Counting and Representing Intersections Among Triangles in Three Dimensions*

Esther Ezra and Micha Sharir
School of Computer Science
Tel Aviv University
{estere,michas}@post.tau.ac.il

Abstract

We present an algorithm that efficiently counts all intersecting triples among a collection T of triangles in $\mathbb{R}^3$ in nearly quadratic time. This solves a problem posed by Pellegrini [31]. Using a variant of the technique, one can represent the set of all κ triple intersections, in compact form, as the disjoint union of complete tripartite hypergraphs, which requires nearly quadratic construction time and storage. Our approach also applies to any collection of planar objects of constant description complexity in $\mathbb{R}^3$, with the same performance bounds. We also prove that this counting problem belongs to the 3SUM-hard family, and thus our algorithm is likely to be nearly optimal in the worst case.

Keywords

Triangles in three dimensions, curve-sensitive cuttings, counting intersections, arrangements, 3SUM-hard problems.

1 Introduction

The main problem solved in this paper is the following. We are given a collection T of n triangles in $\mathbb{R}^3$, and wish to count efficiently the number of nonempty triple intersections among the triangles of T. This problem was posed by Pellegrini [31]. In a variant of the

*Work on this paper has been supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israeli Binational Science Foundation, by a grant from the Israel Science Fund, Israeli Academy of Sciences, for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. A preliminary version of this paper has appeared in *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, 2004, pp. 210–219.

problem, we wish to represent the set of all these intersections in compact form, as the disjoint union of complete tripartite hypergraphs.

Intersection problems are among the most basic problems in computational geometry. Many intersection problems involving geometric objects in the plane have been investigated, such as reporting all intersections in a set of general arcs [9], detecting a *red-blue* intersection between two sets of "red" and "blue" Jordan arcs [6], and counting intersections in a set of segments [2, 11, 23, 31], or in a set of circular arcs [5]. The best algorithm for counting intersections among n line segments takes $O(n^{4/3}\log n)$ time [1, 27]. In contrast, there exist much fewer studies of intersection problems involving objects in three dimensions. de Berg *et al.* [19] provide a procedure for constructing all intersecting pairs in a collection of n triangles in $\mathbb{R}^3$ in an output-sensitive manner. The running time of this procedure is $O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon})$, for any $\varepsilon > 0$, where κ is the number of intersecting pairs.[1] Agarwal and Matoušek [4] provide an algorithm for segment intersection queries, in which we are given a collection of n triangles in $\mathbb{R}^3$ and, for any query segment e, wish to report all the input triangles that e intersects. It follows from their analysis that m queries can be answered in time $O(m^{4/5+\varepsilon}n^{4/5+\varepsilon} + \kappa)$, where κ is the total output size to all queries. Applying this algorithm to the $3n$ edges of n given triangles, we obtain a procedure that reports all κ intersecting pairs of the triangles, in time $O(n^{8/5+\varepsilon}+\kappa)$. A later work of Agarwal *et al.* [3] presents an algorithm for counting or reporting all intersecting pairs in a collection of convex polytopes in three dimensions. Counting requires time $O(n^{8/5+\varepsilon})$, where n is the overall complexity of the input polytopes, and reporting takes $O(n^{8/5+\varepsilon}+\kappa)$ time, where κ is the number of intersecting pairs of polytopes. (This is a potential improvement over the algorithm of [4], since κ counts here the number of intersecting pairs of polytopes, and not of pairs of facets.)

Our Results

We present an efficient algorithm that counts all intersecting triples among a collection T of n triangles in $\mathbb{R}^3$ in nearly quadratic time. Our algorithm is recursive, and exploits 3-dimensional "curve-sensitive" cuttings that were recently introduced by Koltun and Sharir [26]. A cutting of this kind is a standard $(1/r)$-cutting (see [29]) of an arrangement of surfaces (the input triangles in our case), which is also "sensitive" to a set of curves (the triangle edges), in the sense that the overall number of crossings between the curves and the cells of the cutting is small. See [26] and below for more details. More specifically, we recursively partition $\mathbb{R}^3$ using such a cutting. Each triangle is decomposed into portions that lie in different cells of the cutting. We take each such portion Δ, and intersect it with all other triangles, obtaining a system of segments within Δ. We then count the number of intersections between these segments, applying standard techniques that count intersections between lines, and between line-segments and lines in the plane [2]. The recursion is handled in a careful manner that ensures that the algorithm indeed

[1]Many of the bounds stated in this paper are of the form $O(n^{\gamma+\varepsilon})$. They hold for any $\varepsilon > 0$, and the constant of proportionality depends on ε, and typically tends to ∞ as $\varepsilon \to 0$.

runs in nearly quadratic time.

The main technical issue that our algorithm faces is the need to avoid direct computation of the intersections among the segments on each triangle. Doing so, using the best algorithms for this planar problem [1, 27], would result in an overall running time $O(n \cdot n^{4/3} \log n) = O(n^{7/3} \log n)$.

