On Ray Shooting for Triangles in 3-Space and Related Problems

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

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

Abstract

We consider several problems that involve lines in three dimensions, and present improved algorithms for solving them. The problems include (i) ray shooting amid triangles in $\mathbb{R}^3$, (ii) reporting intersections between query lines (segments, or rays) and input triangles, as well as approximately counting the number of such intersections, (iii) computing the intersection of two nonconvex polyhedra, (iv) detecting, counting, or reporting intersections in a set of lines in $\mathbb{R}^3$, and (v) output-sensitive construction of an arrangement of triangles in three dimensions.

Our approach is based on the polynomial partitioning technique.

For example, our ray-shooting algorithm processes a set of n triangles in $\mathbb{R}^3$ into a data structure for answering ray shooting queries amid the given triangles, which uses $O(n^{3/2+\varepsilon})$ storage and preprocessing, and answers a query in $O(n^{1/2+\varepsilon})$ time, for any $\varepsilon > 0$. This is a significant improvement over known results, obtained more than 25 years ago, in which, with this amount of storage, the query time bound is roughly $n^{5/8}$. The algorithms for the other problems have similar performance bounds, with similar improvements over previous results.

We also derive a nontrivial improved tradeoff between storage and query time. Using it, we obtain algorithms that answer m queries on n objects in

$$\max\left\{O(m^{2/3}n^{5/6+\varepsilon} + n^{1+\varepsilon}),\ O(m^{5/6+\varepsilon}n^{2/3} + m^{1+\varepsilon})\right\}$$

time, for any $\varepsilon > 0$, again an improvement over the earlier bounds.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Ray shooting, Three dimensions, Polynomial partitioning, Tradeoff

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.34

Related Version *Full Version*: https://arxiv.org/abs/2102.07310

Funding *Esther Ezra*: Work partially supported by NSF CAREER under grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation.
Micha Sharir: Work partially supported by ISF Grant 260/18, by grant 1367/2016 from the German-Israeli Science Foundation (GIF), and by Blavatnik Research Fund in Computer Science at Tel Aviv University.

Acknowledgements We wish to thank Pankaj Agarwal for the useful interaction concerning certain aspects of the range searching problem.

1 Introduction

In this paper we consider several algorithmic problems that involve, explicitly or implicitly, a finite set of lines in three dimensions. The main problems that we consider are:

(i) *Ray shooting amid triangles in three dimensions.* Given a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$, preprocess $\mathcal{T}$ into a data structure that supports efficient ray-shooting queries, each of which specifies a ray ρ and asks for the first triangle of $\mathcal{T}$ that is hit by ρ, if such a triangle exists.

(ii) *Intersection reporting, emptiness, and approximate counting queries amid triangles in three dimensions.* For a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$, preprocess $\mathcal{T}$ into a data structure that supports efficient intersection reporting (resp., emptiness) queries, each of which specifies a line, ray, or segment ρ and asks for reporting the triangles of $\mathcal{T}$ that ρ intersects (resp., determining whether such a triangle exists). We want the queries to be output-sensitive, whose cost is a small (sublinear) overhead plus a nearly linear term in the output size k. For approximate counting queries, we want to preprocess $\mathcal{T}$ into a data structure, such that given a query ρ as above, it computes the number of triangles of $\mathcal{T}$ that ρ intersects, up to some prescribed small relative error.

(iii) Compute the intersection of two nonconvex polyhedra. The complexity of the intersection can be quadratic in the complexities of the input polyhedra, and we therefore seek an output-sensitive solution, where the running time is a small (subquadratic) overhead plus a term that is nearly linear in k, where k is the complexity of the intersection.

(iv) Detect, count, or report intersections in a set of lines in 3-space. Again, in the reporting version we seek an output-sensitive solution, as above.

(v) Output-sensitive construction of an arrangement of triangles in three dimensions.

All these problems, or variants thereof, have been considered in several works during the 1990s; see [3, 4, 11, 13, 14, 21, 23] for a sample of these works. See also Pellegrini [24] for a recent comprehensive survey of the state of the art in this area.

Pellegrini [23] presents solutions to some of these problems, including efficient data structures (albeit less efficient than ours) for the ray-shooting problem, and also (a) an output-sensitive algorithm for computing the intersection of two nonconvex polyhedra in time $O(n^{8/5+\varepsilon} + k \log k)$, for any $\varepsilon > 0$, where n is the number of vertices, edges, and facets of the two polyhedra and k is the (similarly defined) complexity of their intersection; (b) an output-sensitive algorithm for constructing an arrangement of n triangles in 3-space in $O(n^{8/5+\varepsilon} + k \log k)$ time, where k is the output size; and (c) an algorithm that, in $O(n^{8/5+\varepsilon})$ expected time, counts all pairs of intersecting lines, in a set of n lines in 3-space.

Background. Algorithmic problems that involve lines in three dimensions have been studied for more than 30 years, covering the problems mentioned above and several others. An early study by McKenna and O'Rourke [22] has developed some of the tools and techniques for tackling these problems. Various techniques for ray shooting, and for the related problems of computing and verifying depth orders and hidden surface removal have been studied in de Berg's dissertation [13], and later by de Berg et al. [14]. Another work that developed some of the infrastructure for these problems is by Chazelle et al. [11], who presented several combinatorial and algorithmic results for problems involving lines in 3-space. Agarwal and Matoušek [3] reduced ray shooting problems, via parametric search, to segment emptiness problems (where the query is a segment and we want to determine whether it intersects any input object), and obtained efficient solutions via this reduction. See also [21] and [4] for studies of some additional and special cases of the ray shooting problem.

Most of the works cited above suffer from the "curse' of the four-dimensionality of (the parametric representation of) lines in space, which leads to algorithms whose complexity is inferior to those obtained in our work. Nevertheless, there are a few instances where better solutions can be obtained, such as in [10, 11] and some other works.

Our results. Using the polynomial partitioning technique of [16, 17], we derive more efficient algorithms for the problems listed above. In our first main result, presented in Section 2, we tackle the ray-shooting problem, and construct a data structure on an input set of n triangles,

which requires $O(n^{3/2+\varepsilon})$ storage and preprocessing, so that a ray shooting query can be answered in $O(n^{1/2+\varepsilon})$ time, for any $\varepsilon > 0$. We then extend the technique, in Section 3, to obtain an equally-efficient data structure for the segment-triangle intersection reporting, emptiness, and approximate counting problems, where in the case of reporting the query time bound has an additional term that is nearly linear in the output size.

