbb{R}^3$, $\mathsf{NN}(q, \mathcal{T})$ can be computed in $O^*(n/s^{1/3})$ expected time.*

B.2 The case of triangles

Next, we consider the general case where the input is a set $\boldsymbol{\Delta} = \{\triangle_1, \ldots, \triangle_n\}$ of n triangles in $\mathbb{R}^3$, and the query objects are also triangles in $\mathbb{R}^3$. Let $\mathcal{E}$ and P be the set of edges and vertices, respectively, of the triangles in $\boldsymbol{\Delta}$.

We use the following lemma, whose proof is straightforward.

▶ **Lemma 20.** *Let $\triangle, \triangle'$ be two triangles in $\mathbb{R}^3$. The closest pair of points in $\triangle \times \triangle'$ satisfy the following properties:*

- **(i)** *If $\triangle$ and $\triangle'$ intersect, the closest pair is (p, p), where p is an intersection point. Furthermore, in this case, either an edge of $\triangle$ intersects $\triangle'$, or an edge of $\triangle'$ intersects $\triangle$.*
- **(ii)** *If $\triangle$ and $\triangle'$ do not intersect, let $(p, p') \in \triangle \times \triangle'$ be the closest pair of points. Then either (a) p is a vertex of $\triangle$, (b) p' is a vertex of $\triangle'$, or (c) each of p and p' lies on the relative interior of respective edges of $\triangle$ and $\triangle'$. In cases (a) and (b) the other point p or p' can lie anywhere on its triangle.*

Overall data structure. Using Lemma 20, we construct five separate data structures, one for each case of the lemma.

Ψ_1: The goal here is to preprocess $\boldsymbol{\Delta}$ into a data structure that, for a query segment e, returns a triangle of $\boldsymbol{\Delta}$ intersecting e, if one exists. Using a 4-dimensional semi-algebraic range searching data structure [2, 4, 8], for a storage parameter $s \in [n, n^4]$, a query can be answered in $O^*(n/s^{1/4})$ time using $O^*(s)$ space and expected preprocessing cost. Roughly, a segment e intersects a triangle $\triangle$ iff (i) the endpoints of e lie on different sides of the plane supporting $\triangle$, and (ii) the oriented line supporting e has the same (positive or negative) orientation with respect to the three lines supporting the edges

of $\triangle$ and cyclically oriented around $\triangle$. Each of these constraints can be restated as a range searching task, where in (i) we face range searching where both input and query objects have parametric dimensions 3, and in (ii) these dimensions are both 4.

Ψ_2: The goal here is to preprocess $\mathcal{E}$, the set of edges of the triangles of $\mathbf{\Delta}$, into a data structure that, for a query triangle τ, returns a segment e of $\mathcal{E}$ intersecting τ, if one exists. Again, using a 4-dimensional multi-level semi-algebraic range searching data structure [2, 4, 8], for a storage parameter $s \in [n, n^4]$, a query can be answered in $O^*(n/s^{1/4})$ time using $O^*(s)$ space and expected preprocessing time. (Symmetric but similar considerations to those used for Ψ_1 apply here too.)

Ψ_3: The goal here is to preprocess $\mathbf{\Delta}$ into a data structure that, for a query point q, returns $\mathsf{NN}(q, \mathbf{\Delta})$. Using Corollary 19, for a storage parameter $s \in [n, n^3]$, such a structure, with $O^*(s)$ storage and expected preprocessing time, can be constructed so that a query can be answered in $O^*(n/s^{1/3})$ time.

Ψ_4: The goal here is to preprocess P, the set of vertices of the triangles of $\mathbf{\Delta}$, into a data structure that, for a query triangle τ, returns $\mathsf{NN}(\tau, P)$. Using Corollary 17, for a storage parameter $s \in [n, n^4]$, such a structure, with $O^*(s)$ storage and expected preprocessing time, can be constructed so that a query can be answered in $O^*((n/s^{1/4})^{8/9})$ time.

Ψ_5: The goal here is to preprocess $\mathcal{E}$, the set of edges of the triangles of $\mathbf{\Delta}$, into a data structure that, for a query segment g, returns $\mathsf{NN}(g, \mathcal{E})$. Using Theorem 14, for a storage parameter $s \in [n, n^5]$, such a structure, with $O^*(s)$ storage and expected preprocessing time, can be constructed so that a query can be answered in $O^*((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15})$ time.

We note that for Ψ_1, and Ψ_2, we can achieve better performance, using the mechanisms in [5] and [20], but these improvements do not affect the total query bound – see below.

Query procedure. Let τ be a query triangle. First, by querying Ψ_1 with the edges of τ, we determine whether an edge of τ intersects any triangle of $\mathbf{\Delta}$. If so, we return such a triangle and stop. Otherwise, we query Ψ_2 with τ. If an edge of a triangle $\triangle \in \mathbf{\Delta}$ intersects τ, we return $\triangle$ and stop.

So assume that τ does not intersect any triangle of $\mathbf{\Delta}$. We first query Ψ_3 with the vertices of τ and compute the triangle $\triangle_1$ that is closest to a vertex of τ. Next, we query Ψ_4 with τ and compute $p_\tau = \mathsf{NN}(\tau, P)$. Let $\triangle_2$ be the triangle of $\mathbf{\Delta}$ that contains p_τ. Finally, we query Ψ_5 with the edges of τ and compute the segment (triangle edge) e_τ of $\mathcal{E}$ that is nearest to an edge of τ (where the distance is attained at interior points of both segments). Let $\triangle_3 \in \mathbf{\Delta}$ be the triangle that contains e_τ. Among $\triangle_1, \triangle_2, \triangle_3$, we return the one that is closest to τ. The overall query time is $O^*((n/s^{1/5})^{15/16} + (n/s^{1/4})^{16/15})$; this is the cost of a query on Ψ_5, which dominates the cost of all the other queries, as is easily verified. Hence, we obtain Theorem 15.