Using a variant of this technique, it is possible to construct a representation of the triple-intersection hypergraph of the triangles in T as the disjoint union of complete tripartite subhypergraphs $\{A_i \times B_i \times C_i\}_{i=1}^{s}$ (where s is the overall number of subhypergraphs), so that $\sum_{i=1}^{s}(|A_i| + |B_i| + |C_i|) = O(n^{2+\varepsilon})$. The construction takes $O(n^{2+\varepsilon})$ time as well.

One motivation for constructing such a compact representation is the ability to sample a random vertex of the arrangement $\mathcal{A}(T)$, without having to construct this arrangement explicitly. This technique has been recently used by the authors [21] to efficiently construct the union of n simplices in three dimensions, when the union is determined by $\xi = o(n)$ simplices. Another application of our results is an algorithm that finds the k-th highest vertex in an arrangement of n triangles in $\mathbb{R}^3$, in time $O(n^{2+\varepsilon})$. This problem extends similar problems posed for collections of segments or lines in the plane. In the case of lines, this is the classical *slope selection* problem, which has been extensively studied in [10, 16, 17, 25, 28]. In particular, the solution in [28] uses a technique of random selection of vertices and searching through them, which is the same general approach followed by our algorithm. (We note, though, that when the input objects are *planes* in 3-space, finding the k-th highest vertex can be found by a simple adaptation of the techniques for the planar version of the problem, without having to use our machinery. This algorithm also runs in nearly quadratic time.)

We also extend our technique to count or represent all intersecting triples among n planar objects of constant description complexity[2] that lie in distinct planes. These extensions also run in nearly quadratic time. The preceding applications, of selecting a random triple intersection and of finding the k-th highest vertex in an arrangement of such objects, can be extended to objects of this kind, with the same asymptotic performance bounds.

Finally, we show that it is unlikely that the triangle intersection counting problem has a subquadratic solution, since it belongs to the 3SUM-hard family [22], and thus our algorithm is likely to be nearly worst-case optimal.

We note that the problem of *reporting* all intersecting triples among the triangles of T is much simpler, and can be trivially solved in $O(n^2 \log n + X)$ time, where X is the overall number of triple intersections in T, as follows. We intersect each triangle $t \in T$ with the other triangles, obtaining $O(n)$ segments on t. We then run a segment intersection reporting procedure (such as those of [12, 30]) on these segments, which constructs all X_t triple intersections that involve t, in time $O(n \log n + X_t)$, for a total of $O(n^2 \log n + X)$ time over all the triangles of T. The phenomenon that reporting is

[2]That is, each object is defined as a Boolean combination of a constant number of polynomial equalities and inequalities of constant maximum degree.

simpler than counting also arises for intersecting segments in the plane.

The paper is organized as follows. In the next section we present a nearly quadratic algorithm that counts all intersecting triples among a collection of n triangles in $\mathbb{R}^3$. In Section 3 we show how these intersections can be represented as the disjoint union of complete tripartite hypergraphs, with an overall storage (and construction time) that is nearly quadratic in the size of the input. In Section 4 we show that the triangle intersection counting problem belongs to the 3SUM-hard family. In Section 5 we extend our algorithm to count intersecting triples among a collection of planar objects of constant description complexity that lie in distinct planes in $\mathbb{R}^3$. We give concluding remarks and suggestions for further research in Section 6.

2 Counting Intersecting Triples Among Triangles in $\mathbb{R}^3$

Given a collection T of n triangles in $\mathbb{R}^3$, we present an algorithm that efficiently counts all intersecting triples among the triangles in T. For simplicity of presentation, we assume that the triangles in T are in general position.

Preliminary Alternative Stage

If the number κ of pairs of intersecting triangles is significantly smaller than $n^{5/3}$ then the problem can be solved in subquadratic time as follows. We first report all κ intersection segments of pairs of triangles in $O(n^{8/5+\varepsilon} + \kappa)$ time, using segment intersection queries in 3-space [4], where we preprocess the input triangles, and query with the triangle edges. These queries produce the set of all intersection points between the triangle edges and other triangles. The intersection segments between the triangles are delimited by these points, and can then be easily reconstructed. We count, on each triangle t, the number of intersections among the κ_t intersection segments that lie on t (where we have $\sum_t \kappa_t = 2\kappa$). Using the algorithm of [1, 27], this takes $O(\kappa_t^{4/3} \log n)$ time, for each triangle t, for a total of $O\left(\sum_t \kappa_t^{4/3} \log n\right) = O(\kappa n^{1/3} \log n)$ time. Thus the overall running time of this algorithm is $O(n^{8/5+\varepsilon} + \kappa n^{1/3} \log n)$, which is subquadratic when $\kappa = o(n^{5/3}/\log n)$.

To detect such favorable situations, we first run this algorithm, as a preliminary stage. If the running time of this step becomes quadratic, we abandon it, and run the main algorithm, presented in detail below.

2.1 Ingredients of the Algorithm

Curve-Sensitive Cuttings

We use a recent result of Koltun and Sharir [26] on the existence of "curve-sensitive" cuttings. In our context, it implies the following result. For any $r \le n$ there exists a $(1/r)$-cutting Ξ for T of size $O(r^{3+\varepsilon})$, which is a partition of $\mathbb{R}^3$ into $O(r^{3+\varepsilon})$ simplices, such that every simplex (also referred to as a *cell* of Ξ) is crossed by at most $\frac{n}{r}$ triangles of T, with the additional property that the number of crossings between the edges of the triangles and the cells of Ξ is $O(n^{1+\varepsilon}r)$. The time needed to construct such a cutting, when r is at most $O(n^{\varepsilon})$, is $O(n^{1+\varepsilon'})$, for any $\varepsilon' > 0$ that is sufficiently larger than ε.