These are significant improvements over previous results, which, as already noted, have treated the lines supporting the edges of the input triangles and the line supporting the query ray (or segment) as points or surfaces in a suitable four-dimensional parametric space (in many of the earlier works, lines were actually represented as points on the *Klein quadric* in five-dimensional projective space; see [8, 19, 24, 26]). As a result, the algorithms obtained by these techniques were less efficient.

A weakness, or rather an intriguing peculiarity, of our analysis is that it does not provide a desirably sharp tradeoff between storage and query time. To make this statement more precise, the tradeoff that the earlier solutions provide, say for the ray shooting problem for specificity, is that, for n input triangles and with s storage, for s varying between n and n^4, a ray-shooting query takes $O(n^{1+\varepsilon}/s^{1/4})$ time; see, e.g., [24] (the "4" in the exponent comes from the fact that lines in 3-space are represented as objects in four-dimensional parametric space). Thus, with storage $O(n^{3/2+\varepsilon})$, which is what our solution uses, the query time becomes about $O(n^{5/8})$, considerably weaker than our bound.

An ambitious, and maybe unrealistic goal would be to improve the tradeoff so that the query time is only $O(n^{1+\varepsilon}/s^{1/3})$. (This does indeed coincide with the bound that our main result gives, as the storage that it uses is $O(n^{3/2+\varepsilon})$, but this coincidence only holds for this amount of storage.) Although not achieving this goal, still, combining our technique with the known, aforementioned "4-dimensional" tradeoff, we are able to obtain an "in between" tradeoff, which we present in Section 4, where, with s storage, the query cost is

$$Q(n,s) = \begin{cases} O\left(\frac{n^{5/4+\varepsilon}}{s^{1/2}}\right), & s = O(n^{3/2+\varepsilon}), \\ O\left(\frac{n^{4/5+\varepsilon}}{s^{1/5}}\right), & s = \Omega(n^{3/2+\varepsilon}). \end{cases} \tag{1}$$

Note that this tradeoff contains our bounds $(s, Q) = \big(O(n^{3/2+\varepsilon}), O(n^{1/2+\varepsilon})\big)$, as a special case, that at the extreme ends $s = \Theta(n)$, $s = \Theta(n^4)$, of the range of s we get $Q = O(n^{3/4+\varepsilon})$, $Q = O(n^{\varepsilon})$, respectively,[1] as in the older tradeoff, and that the new tradeoff is better for any in-between value of s. A comparison between the two tradeoffs is illustrated in Figure 1. Our improved tradeoff applies to all the problems studied in this paper: the overall cost of processing m queries with n input objects, including preprocessing cost, is

$$\max\Big\{O(m^{2/3}n^{5/6+\varepsilon} + n^{1+\varepsilon}),\ O(n^{2/3}m^{5/6+\varepsilon} + m^{1+\varepsilon})\Big\}, \tag{2}$$

for any $\varepsilon > 0$; for the output-sensitive problems, this bounds the total overhead cost. The first (resp., second) bound dominates when $n \geq m$ (resp., $n \leq m$).

We then present, in Section 5, extensions of our technique for solving the other problems (iii), (iv) and (v) listed above. In all these applications, our algorithms are output-sensitive for the reporting versions, so that the query time bound, or the full processing cost bound, contains an additional term that is nearly linear in the output size. See Section 5.

Due to lack of space, many details of the above results are omitted in this version. They can all be found in the full version [15].

[1] The actual query time in the older tradeoff, with maximum storage, is $Q = O(\log n)$.

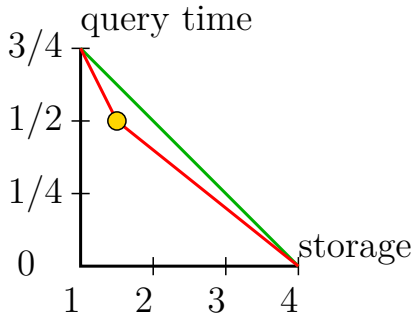

Figure 1 The old tradeoff (green) and the new tradeoff (red). The x-axis is the storage as a function of n, and the y-axis is the query cost. Both axes are drawn in a logarithmic scale.

2 Ray shooting amid triangles

Let $\mathcal{T}$ be a collection of n triangles in $\mathbb{R}^3$. We fix some sufficiently large constant parameter D, and construct a partitioning polynomial f of degree $O(D)$ for $\mathcal{T}$, so that each of the $O(D^3)$ connected components τ of $\mathbb{R}^3 \setminus Z(f)$ (the cells of the partition) is crossed by at most n/D^2 triangle edges. We refer to triangles with an edge that crosses τ as *narrow triangles* (with respect to τ), and refer to the remaining triangles that cross τ (but none of their edges do) as *wide triangles*. We denote the set of narrow (resp., wide) triangles in τ by $\mathcal{N}_\tau$ (resp., $\mathcal{W}_\tau$). The existence of such a partitioning polynomial is implied, as a special case, by the general machinery developed in Guth [16]. An algorithm for constructing f is given in a recent work of Agarwal et al. [2]. It runs in $O(n)$ time, for any constant value of D, where the constant of proportionality depends (polynomially) on D.

For technical reasons, we want to turn any query ray into a bounded segment, and we do it by enclosing all the triangles of $\mathcal{T}$ by a sufficiently large bounding box B_0, and by clipping any query ray to its portion within B_0.

For each (bounded) cell $\tau \subseteq B_0$ of the partition, we take the set $\mathcal{W}_\tau$ of wide triangles in τ, and prepare a data structure for efficient segment-shooting queries into the triangles of $\mathcal{W}_\tau$, by segments that are fully contained in τ. The nontrivial details of this procedure are given in Section 2.1. As we show there, we can construct such a structure with storage and preprocessing $O(|\mathcal{W}_\tau|^{3/2+\varepsilon}) = O(n^{3/2+\varepsilon})$, for any $\varepsilon > 0$ (where the choice of D depends on ε), and each segment-shooting query takes $O(|\mathcal{W}_\tau|^{1/2+\varepsilon}) = O(n^{1/2+\varepsilon})$ time.

The preprocessing then recurses within each such cell τ of the partition, with the set $\mathcal{N}_\tau$ of the narrow triangles in τ. The recursion terminates when the number of input triangles becomes smaller than the constant threshold $n_0 := O(D^2)$, in which case we simply output the list of triangles in the subproblem.

A query, with a ray (now turned into a segment) ρ, emanating from a point q, is answered as follows. We first consider the case where ρ (that is, the line containing ρ) is not fully contained in $Z(f)$, and discuss the (simpler, albeit still involved) case where $\rho \subset Z(f)$, later.