We note that, for the case of triangles, one can obtain such a cutting using a simpler construction than that in [26]. Specifically, the cutting is constructed from a random sample R of $O(r \log r)$ of the planes containing the triangles of T. We form the arrangement $\mathcal{A}(R)$ of R and triangulate each of its cells using the Dobkin-Kirkpatrick hierarchical decomposition of convex polytopes [18], which has the property that a line that crosses a cell C crosses only $O(\log r)$ of its simplices. The random sampling theory of [14, 15, 24] implies that, with high probability, the resulting decomposition is indeed a $(1/r)$-cutting for T.[3] Since a line (or, rather, an edge of a triangle) crosses at most $O(r \log r)$ cells of $\mathcal{A}(R)$ (it has to cross a plane of R to move from one cell to another), it crosses at most $O(r \log^2 r)$ simplices, so the total number of edge-cell crossings is $O(nr \log^2 r)$. A cutting of this kind, together with the distribution of the input triangles among its simplices, can be constructed in time $O(nr^2 \log^2 r)$: We construct $\mathcal{A}(R)$ and hierarchically decompose each of its cells into simplices, in a total of $O(r^3 \log^3 r)$ time [18]. We next compute, for every resulting simplex Δ, the subset T^{Δ} of the triangles of T that cross Δ, in overall time $O(nr^2 \log^2 r)$ [13]. To do so, we trace, for each triangle t of T, all the cells of $\mathcal{A}(R)$ that t crosses. By the Zone Theorem (see [32]), the overall complexity of these cells is $O(r^2 \log^2 r)$, which also bounds (i) the number of simplices into which these cells are decomposed, and (ii) the cost of tracing all these cells (see [32] for similar algorithms). The resulting procedure has the asserted time bound.

However, for more general planar objects that we will consider in Section 5, this simpler approach does not work, and the more general curve-sensitive cutting of [26] is needed.

The Recursive Decomposition—An Overview

We construct an "edge-sensitive" $(1/r)$-cutting Ξ, as described above, with a value of r that will be specified later, and count the intersecting triples in each cell of Ξ separately. Fix a cell Δ of Ξ. We classify each triangle $t \in T$ that intersects Δ as being either *long*

[3] Here, and in several subsequent steps of our algorithms, we construct cuttings based on a random sample of the input objects. For simplicity of presentation, we bypass issues such as verifying that our sample is indeed a good sample, or reducing the size of the sample by the 2-stage sampling method of [13], or replacing the randomized approach by a deterministic one. All these issues are fairly standard by now, and can be applied in our settings as well.

in Δ, if $\partial t \cap \Delta = \emptyset$, or *short*, otherwise. Each intersecting triple in Δ is consequently classified as **LLL**, if all three triangles that form the intersection are long in Δ, **LLS**, if two of these triangles are long and one is short, **LSS**, if one of these triangles is long and two are short, or **SSS**, if all three triangles are short.

In what follows we assume that each triangle (long or short) that crosses Δ is clipped to within Δ. In particular, for any long triangle t, $t \cap \Delta$ is a triangle or a quadrilateral. For short triangles, $t \cap \Delta$ is at most a 7-gon: Since Δ is a simplex, the plane containing t intersects Δ in at most a quadrilateral, and the edges of t contribute at most three additional edges to the cross-section.

We count the number of intersecting triples within each cell Δ_0 by further partitioning Δ_0 into smaller subcells Δ, and recursively derive from each such subcell new subproblems. We partition Δ_0 using the same kind of sensitive $(1/r)$-cutting, for the same r, with respect to the set of long and short triangles in Δ_0, and the set of edges bounding the short triangles in Δ_0 (and crossing Δ_0). Initially, Δ_0 is the entire three-dimensional space, and all triangles are short in Δ_0, but they may become long in further recursive steps.

Let us denote by $N_S = N_S^{\Delta_0}$ the overall number of short input triangles (within a cell Δ_0) and by $N_L = N_L^{\Delta_0}$ the overall number of long input triangles (within Δ_0). During each step of the recursion, after partitioning Δ_0 into smaller subcells Δ, we immediately dispose of any new LLL and LLS intersections within each subcell Δ, using two respective simple algorithms that run in time $O\left((N_L^{\Delta})^2 \log N_L^{\Delta}\right)$ and $O(N_S^{\Delta} N_L^{\Delta} \log N_L^{\Delta})$, respectively. These intersections are not recounted during any further recursive substep. At the bottom of the recursion (when $N_S < \max\left\{\sqrt{N_L}, c\right\}$, for some constant $c \geq 3$), we use two additional simple algorithms that count intersecting triples of types LSS and SSS, which run in time $O({N_S}^3 + N_S N_L \log N_S)$. We note that the goal of the recursive step is only to count efficiently intersecting triples of type LSS and SSS; the (new) intersecting triples of types LLL and LLS are counted before entering the recursive step. (Of course, each recursive step may generate its own LLL and LLS intersections, involving triangles that were short in the input but became long in some of its subproblems. In other words, some of the LSS and SSS intersections within a cell Δ may be counted as LLL or LLS intersections in various recursive subproblems within Δ.)

We first describe these four simple intersection counting algorithms, and then present in detail the complete recursive algorithm, which uses these simple algorithms as subroutines.

Counting Intersections of Type LLL

Let Δ be a simplex cell of (some recursive cutting) Ξ and let L^{Δ} denote the set of clipped long triangles in Δ. Let $N_L = N_L^{\Delta} = |L^{\Delta}|$ denote, as above, the total number of long triangles in Δ. We apply the planar algorithm of Agarwal [2] to each clipped triangle $t \in L^{\Delta}$. That is, we intersect t with all the other (clipped) triangles in L^{Δ}, and count all intersecting pairs within t. Since the boundary of every triangle $t' \in L^{\Delta}$ lies outside

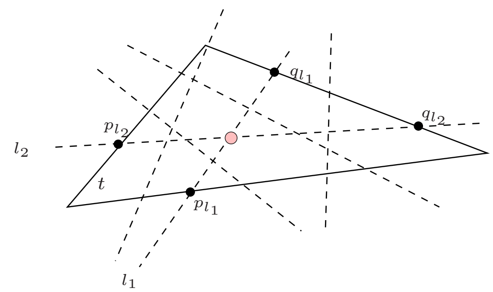

Figure 1: The long triangles that intersect the triangle t, drawn as lines crossing t. Two lines l_1 and l_2 intersect within t if and only if their intersection points p_{l_1}, p_{l_2}, q_{l_1}, q_{l_2} with ∂t interleave along ∂t.

Δ, t' must cross t in a line segment, both of whose endpoints lie on ∂t; see Figure 1. As shown in [2], this problem can be solved in time

$$O\left(\left|L^{\Delta}\right| \log \left|L^{\Delta}\right|\right) = O(N_L \log N_L),$$

by sorting the intersection points of these lines with ∂t along ∂t in a clockwise direction, say, and by counting all pairs whose intersection points appear along ∂t in an *interleaved order*, as illustrated in Figure 1. It follows that the overall running time needed for counting all LLL intersections over all the clipped long triangles within Δ is $O(N_L^2 \log N_L)$. (If we follow this approach verbatim, we need to divide the final count by 3, since each intersection is counted three times; see below for our modified approach.)

Note that once a triangle has become long in a cell Δ, it will remain long in all recursive steps involving subcells of Δ. Since we need to ensure that each LLL intersection is counted only once, we count only intersections that involve at least one *new* long triangle (a triangle that is short in the parent cell of Δ but long in Δ). To do so, we take only new long triangles as the base triangles t, within which the planar counting algorithm is applied. Moreover, we enumerate the new long triangles as $t_1, \ldots, t_{N_L}$, and apply the algorithm, within each t_i, only to the new long triangles t_j, for $j > i$, and to all the old long triangles. With these modifications, the running time of the algorithm just presented is $O(N_L N_L^0 \log N_L)$, where N_L^0 is the number of new long triangles (which are also included among all N_L long triangles). Note that, with all these modifications, each intersection point is now counted only once.

Counting Intersections of Type LLS

We use a similar approach as in the LLL case. Let $N_S = N_S^{\Delta}$ denote the number of short triangles in Δ. We apply the preceding two-dimensional scheme within each short triangle. That is, we intersect each short triangle with all the long triangles, obtaining $O(N_L)$ lines on each such (clipped) triangle. Then we count all intersecting pairs within each short triangle, using the preceding algorithm. The overall running time is thus $O(N_S N_L \log N_L)$. Here too we need to ensure that no intersection is recounted in further recursive substeps. This is done as follows: On each short triangle in Δ, we solve a *bichromatic* version of the problem, which counts all intersections between the new long triangles and the old long triangles. The algorithm for solving this problem is similar to the preceding one, and runs in time $O(N_L \log N_L)$; see [2] for further details. Then we count the intersections involving only new long triangles, using the two-dimensional procedure described in the LLL case. It thus follows that the running time of the modified algorithm remains $O(N_S N_L \log N_L)$.

Counting Intersections of Type LSS

Let S^Δ denote the set of (clipped) short triangles in Δ, so $N_S = N_S^\Delta = |S^\Delta|$. Intersect each (clipped) triangle $t \in S^\Delta$ with all the other triangles of S^Δ and L^Δ. We thus face the problem of counting intersecting pairs involving a long segment (whose endpoints lie on the boundary of t) and a short segment, within every triangle $t \in S^\Delta$. Note that each such problem has an input of $O(N_S)$ short segments and $O(N_L)$ long segments. Since the short segments are confined to within t, we may replace the long segments by their containing lines, without affecting the set of intersecting pairs. The problem can then be solved in $O({N_S}^2 + N_L \log N_S)$ time, using an approach presented in [2], in which we construct the arrangement of the lines dual to the endpoints of the primal segments (representing short triangles), and then locate in this arrangement all points that are dual to the primal lines (representing long triangles). Since each face of the arrangement consists of points dual to lines that cross a fixed set of segments, this easily yields the count of the intersections between the (primal) segments and the (primal) lines.