The case where $\rho \not\subset Z(f)$. We use a standard model of algebraic computation, in which computations involving polynomials of constant degree, such as computing (some discrete representation of) their roots, performing comparisons and algebraic computations (of constant degree) with these roots, and so on, can be done exactly in constant time $C(\delta)$, that depends on the degree δ of the polynomial; see, e.g., [6, 7].

Using this model, we first locate the cell of the partition that contains the starting endpoint q of the segment ρ, in constant time (recalling that D is a constant). One way of doing this is to construct the *cylindrical algebraic decomposition (CAD)* of $Z(f)$ (see [12,25]), associate with each cell σ of the CAD the cell of $\mathbb{R}^3 \setminus Z(f)$ that contains it (or an indication that σ is contained in $Z(f)$), and then search with q in the CAD, coordinate by coordinate (see, e.g., [2] for more details). We then find, in constant time, the $t = O(D)$ points of intersection of ρ with $Z(f)$, and sort them into a sequence $P := (p_1, \ldots, p_t)$ in their order along ρ; we assume that $p_t \in \partial B_0$, and ignore the suffix of ρ from p_t onwards. The points in P partition ρ into a sequence of segments, each of which is a connected component of the intersection of ρ with some cell. The first segment is $e_1 = qp_1$, the subsequent segments are $e_2 = p_1p_2$, $e_3 = p_2p_3, \ldots, e_t = p_{t-1}p_t$. We denote by τ_i the cell containing the i-th segment, for $i = 1, \ldots, t$ (a cell can appear several times in this sequence). See Figure 2.

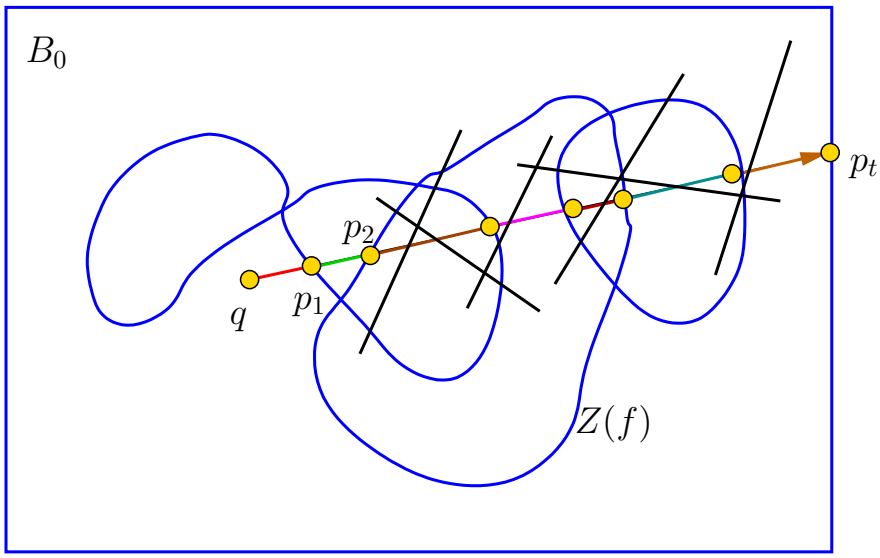

Figure 2 A two-dimensional rendering of the the general structure of the ray-shooting mechanism.

We now process the segments e_i in order. For each segment e_i, let τ_i denote the partition cell that contains e_i. We first perform a ray-shooting (or rather a segment-shooting) query in the structure for $\mathcal{W}_{\tau_i}$ with the segment e_i. As already mentioned (and will be described in Section 2.1), this step can be performed in $O(n^{1/2+\varepsilon})$ time, with $O(n^{3/2+\varepsilon})$ storage and preprocessing, for any $\varepsilon > 0$. We then query with e_i in the substructure recursively constructed for $\mathcal{N}_{\tau_i}$. If at least one of the two queries succeeds, i.e., outputs a point that lies on e_i, we report the point nearest to the starting point of e_i, and terminate the whole query. If both queries fail, we proceed to the next segment e_{i+1} and repeat this step. If the mechanism fails for all the segments, we report that ρ does not hit any triangle of $\mathcal{T}$.

The case where $\rho \subset Z(f)$. We use the cylindrical algebraic decomposition (CAD) of $Z(f)$ (see [12,25]), already discussed earlier. One of its by-products is a *stratification* of $Z(f)$, which is a decomposition of $Z(f)$ into pairwise disjoint relatively open patches of dimensions 0, 1, and 2, called *strata* (each stratum is a cell of the CAD), so that each of the two-dimensional strata is xy-monotone and its relative interior is free of any singularities of $Z(f)$, and $Z(f)$ is the union of the closures of these two-dimensional strata, ignoring possible components of $Z(f)$ of dimension at most 1. We compute the intersection arcs

$\gamma_\Delta := Z(f) \cap \Delta$, for $\Delta \in \mathcal{T}$, and distribute each arc amid the closures of the two-dimensional strata that it traverses. We then project the closure of each two-dimensional stratum σ onto the xy-plane, including the portions of the arcs γ_Δ that the closure contains, and preprocess the resulting collection of $O(n)$ algebraic arcs, each of degree $O(D) = O(1)$, into a planar ray-shooting data structure, whose details are omitted in this version, and can be found in the full version [15].[2] As we show there, we can answer a ray-shooting query in $O(n^{1/2+\varepsilon})$ time, using $O(n^{3/2+\varepsilon})$ storage, for any $\varepsilon > 0$, where the constants of proportionality depend on ε, as does the choice of D. The overall storage complexity, over all the (projected) strata of $Z(f)$, is thus $O(n^{3/2+\varepsilon})$, and the overall query time, over all strata met by the query ray ρ, is $O(n^{1/2+\varepsilon})$, for a larger constant of proportionality (that depends on ε).

The recursion on D terminates when the query ray comes to lie on the zero set of the current partitioning polynomial. We solve the problem in such a recursive instance using a (nonrecursive) procedure, as detailed in the full version [15], and terminate the (current branch of the) recursion. That is, the leaves of the D-recursion tree represent either constant-size subproblems or subproblems on the zero set of the current partitioning polynomial, and the inner nodes represent subproblems of shooting within the partition cells.