To make sure that each intersection is counted only once, we enumerate the short triangles as $t_1, \ldots, t_{N_S}$, and make each triangle t_i process only short segments that are formed by its intersections with triangles t_j with $j > i$. Thus, the overall running time of the algorithm, for a fixed cell Δ, is $O\left({N_S}^3 + N_S N_L \log N_S\right)$.

Counting Intersections of Type SSS

We count all intersecting triples of type SSS using a brute-force algorithm which examines all triples in time $O(N_S^3)$. Note that this bound is subsumed by the bound on the time needed to count LSS intersections.

The Overall Recursive Algorithm

Each step of the algorithm involves a simplex Δ_0, which, in the initial step, is the entire 3-space, and in further recursive steps is a cell of a cutting of some larger simplex. The algorithm receives as input a set of N_S short triangles and a set of N_L long triangles clipped to within Δ_0.

If $N_S \le \max\left\{\sqrt{N_L}, c\right\}$, for some constant $c \ge 3$, we stop the recursion and compute the number of LSS and SSS intersections, using the explicit algorithms described above.

Note that, in this case, there is no need to count intersecting triples of type LLL and LLS, since all intersecting triples of these types have already been counted in the preceding step that has processed the parent cell of Δ_0.

If $N_S > \max\left\{\sqrt{N_L}, c\right\}$, we first compute a $(1/r)$-cutting Ξ of the arrangement of all long and short triangles within Δ_0, which is also sensitive to the edges of the short triangles, in the sense defined above. For technical reasons, we need to construct a sensitive $(1/r)$-cutting of Δ_0 that has the property that each subcell Δ of Δ_0 is crossed by at most $N_S^{\Delta_0}/r$ short triangles in Δ_0 and by at most $N_L^{\Delta_0}/r$ long triangles in Δ_0. This problem can be solved by sampling two subsets of $O(r \log r)$ triangles each, one from the long triangles in Δ_0 and one from the short ones. The standard ε-net theory [24] implies that the resulting cutting has the desired property with high probability.

Next we count all LLL and LLS intersections within each subcell Δ of Δ_0, that involve at least one new long triangle (a triangle that is short in Δ_0 but long in Δ), using the algorithms described above. We then continue to solve the problem recursively in every cell $\Delta \in \Xi$, applying the analysis presented in detail below.

Since there are only $O(N_S^{1+\varepsilon} r)$ crossings between short triangles and the cells of Ξ,[4] it follows that, for any $r^2 \le s \le r^{3+\varepsilon}$, the number of cells in Ξ that are crossed by at least $\frac{N_S^{1+\varepsilon} r}{s}$ short triangles is at most $O(s)$. (The case $s < r^2$ cannot arise, since each cell of Ξ is intersected by at most $\frac{N_S}{r}$ short triangles of T, due to the sampling that we have used.) We partition the set of all cells in Ξ into at most $\log\left(\frac{M}{r^2}\right)$ subsets, where M is the overall number of cells in Ξ (note that $\log\left(\frac{M}{r^2}\right) = O(\log r)$), so that the i-th subset Ξ_i contains $O(2^i r^2)$ cells of Ξ, each of which satisfies (recall that S^Δ denotes the set of short triangles in Δ)

$$\frac{N_S^{1+\varepsilon}}{2^i r} \le |S^\Delta| \le \frac{2N_S^{1+\varepsilon}}{2^i r},$$

for $i = 0, \ldots, \log\left(\frac{M}{r^2}\right)$. Note that $|S^\Delta| = O\left(\frac{N_S^{1+\varepsilon}}{r^{2+\varepsilon}}\right)$ for each of the $O(r^{3+\varepsilon})$ cells Δ in the last subset.

We now create recursive subproblems, one in each cell $\Delta \in \Xi_i$, over all $i = 0, \ldots, \log\left(\frac{M}{r^2}\right)$, where the number of short triangles in Δ is $O\left(\frac{N_S^{1+\varepsilon}}{2^i r}\right)$, and the number of long triangles in Δ is at most $\frac{N_S+N_L}{r}$, because of the cutting property, and because some of the short triangles in the parent cell Δ_0 may have become long in Δ.

We estimate the cost of computing the LLL and LLS intersections within each cell Δ of Ξ in the following crude manner. The number of new long triangles in Δ is at most $\frac{N_S}{r}$, the overall number of long triangles in Δ is at most $\frac{N_S+N_L}{r}$, and the number of short triangles in Δ is at most $\frac{N_S}{r}$. Hence the cost of computing the LLL and LLS intersections within Δ is $O\left(\frac{N_S(N_S+N_L)\log(N_S+N_L)}{r^2}\right)$. Summing over all cells Δ, the overall

[4] We use here the more general and slightly weaker bound of [26] for the number of edge-cell crossings, rather than the slightly improved bound that can be obtained from the Dobkin-Kirkpatrick hierarchical decomposition. This does not affect the asymptotic running time bound, and allows us to extend the analysis essentially verbatim to the case of general planar objects, which is undertaken in Section 5.

running time is

$$O\left(\frac{r^{3+\varepsilon}N_S(N_S+N_L)\log(N_S+N_L)}{r^2}\right) = O\left(r^{1+\varepsilon}N_S(N_S+N_L)\log(N_S+N_L)\right).$$

Let $F(N_S, N_L)$ denote the maximum time needed to count all intersecting triples at a recursive step involving N_S short triangles and N_L long triangles. Then F satisfies the following recurrence:

$$F(N_S,N_L) \le \begin{cases} O\left(r^{1+\varepsilon}N_S(N_S+N_L)\log(N_S+N_L) + (N_S+N_L)^{1+\varepsilon'}\right) & \\ + \sum_{i=0}^{\log\left(\frac{M}{r^2}\right)} O(2^i r^2) F\left(\frac{2N_S{}^{1+\varepsilon}}{2^i r}, \frac{N_S+N_L}{r}\right), & \text{if } N_S > \max\left\{\sqrt{N_L}, c\right\} \\ & \\ O(N_S^3 + N_S N_L \log N_S), & \text{if } N_S \le \max\left\{\sqrt{N_L}, c\right\}, \end{cases}$$

where $c \ge 3$ is constant. The first term is the time needed to count the LLL and LLS intersections in the current recursive step, and the term $O\left((N_S+N_L)^{1+\varepsilon'}\right)$ is the time needed to construct the curve-sensitive cutting, for any $\varepsilon' > 0$, whose choice is correlated with the choice of r.

To solve the recurrence, for a given $\varepsilon > 0$, we substitute $r = (N_S+N_L)^{c'\varepsilon''}$, for an appropriate constant $c' > 0$ and for $\varepsilon'' \ll \varepsilon$. It is then easy to see, using induction on N_S and N_L, that the solution is

$$F(N_S, N_L) = O(N_S(N_S+N_L)^{1+\varepsilon}), \text{ for any } \varepsilon > 0, \tag{1}$$

with a constant of proportionality that depends on ε. (Note that in the case $N_S \le \sqrt{N_L}$, the term $O(N_S^3)$, that appears in the bound for the cost of counting all intersecting triples of types LSS and SSS, is dominated by the term $O(N_S N_L \log N_S)$, so (1) does hold in this case too.)

The algorithm begins with Δ_0 equal to the entire three-dimensional space, and $N_S = n$, $N_L = 0$. Note that at this point there are only intersecting triples of type SSS, so the preceding algorithm will count all of them. (As already remarked, the recursive process will generate the other types of intersections as the space is progressively cut up into subcells.)

In summary, we have shown:

Theorem 2.1 *The number of intersecting triples in a set of n triangles in $\mathbb{R}^3$ can be counted in time*

$$\min\left\{O(n^{8/5+\varepsilon} + \kappa n^{1/3}\log n), O(n^{2+\varepsilon})\right\},$$

where κ is the number of pairs of intersecting triangles.

Remark. We note that by slightly modifying this algorithm, we can solve the following trichromatic variant of the problem in nearly quadratic time:

Theorem 2.2 *Given three sets, T_r of n_r "red" triangles, T_b of n_b "blue" triangles, and T_g of n_g "green" triangles, all in $\mathbb{R}^3$, one can count the number of triples in $T_r \times T_b \times T_g$ with nonempty intersection, in time*

$$\min\left\{O(N^{8/5+\varepsilon} + \kappa N^{1/3}\log N), O(N^{2+\varepsilon})\right\},$$

where $N = n_r+n_b+n_g$, and κ is the number of bichromatic pairs of intersecting triangles.

3 Compact Representation of All Intersecting Triples

Given a collection $T = \{t_1, \ldots, t_n\}$ of n triangles in $\mathbb{R}^3$, we represent the set of all intersecting triples among the triangles of T as a 3-*uniform hypergraph* [8] $H = (T, E)$, where

$$E = \{\{t_i, t_j, t_k\} \quad | \quad 1 \le i < j < k \le n \text{ and } t_i \cap t_j \cap t_k \ne \emptyset\}.$$

The size of the above representation is $\Theta(n^3)$ in the worst case. Our goal is to provide a compact representation for H of nearly quadratic size, that stores the intersection triples only implicitly. As noted in the introduction, one application of such a compact representation is for sampling a random element out of the set of all intersecting triples in T, without having to list all these intersections explicitly. We discuss this and another application, already mentioned in the introduction, at the end of this section.

The compact representation for H that we seek (defined analogously to that in [7]) is a collection $\mathcal{H} = \{H_i = (T_i, E_i)\}_{i=1}^s$, of s subhypergraphs of H, such that

1. Each H_i is a complete tripartite hypergraph, that is, the set T_i of its vertices can be partitioned into three disjoint subsets A_i, B_i and C_i, such that $E_i = A_i \times B_i \times C_i$, and $E_i \subseteq E$.
2. $E = \bigcup_{i=1}^s E_i$.
3. $E_i \bigcap E_j = \emptyset$, for $i \ne j$.