Analysis. The correctness of the procedure is fairly easy to establish. Denote by $S(n)$ the maximum storage used by the structure for a set of at most n triangles, and denote by $S_0(n)$ (resp., $S_1(n)$) the maximum storage used by the auxiliary structure for a set of at most n wide triangles in a cell of the partition, as analyzed in Section 2.1 (resp., for a set of at most n intersection arcs on $Z(f)$, which we process for planar ray-shooting (see the full version [15]). Then $S(n)$ obeys the recurrence

$$S(n) = O(D^3)S_0(n) + S_1(n) + O(D^3)S(n/D^2), \tag{3}$$

for $n > n_0$, and $S(n) = O(n)$ for $n \le n_0$, where $n_0 := cD^2$, for a suitable constant $c \ge 1$. We show, in the respective Section 2.1 and the full version [15], that $S_0(n) = O(n^{3/2+\varepsilon})$ and $S_1(n) = O(n^{3/2+\varepsilon})$, for any $\varepsilon > 0$, where both constants of proportionality depend on D and ε, from which one can easily show that the solution of (3) is $S(n) = O(n^{3/2+\varepsilon})$, for a slightly larger, but still arbitrarily small $\varepsilon > 0$; to achieve this bound, we need to take D to be $2^{\Theta(1/\varepsilon)}$, as will follow from our analysis. Regarding the bound on the preprocessing time $T(n)$, we obtain a similar recurrence as in (3), namely,

$$T(n) = O(n) + O(D^3)T_0(n) + T_1(n) + O(D^3)T(n/D^2),$$

where the non-recursive linear term is the time to compute the polynomial f, and $T_0(n)$, $T_1(n)$ are defined in an analogous manner as above, and have similar upper bounds as $S_0(n)$, $S_1(n)$ (see Section 2.1 and the full version [15]).

Similarly, denote by $Q(n)$ the maximum query time for a set of at most n triangles, and denote by $Q_0(n)$ (resp., $Q_1(n)$) the maximum query time in the auxiliary structure for a set of at most n wide triangles in a cell of the partition (resp., for a set of at most n intersection arcs within $Z(f)$, when the query ray lies on $Z(f)$). Then $Q(n)$ obeys the recurrence (recall that the recursion terminates when the query ray lies on the zero set)

$$Q(n) = \max\left\{O(D)Q_0(n) + O(D)Q(n/D^2),\ Q_1(n)\right\}, \tag{4}$$

[2] This specific planar ray-shooting problem, amid constant-degree algebraic arcs, has not received full attention in the past, although several algorithms have been proposed, mostly with suboptimal solutions. Consult, e.g., Table 2 in Agarwal [1]; see also [5, 20].

for $n > n_0$, and $Q(n) = O(n) = O(1)$ for $n \leq n_0$. Again, the analysis in Section 2.1 and the full version [15] shows that $Q_0(n) = Q_1(n) = O(n^{1/2+\varepsilon})$, for any $\varepsilon > 0$ (where the choice of D depends on ε, as above), from which one can easily show, using induction on n, that the solution of (4) is $Q(n) = O(n^{1/2+\varepsilon})$, for a slightly larger but still arbitrarily small $\varepsilon > 0$. The main result of this section is therefore:

▶ **Theorem 1.** *Given a set T of n triangles in three dimensions, and a prescribed parameter $\varepsilon > 0$, we can process T into a data structure of size $O(n^{3/2+\varepsilon})$, in time $O(n^{3/2+\varepsilon})$, so that a ray shooting query amid these triangles can be answered in $O(n^{1/2+\varepsilon})$ time.*

2.1 Ray shooting into wide triangles

Preliminaries. In this subsection we present and analyze our procedure for ray shooting in the set $\mathcal{W}_\tau$ of the wide triangles in a cell τ of the partition. We then present, only in the full version [15], a different approach that yields a procedure for ray shooting within $Z(f)$. Both procedures have the performance bounds stated in Theorem 1. The efficiency of our structures depends on D being a constant, since the constants of proportionality depend polynomially (and rather poorly) on D.

We thus focus now on ray shooting in a set of wide triangles within a three-dimensional cell τ of the partition. To appreciate the difficulty in solving this subproblem, we make the following observation. A simple-minded approach might be to replace each wide triangle $\Delta \in \mathcal{W}_\tau$ by the plane h_Δ supporting it. Denoting the set of these planes as $\mathcal{H}_\tau$, we could then preprocess $\mathcal{H}_\tau$ for ray-shooting queries, each of which specifies a query ray ζ and asks for the first intersection of ζ with the planes of $\mathcal{H}_\tau$. Using standard machinery (see, e.g. [1]), this would result in an algorithm with the performance bounds that we want. However, this approach is problematic, since, even though Δ is wide in τ, h_Δ could intersect τ in several connected components, some of which lie outside Δ. See Figure 3 for an illustration. In such cases, ray shooting amid the planes in $\mathcal{H}_\tau$ is not equivalent to ray shooting amid the triangles of $\mathcal{W}_\tau$, even for rays, or rather portions thereof, that are contained in τ.

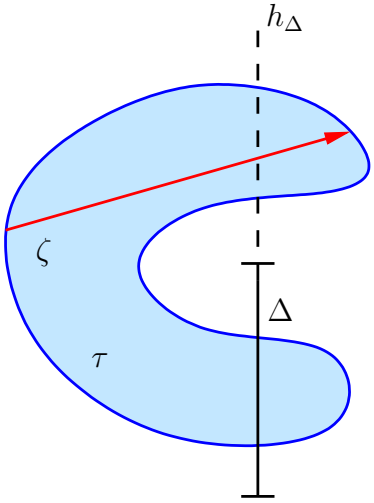

Figure 3 Wide triangles cannot be replaced by their supporting planes for ray shooting within τ.

Our solution is therefore more involved, and proceeds as follows.

Canonical sets of wide triangles. Consider first, for exposition sake, the case where the starting point of the shooting segment lies on $\partial\tau$ (the terminal point always lies on $\partial\tau$). As we will show, for each such segment query, the set of wide triangles in $\mathcal{W}_\tau$ that it intersects can be decomposed into a small collection of precomputed "canonical" subsets, where in each canonical set the wide triangles can be treated as planes (for that particular query segment). The overall size of these sets, over all possible segment queries, is $O(n^{3/2+\varepsilon})$, for any $\varepsilon > 0$.

Actually, to prepare for the complementary case, where the starting point of the query segment lies inside τ, we calibrate our algorithm, so that we control the storage that it uses, and consequently also the query time bound. We introduce a *storage parameter* s, which can range between n and n^2, as a second input to our procedure, and then require that the actual storage and preprocessing cost be both $O(s^{1+\varepsilon})$, for any $\varepsilon > 0$. This relaxed notion of storage offers some simplification in the analysis.

For each $\Delta \in \mathcal{W}_\tau$, let γ_Δ denote the intersection curve of Δ with $\partial\tau$. Note that γ_Δ does not have to be connected – it can have up to $O(D^2)$ connected components, by Harnack's curve theorem [18] (applied on the plane containing Δ). Note also that $\partial\tau$ does not have to be connected, so γ_Δ can have nonempty components on different connected components of $\partial\tau$, as well as several components on the same connected component of $\partial\tau$.

We construct the locus S_τ of points on $\partial\tau$ that are either singular points of $Z(f)$ or points with z-vertical tangency. Since D is constant, S_τ is a curve of constant degree (by Bézout's theorem, its degree is $O(D^2)$). We take a random sample $\mathcal{R}$ of r_0 triangles of $\mathcal{W}_\tau$, where the analysis dictates that we choose $r_0 = D^{\Theta(1/\varepsilon)}$, for the arbitrarily small prescribed $\varepsilon > 0$. Since we have chosen D to be $2^{\Theta(1/\varepsilon)}$, the actual choice of r_0 is $2^{\Theta(1/\varepsilon^2)}$.

Let $\Gamma_\mathcal{R} = \{\gamma_\Delta \mid \Delta \in \mathcal{R}\}$, and let $\mathcal{A}_0 = \mathcal{A}(\Gamma_\mathcal{R} \cup \{S_\tau\})$ denote the arrangement of these curves within $\partial\tau$, together with S_τ. By construction, each face of $\mathcal{A}_0$ is xy-monotone and does not cross any other branch of $Z(f)$ (at a singular point). We partition each face φ of $\mathcal{A}_0$ into pseudo-trapezoids (called trapezoids for short), using a suitably adapted version of a two-dimensional vertical decomposition scheme. Let $\mathcal{A}_0^*$ denote the collection of these trapezoids on $\partial\tau$. The number of trapezoids in $\mathcal{A}_0^*$ is proportional to the complexity of $\mathcal{A}_0$, which is $O_D(r_0^2) = O(1)$.

We assume that the trapezoids are relatively open. For exposition sake, we only handle here two-dimensional trapezoids; lower-dimensional boundary features are handled in the full version [15]. Let ψ_1, ψ_2 be two distinct trapezoids of $\mathcal{A}_0^*$. Let $S(\psi_1, \psi_2)$ denote the collection of all segments e such that one endpoint of e lies in ψ_1, the other endpoint lies in ψ_2, and the relative interior of e is fully contained in the open cell τ. We can parameterize such a segment e by four real parameters, two for the starting endpoint of e and two for its other endpoint. Denote by $\mathcal{F}$ the corresponding (at most) four-dimensional parametric space. Since each of τ, ψ_1, ψ_2 is of constant complexity, $S(\psi_1, \psi_2)$ is a semi-algebraic set in $\mathcal{F}$ of constant complexity, implicitly expressed by the quantified formula

$$S(\psi_1, \psi_2) = \{(p_1, p_2) \mid p_1 \in \psi_1, p_2 \in \psi_2, \text{ and } p_1p_2 \subset \tau\},$$

where p_1p_2 denotes the line-segment connecting p_1 to p_2. Using the singly exponential quantifier-elimination algorithm in [7, Theorem 14.16], we can construct, in $O_D(1)$ time, a quantifier-free semi-algebraic representation of $S(\psi_1, \psi_2)$ of $O_D(1)$ complexity, and we can decompose $S(\psi_1, \psi_2)$ into its connected components, in $O_D(1)$ time as well.

For each segment $e \in S(\psi_1, \psi_2)$, let $\mathcal{T}(e)$ denote the set of all wide triangles of $\mathcal{W}_\tau$ that e crosses. We have the following technical lemma, whose proof can be found in [15].

▶ **Lemma 2.** *In the above notations, each connected component C of $S(\psi_1, \psi_2)$ can be associated with a fixed set $\mathcal{T}_C$ of wide triangles 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 triangle in $\mathcal{T}(e) \setminus \mathcal{T}_C$ crosses either ψ_1 or ψ_2.*

The collection of all these sets $\mathcal{T}_C$, over all connected components C, and all pairs of trapezoids (ψ_1, ψ_2), is part of the whole output collection of canonical sets over τ; the rest of this collection is constructed recursively over the trapezoids ψ of $\mathcal{A}_0^*$.

The algorithm. For each trapezoid ψ of $\mathcal{A}_0^*$, the *conflict list* K_ψ of ψ is the set of all wide triangles that cross ψ. By standard random sampling arguments [9], with high probability, the size of each K_ψ is $O_D\left(\frac{n}{r_0}\log r_0\right)$. (Special cases, involving lower-dimensional boundary features and triangles fully containing trapezoids, are handled in the full version [15].)

For every pair of trapezoids ψ_1, ψ_2, we compute $S(\psi_1, \psi_2)$ and decompose it into its connected components. We pick some arbitrary but fixed segment e_0 from each component C, compute the set $\mathcal{T}(e_0)$ of the wide triangles that cross e_0, and remove from it any triangle that crosses $\psi_1 \cup \psi_2$, thereby obtaining the set $\mathcal{T}_C$. All this takes $O_D(r_0^4 n) = O_D(n)$ time, and the overall size of the produced canonical sets is also $O_D(n)$.

Let s be the storage parameter associated with the problem, as defined earlier, and recall that we require that $n \le s \le n^2$. Each canonical set $\mathcal{T}_C$ is preprocessed into a data structure that supports ray shooting in the set of *planes* $\mathcal{H}_C = \{h_\Delta \mid \Delta \in \mathcal{T}_C\}$, where h_Δ is the plane supporting Δ. We construct these structures so that they use $O(s^{1+\varepsilon})$ storage (and preprocessing), for any $\varepsilon > 0$, and a query takes $O(n\,\mathrm{polylog}(n)/s^{1/3})$ time (see, e.g., [1]).

We now recurse on each conflict list K_ψ, over all trapezoids ψ of $\mathcal{A}_0^*$. Each subproblem uses the same parameter r_0, but now the storage parameter that we allocate to each subproblem is only s/r_0^2. We keep recursing until we reach conflict lists of size close to n^2/s. More precisely, after j levels of recursion, we get a total of at most $(c_0 r_0^2)^j = c_0^j r_0^{2j}$ subproblems, each involving at most $\left(\frac{c_1 \log r_0}{r_0}\right)^j n$ wide triangles, for some constants c_0, c_1 that depend on D, and thus on ε, but are considerably smaller than $r_0 = D^{\Theta(1/\varepsilon)}$.