Clearly, the storage needed for such a compact representation is $\sum_{i=1}^s |T_i| = \sum_{i=1}^s (|A_i| + |B_i| + |C_i|)$, since the edges of H are now defined implicitly. We show that the algorithm described in Section 2 can be modified to produce such a compact representation of H, with $\sum_{i=1}^s |T_i| = O(n^{2+\varepsilon})$, and that the running time also remains $O(n^{2+\varepsilon})$. (We remark that our compact representation is somewhat degenerate, in the sense that in each output subhypergraph H_i, one of the three sets of vertices is a singleton.)

As in the preceding section, we first report, as a preliminary stage, all κ intersection segments of pairs of triangles in $O(n^{8/5+\varepsilon} + \kappa)$ time. Then we run the algorithm of Agarwal [2], in a manner similar to that described in Section 2, but slightly modified, so that it produces all intersecting triples as the disjoint union of complete tripartite hypergraphs composed of precomputed canonical subsets of triangles, so that the total size of all these subsets (as well as the running time needed to construct them) is

$O(\kappa n^{1/3+\varepsilon})$, for any $\varepsilon > 0$. (With some care, we can ensure that no intersection point is implicitly constructed more than once — the simple modifications applied in the main algorithm can be used here as well.) Hence, when the number κ of intersection segments is significantly smaller than $n^{5/3}$, this method will yield a compact representation of subquadratic size and the construction time will also be subquadratic. As above, we abandon this alternative computation when its output size (as well as its running time) becomes more than quadratic, and resort to the main algorithm.

The main algorithm follows the same recursive mechanism as in the preceding section, but the four simple counting algorithms (that the recursive algorithm uses as subroutines) are now modified, so that they construct a compact representation for all relevant intersecting triples (instead of counting them). We first describe these simple algorithms.

Representing Intersections of Types LLL and LLS

We describe the compact representation of intersecting triples of type LLS; the intersecting triples of type LLL are handled similarly. Fix a cell Δ. For each (clipped) short triangle t in Δ, let L_t denote the set of lines obtained by intersecting t with all the long triangles in Δ. (Recall that the actual algorithm is slightly modified, to account only for intersections that involve new long triangles; for simplicity of presentation, we ignore this issue in what follows.) It is sufficient to obtain a separate compact representation of all intersecting pairs of lines in L_t, for each short triangle t in Δ, a task that proceeds as follows. Sort the intersection points of the lines in L_t with ∂t, and turn the resulting circular sequence into a linear sequence σ_t by breaking it at some arbitrary point. For each original triangle t_i, for $i = 1, \ldots, N_L = N_L^\Delta$, that intersects t in a line l^i, we write l_1^i (resp., l_2^i) for its first (resp., second) intersection point in σ_t. We want to represent compactly all pairs $\{t_i, t_j\}$, $i \neq j$, that satisfy $l_1^i < l_1^j < l_2^i < l_2^j$ (in the order within σ_t).

This is an instance of standard 2-dimensional orthogonal range searching. For the sake of completeness, we spell out the details. We use a 2-level tree-like structure. The first-level structure T_1 stores the points l_1^i in sorted order. For each node (subtree) v of T_1, we construct a secondary structure $T_2(v)$ that stores all the points l_2^i whose matching points l_1^i are stored at v. We now query with each triangle t_j. We first search with l_1^j in T_1, and find all elements that (strictly) precede l_1^j, represented as the disjoint union of $O(\log N_L)$ subtrees. For each subtree (rooted at some node) v, we go to $T_2(v)$ and search there for all elements that lie (strictly) between l_1^j and l_2^j, again, obtaining them as a collection of $O(\log N_L)$ subtrees. Altogether, t_j "lands" in $O(\log^2 N_L)$ subtrees of the secondary structures. For each such subtree τ, we collect the set A_τ of all triangles of T in Δ that reach it as queries, and output the complete bipartite graph $A_\tau \times B_\tau$, where B_τ is the set of triangles that are stored at τ (more precisely, those triangles t_i whose second intersection points l_2^i are stored there).

The overall size of the vertex sets of the output graphs is $O(N_L \log^2 N_L)$. Indeed, the overall size of all the secondary trees is $O(N_L \log N_L)$, and the overall size of all subtrees of a secondary tree τ with k vertices is $O(k \log k)$, which implies that $\sum_\tau |B_\tau| = O(N_L \log^2 N_L)$. Similarly, since each triangle reaches as a query $O(\log^2 N_L)$ subtrees, we also have $\sum_\tau |A_\tau| = O(N_L \log^2 N_L)$. We now add, for each subtree τ, the tripartite hypergraph $\{t\} \times A_\tau \times B_\tau$ to the output representation. Hence, the overall size of the sets of the compact representation, over all short triangles t in Δ, is $O(N_S^\Delta N_L^\Delta \log^2 N_L^\Delta)$. The time for constructing such a representation has the same upper bound. Note that the output consists of *edge-disjoint* complete tripartite 3-uniform hypergraphs, due to the fact that each triple intersection is represented exactly once—see below, with one of the three vertex sets in each hypergraph being a singleton.

We need to ensure that each triple intersection is represented only once. Proceeding as in Section 2, the algorithm can be modified so that it represents only bichromatic intersections between the new long triangles and all the long triangles. This can be done by constructing the two-level tree-like structure for all the long triangles as above, but querying only with the points obtained by the new long triangles.