We stop the recursion at the first level j^* at which $(c_1 r_0 \log r_0)^{j^*} \ge s/n$. As a result, we have ${r_0}^{j^*} \le s/n$, and we get $c_0^{j^*} {r_0}^{2j^*} = O(s^2/n^{2-\varepsilon})$ subproblems, for any $\varepsilon > 0$, where the choice of D (and therefore also of c_0, c_1 and r_0) depends, as above, on ε. With a suitable choice of D, each of these subproblems involves at most

$$\left(\frac{c_1 \log r_0}{r_0}\right)^{j^*} n = \left(\frac{(c_1 \log r_0)^2}{c_1 r_0 \log r_0}\right)^{j^*} n \le (c_1 \log r_0)^{2j^*} \cdot \frac{n^2}{s} = \frac{n^{2+\varepsilon}}{s}$$

triangles, for any $\varepsilon > 0$. Hence the overall size of the inputs and of the canonical sets, at all recursive subproblems, is $O\left(\frac{s^2}{n^{2-\varepsilon}}\right) \cdot \frac{n^{2+\varepsilon}}{s} = O(sn^{2\varepsilon}) = O(s^{1+\varepsilon})$, for a slightly larger $\varepsilon > 0$.

Note that the canonical sets that we encounter when querying with a fixed segment e are not necessarily pairwise disjoint. This is because the sets K_{ψ_1} and K_{ψ_2} are not necessarily disjoint (they are disjoint of $\mathcal{T}_C$, though). This does not pose a problem for ray shooting queries, but will be problematic for *counting queries*; see Section 3.

At the bottom of the recursion, each subproblem contains at most $n^{2+\varepsilon}/s$ wide triangles, which we merely store in the structure. As just calculated, the overall storage that this requires is $O(s^{1+\varepsilon})$, for a slightly larger ε, as above. We obtain the following recurrence for the overall storage $S(N_W, s_W)$ for the structure constructed on N_W wide triangles, where s_W denotes the storage parameter allocated to the structure (at the root $N_W = n$, $s_W = s$).

$$S(N_W, s_W) = \left\{ \begin{array}{ll} O_D(r_0^4 s_W^{1+\varepsilon}) + c_0 r_0^2 S\left(\frac{c_1 N_W \log r_0}{r_0},\ \frac{s_W}{r_0^2}\right) & \text{for } N_W \ge n^{2+\varepsilon}/s, \\ O(N_W) & \text{for } N_W < n^{2+\varepsilon}/s. \end{array} \right\}$$

Throughout the recursion we have $N_W \le s_W \le N_W^2$ (see the full version for details).

Unfolding the first recurrence up to the terminal level j^*, where $N_W < n^{2+\varepsilon}/s$, the sum of the nonrecursive overhead terms $O_D(r_0^4 s_W^{1+\varepsilon})$, over all nodes at a fixed level j, is

$$c_0^j r_0^{2j} \cdot O\left(\frac{s_W^{1+\varepsilon}}{r_0^{2j(1+\varepsilon)}}\right) = O\left(\frac{c_0^j}{r_0^{2j\varepsilon}} s_W^{1+\varepsilon}\right) = O\left(s_W^{1+\varepsilon}\right),$$

by the choice of r_0. Hence, starting the recurrence at $(N_W, s_w) = (n, s)$, the overall contribution of the overhead terms (over the logarithmically many levels) is $O(s^{1+\varepsilon})$, for a slightly larger ε. At the bottom of recurrence, we have, as already noted, $O(s^2/n^{2-\varepsilon})$ subproblems, each with at most $O(n^{2+\varepsilon}/s)$ triangles, so the sum of the terms N_W at the bottom of recurrence is also $O(s^{1+\varepsilon})$. In other words, the overall storage used by the data structure is $O(s^{1+\varepsilon})$. Using similar considerations, one can show that the overall preprocessing time is $O(s^{1+\varepsilon})$ as well, since the time obeys essentially the same recurrence.

Answering a query. To perform a query with a segment e that starts at a point a (that lies anywhere inside τ), we extend e from a backwards, find the first intersection point a' of the resulting backward ray with $\partial\tau$, and denote by e' the segment that starts at a' and contains e. See Figure 4 for an illustration. This takes $O_D(1)$ time. This step is vacuous when e starts on $\partial\tau$, in which case we have $e' = e$.

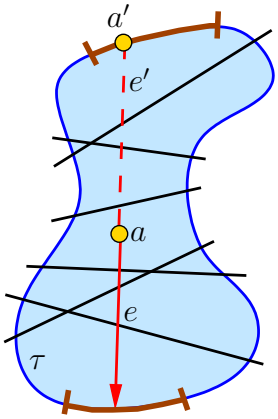

Figure 4 Segment shooting from inside the cell τ: Extending the segment backwards and the resulting canonical set of triangles.

We find the pair of trapezoids ψ_1, ψ_2 that contain the endpoints of e', find the connected component $C \subseteq S(\psi_1, \psi_2)$ that contains e', and retrieve the canonical set $\mathcal{T}_C$. We then perform segment shooting along e from a in the structure constructed for $\mathcal{H}_C$, and then continue recursively in the subproblems for K_{ψ_1} and K_{ψ_2}. We output the triangle that e hits nearest to a, or else report that e does not hit any wide triangle inside τ. See the full version [15] for the case where both endpoints of e' lie in the same trapezoid ψ.

The correctness of the procedure follows from the fact that e' intersects all the triangles of $\mathcal{T}_C$, and thus replacing these triangles by their supporting planes cannot produce any new (false) intersection of any of these triangles with e, and any other wide triangle that e hits must belong to $K_{\psi_1} \cup K_{\psi_2}$.

The query time $Q(N_W, s_W)$ satisfies the recurrence

$$Q(N_W, s_W) = \left\{ \begin{array}{ll} O_D(1) + O\left(\frac{N_W \operatorname{polylog}(N_W)}{s_W^{1/3}}\right) + 2Q\left(\frac{c_1 N_W \log r_0}{r_0}, \frac{s_W}{r_0^2}\right) & \text{for } N_W \geq n^{2+\varepsilon}/s, \\ O(N_W) & \text{for } N_W < n^{2+\varepsilon}/s. \end{array} \right\}.$$

Unfolding the first 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 $N_W \text{polylog}(N_W)/s_W^{1/3}$ is replaced in each of them by the (upper bound) term

$$\frac{\frac{c_1 N_W \log r_0}{r_0}}{\left(\frac{s_W}{r_0^2}\right)^{1/3}} \cdot \text{polylog}(N_W) = \frac{c_1 \log r_0}{r_0^{1/3}} \cdot \frac{N_W \text{polylog}(N_W)}{s_W^{1/3}}.$$

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

$$O\left(\sum_{j \geq 0} \left(\frac{2c_1 \log r_0}{r_0^{1/3}}\right)^j\right) \cdot \frac{n \text{ polylog}(n)}{s^{1/3}} = O\left(\frac{n \text{ polylog}(n)}{s^{1/3}}\right),$$

provided that r_0 is sufficiently large. Adding the cost at the 2^{j^*} subproblems at the bottom level j^* of the recursion, where the cost of each subproblem is at most $n^{2+\varepsilon}/s$, gives an overall bound for the query time of

$$Q(n, s) = O\left(\frac{n \text{ polylog}(n)}{s^{1/3}} + \frac{n^{2+\varepsilon}}{s}\right). \tag{5}$$

Starting with $s = n^{3/2}$, the query time is $O(n^{1/2+\varepsilon})$. We thus obtain

▶ **Proposition 3.** *For a (bounded) cell τ of the polynomial partition, and a set $\mathcal{W}$ of n wide triangles in τ, one can construct a data structure of size and preprocessing cost $O(n^{3/2+\varepsilon})$, so that a segment-shooting query within τ, from any starting point, can be answered in $O(n^{1/2+\varepsilon})$ time, for any $\varepsilon > 0$.*

See the full version [15] for the case where the query ray lies in $Z(f)$. We show:

▶ **Proposition 4.** *For a partitioning polynomial f of sufficiently large constant degree, and a set $\mathcal{W}$ of n triangles, one can construct a data structure of size and preprocessing cost $O(n^{3/2+\varepsilon})$, so that a segment-shooting query with a segment that lies in $Z(f)$, can be answered in $O(n^{1/2+\varepsilon})$ time, for any $\varepsilon > 0$.*

As already concluded, Propositions 3 and 4 complete the proof of Theorem 1.

3 Segment-triangle intersection reporting, emptiness, and approximate counting queries

We extend the technique presented in Section 2 to answer intersection reporting queries amid triangles in $\mathbb{R}^3$. Here too we have a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$, and our goal is to preprocess $\mathcal{T}$ into a data structure that supports efficient intersection queries, each of which specifies a line, ray or segment ρ and asks for reporting the triangles of $\mathcal{T}$ that ρ intersects. In particular, this data structure also supports *segment emptiness* queries, in which we want to determine whether the query segment meets any input triangle. We obtain the following result, whose proof is given in the full version [15].

▶ **Theorem 5.** *Given a collection of n triangles in three dimensions, and a prescribed parameter $\varepsilon > 0$, we can process the triangles into a data structure of size $O(n^{3/2+\varepsilon})$, in time $O(n^{3/2+\varepsilon})$, so that a segment-intersection reporting (resp., emptiness) query amid these triangles can be answered in $O(n^{1/2+\varepsilon} + k \log n)$ (resp., $O(n^{1/2+\varepsilon})$) time, where k is the number of triangles that the query segment crosses.*

Unfortunately, for technical reasons, the method does not extend to segment-triangle intersection (exact) *counting* queries, in which we want to find the (exact) number of triangles that intersect a query segment (or a line or a ray). Briefly, this arises because the canonical sets that a query segment collects are not necessarily pairwise disjoint. As a "compensation", we present a partial solution, which supports queries that approximately count the number of intersections, up to any prescribed relative error $\varepsilon > 0$. Specifically, we obtain the following result, whose proof is given in the full version [15].

▶ **Theorem 6.** *Given a collection of n triangles in three dimensions, and prescribed parameters $\varepsilon, \delta > 0$, where $\delta = \omega(1/n^{\varepsilon})$, we can process the triangles, using random sampling, into a data structure of size $O(n^{3/2+\varepsilon})$, in time $O(n^{3/2+\varepsilon})$, so that, for a query segment e, the number of intersections of e with the input triangles can be approximately computed, up to a relative error of $1 \pm \delta$, with very high probability, in $O(n^{1/2+\varepsilon})$ time.*

4 Tradeoff between storage and query time

In this section, spelled out in the full version [15], we extend the technique in Sections 2 and 3 to obtain a tradeoff between storage (and preprocessing) and query time. A similar tradeoff holds for the problems in Section 5.

Briefly, consider the ray-shooting structure of Section 2, and let s be the storage parameter that we allocate to it, which now satisfies $n \le s \le n^4$. We modify the procedure for ray shooting inside a cell τ by (i) stopping the r_0-recursion at some earlier "premature" level, and (ii) modifying the structure at the bottom of recursion so that it uses the (weaker) ray-shooting technique of Pellegrini [23], instead of a brute-force scanning of the triangles (the current cost of $O(n^2/s)$ is too expensive when s is small). With additional care, we obtain the performance bounds (1) and (2) announced in the introduction.

5 Other applications

5.1 Detecting, counting or reporting line intersections in $\mathbb{R}^3$

It is more convenient, albeit not necessary, to consider the bichromatic version of the problem, in which we are given a set R of n red lines and a set B of n blue lines in $\mathbb{R}^3$, and the detection problem asks whether there exists a pair of intersecting lines in $R \times B$.

An algorithm that solves this problem in $O(n^{3/2+\varepsilon})$ time is easily obtained by regarding the problem as a special degenerate (and much simpler) instance of the ray shooting problem, in which we regard the, say red lines as degenerate triangles (unbounded and of zero area), construct the data structure of Section 2 and query it with each of the blue lines. There exists a red-blue pair of intersecting lines if and only if at least one query has a positive outcome – the corresponding blue query line hits a red line.

Since there are no wide triangles in this special variant, there is no need to construct the auxiliary data structure for wide triangles, as in Section 2.1, and we simply construct the recursive hierarchy of polynomial partitions, with the subset of red lines associated with each cell in each subproblem. A blue query line ℓ is propagated through the cells that it crosses until it reaches bottom-level cells, and we check, in each such cell, whether ℓ intersects any of the red lines associated with the cell.

Handling lines that lie fully in the zero set $Z(f)$ is also an easy task (which can be performed using planar segment-intersection range searching, which also supports counting queries); further details are omitted.

Both correctness and runtime analysis follow easily, as special and simpler instances of the analysis in Section 2. Note that here we do not face the issue of non-disjointness of canonical sets of wide triangles, which has prevented us from extending the technique to segment-triangle intersection counting problems; see Section 3.

5.2 Computing the intersection of two polyhedra

Let K_1 and K_2 be two polyhedra in 3-space, not necessarily convex, each with n edges (so the number of vertices and faces of each of them is $O(n)$). The goal is to compute their intersection $K := K_1 \cap K_2$ in an output-sensitive manner. We note that computing the union $K_1 \cup K_2$ can be done using a very similar approach, within the same time bound.

Details of the procedure are given in the full version [15]. Briefly, the main step is to compute all the vertices of K, each of which is either a vertex of K_1 or of K_2, or an intersection of an edge of one polyhedron and a face of the other. The vertices of the latter type are readily obtain by applying the segment intersection reporting algorithm of Section 3, twice, with the (triangulated) faces of one polyhedron as input and the edges of the other as queries. See [15] for the complete algorithm, which yields:

▶ **Corollary 7.** *Given two arbitrary polyhedra K_1 and K_2 each of complexity $O(n)$, we can compute $K_1 \cap K_2$ in time $O(n^{3/2+\varepsilon} + k \log k)$, where k is the size of the intersection.*

As discussed in the introduction, the overhead term of Pellegrini [23] was $O(n^{8/5+\varepsilon})$.

5.3 Output-sensitive construction of an arrangement of triangles

Let $\mathcal{T}$ be a set of n possibly intersecting triangles in $\mathbb{R}^3$, let $\mathcal{A} = \mathcal{A}(\mathcal{T})$ denote their arrangement, and let k denote its complexity, which, as in Section 5.2, we measure by the number of its vertices, as the number of its other features (edges, faces, and cells) is proportional to k. The goal is to construct $\mathcal{A}$ in an output-sensitive manner with a small, subquadratic overhead. Pellegrini [23] gave such an algorithm that runs in $O(n^{8/5+\varepsilon} + k \log n)$, and the algorithm that we present here reduces the overhead to $O(n^{3/2+\varepsilon})$ time.

As in the previous subsection, we focus on the main step of the algorithm that constructs the features of $\mathcal{A}$ (vertices, edges, and faces) on each triangle of $\mathcal{T}$. The other, simpler complementary steps are discussed in the full version.

Fix a triangle $\Delta \in \mathcal{T}$. We first construct the set of intersection segments $\Delta \cap \Delta'$, for $\Delta' \in \mathcal{T} \setminus \{\Delta\}$. We observe that, for any such segment $e = \Delta \cap \Delta'$, each endpoint of e is either a vertex of Δ, or an intersection of an edge of one triangle with the other triangle.

We therefore take the collection of the $3n$ edges of the triangles of $\mathcal{T}$, and, for each such edge e, apply Theorem 5, which reports all k_e triangles that e meets. This identifies all the intersection segments $\Delta \cap \Delta'$. We then take all the intersection segments within a fixed triangle Δ, and run a sweepline procedure within Δ to obtain the portion of $\mathcal{A}$ on Δ. Gluing these portions to each other, and some additional steps, complete the construction of $\mathcal{A}$.

6 Conclusion

In this paper we have managed to improve the performance of ray shooting amid triangles in three dimensions, as well as of several related problems. The improvement is based on the polynomial partitioning technique of Guth. The improvement is most significant when the storage is about $n^{3/2}$ and the query takes about $n^{1/2}$ time, but one gets an improvement for all values of the storage between n and n^4, except at the very ends of this range. This is a significant improvement, the first in nearly 30 years, in this basic problem.

There are several open questions that our work raises. First, can we improve our tradeoff for all values of storage, beyond the special values of $O(n^{3/2+\varepsilon})$ storage and $O(n^{1/2+\varepsilon})$ query time? Ideally, can we obtain query time of $O(n^{1+\varepsilon}/s^{1/3})$, with s storage, as in the case of ray shooting amid planes? Alternatively, can one establish a lower-bound argument that shows the limitations of our technique?

Another open issue follows from our current inability to extend the technique to counting queries, due to the fact that the canonical sets that we collect during a query are not necessarily pairwise disjoint. It would be interesting to obtain such an extension, or, alternatively, to establish a gap between the performances of the counting and reporting versions of the segment intersection query problem.

Finally, could one obtain similar bounds for non-flat input objects? for shooting along non-straight curves? It would also be interesting to find additional applications of the general technique developed in this paper.

References

1 P. K. Agarwal. Simplex range searching and its variants: A review. In J. Nešetřil M. Loebl and R. Thomas, editors, *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. In *Proc. Internat. Sympos. on Computational Geometry*, pages 5:1–5:14, 2019. Also in `arXiv:1812.10269`.

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

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

5 P. K. Agarwal, M. van Kreveld, and M. Overmars. Intersection queries in curved objects. *J. Algorithms*, 15:229–266, 1993.

6 B. Aronov, E. Ezra, and J. Zahl. Constructive polynomial partitioning for algebraic curves in R^3 with applications. *SIAM J. Comput.*, 49:1109–1127, 2020. Also in *Proc. Sympos. on Discrete Algorithms (SODA)*, 2019, 2636–2648. Also in `arXiv:1904.09526`.

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

8 O. Bottema and B. Roth. *Theoretical Kinematics*. Dover, New York, 1990.

9 B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. A singly exponential stratification scheme for real semi-algebraic varieties and its applications. *Theoretical Computer Science*, 84:77–105, 1991.

10 B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.

11 B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15:428–447, 1996.

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

13 M. de Berg. *Ray Shooting, Depth Orders and Hidden Surface Removal*. Lecture Notes Comput. Sci., 703. Springer Verlag, Berlin, 1993.

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

15 E. Ezra and M. Sharir. On ray shooting for triangles in 3-space and related problems. `arXiv:2102.07310`.

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

17 L. Guth and N.H̃. Katz. On the erdHos distinct distances problem in the plane. *Annals Math.*, 181:155–190, 2015. Also in `arXiv:1011.4105`.

18 C. G. A. Harnack. Über die vielfaltigkeit der ebenen algebraischen kurven. *Math. Ann.*, 10:189–199, 1876.

19 K. Hunt. *Kinematic Geometry of Mechanisms*. Oxford, 1990.

20 V. Koltun. Segment intersection searching problems in general settings. *Discrete Comput. Geom.*, 30:25–44, 2003.

21 J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete Comput. Geom.*, 10:215–232, 1993.

22 M. McKenna and J. O'Rourke. Arrangements of lines in 3-space: A data structure with applications. In *Proc. 4th ACM Sympos. Computational Geometry*, pages 371–380, 1988.

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

24 M. Pellegrini. Ray shooting and lines in space. In J. O'Rourke J. E. Goodman and C. D. Tóth, editors, *Handbook on Discrete and Computational Geometry*, chapter 41, pages 1093–1112. CRC Press, Boca Raton, Florida, Third Edition, 2017.

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

26 J. Stolfi. *Oriented Projective Geometry*. Academic Press, New York, 1991.