Handling LLL intersections is done similarly. We ensure that each triple intersection is represented only once, using similar arguments to those described in the LLL counting algorithm of Section 2. The size of the resulting representation is $O(N_L N_L^0 \log^2 N_L)$, where N_L^0 is defined as in Section 2, and it can be constructed in $O(N_L N_L^0 \log^2 N_L)$ time.

Representing Intersections of Type LSS

Here too we adapt the corresponding algorithm of the preceding section, so that it represents (rather than counts) all intersecting pairs between the $O(N_S)$ short segments and the $O(N_L)$ long segments within every short triangle t in Δ. As in the LSS counting algorithm, to make sure that each intersection is represented only once, we enumerate the short triangles as $t_1, \ldots, t_{N_S}$, and make each triangle t_i process only short segments that are formed by its intersections with triangles t_j with $j > i$. By repeating the following procedure for all short triangles, we obtain a compact representation of all LSS intersections.

Fix a short triangle t, denote by S_t the set of short segments within t, obtained by intersecting t with all the short triangles in Δ (that succeed t in the above enumeration), and by L_t the set of lines within t, obtained by intersecting t with all the long triangles in Δ. Denote by S_t^* the set of the double wedges dual to the segments in S_t, and by L_t^* the set of points dual to the lines in L_t. We wish to obtain a compact representation of the set of all pairs (p, w), where $p \in L_t^*$, $w \in S_t^*$, and $p \in w$. This can be done using standard range searching techniques, which, for the sake of completeness, we spell out next.

We construct a $(1/r)$-cutting Π for the double wedges in S_t^*, for a sufficiently large constant parameter r, and locate all the points of L_t^* in the cells of Π. We subdivide, if needed, each cell in Π into smaller subcells, each containing at most $|L_t^*|/r^2$ points of L_t^* in its interior. Let Π' denote this new set of cells (it is easily seen that this decomposition does not asymptotically increase the number of cells in Π', and thus $|\Pi'| = O(r^2)$). We then compute, for each cell $\pi \in \Pi'$, the set of all the double wedges of S_t^* that fully contain π. The overall running time of this step is $O(r^2|S_t^*| + |L_t^*| \log r)$. Let us denote by $L_t(\pi)$ the set of lines of L_t whose dual points lie in the interior of π, and by $S_t(\pi)$ the set of segments in S_t whose dual double wedges contain π. We add $\{t\} \times L_t(\pi) \times S_t(\pi)$ to the output representation. Since each double wedge may contain $O(r^2)$ cells in its interior, and since $\sum_{\pi\in\Pi'} |L_t(\pi)| = |L_t^*|$, it follows that the overall size of the vertex sets of this compact representation, at this stage, is

$$\sum_{\pi\in\Pi'} (1 + |L_t(\pi)| + |S_t(\pi)|) = O(r^2|S_t^*| + |L_t^*|)$$

(we add 1 for each cell of Π', since $\{t\}$ is also part of the representation). We now recursively continue to construct such a compact representation within each cell π of Π', where the subproblem at π involves the at most $|L_t^*|/r^2$ dual points in π and the at most $|S_t^*|/r$ double wedges whose boundaries cross π. The recursion is stopped when either N_S or N_L becomes smaller than r. We then report all intersecting pairs in a brute-force manner, as a collection of single-edge hypergraphs. The complexity of the representation, at any such bottom step, is $O(r(N_L + N_S))$.

Let $G(N_S, N_L)$ denote the maximum size of the compact representation of all intersecting pairs at a recursive step involving N_S segments and N_L lines. Then G satisfies the following recurrence:

$$G(N_S, N_L) \le \begin{cases} O\left(r^2 N_S + N_L\right) + O(r^2) G\left(\frac{N_S}{r}, \frac{N_L}{r^2}\right), & \text{if } N_S, N_L > r \\ O\left(r\left(N_S + N_L\right)\right), & \text{if } N_S \le r \quad \text{or } N_L \le r. \end{cases}$$

The solution of this recurrence (for a sufficiently large but constant value of r) is easily seen to be

$$G(N_S, N_L) = O(N_S^{2+\varepsilon} + N_L^{1+\varepsilon}),$$

for any $\varepsilon > 0$. We note that the same bound applies for the time needed to construct this representation. Thus the overall size of the compact representation of all intersecting triples of type LSS within a cell Δ is $O((N_S^\Delta)^{3+\varepsilon} + N_S^\Delta (N_L^\Delta)^{1+\varepsilon})$, for any $\varepsilon > 0$.

Representing Intersections of Type SSS

The compact representation for all intersecting triples of type SSS is constructed in a brute-force manner, by examining all triples, and reporting separately each intersecting triple, as a separate single-edge tripartite hypergraph. The overall size of the representation, and the time needed to compute it, are both $O(N_S^3)$. This bound is subsumed by the bound on the size and the construction time of the representation of the intersecting triples of type LSS.

The Overall Compact Representation

We use the same recursive mechanism as in Section 2. To analyze its performance, we let $F(N_S, N_L)$ denote the time needed to construct the compact representation of all intersecting triples at a recursive step involving N_S short triangles and N_L long triangles. (The storage cost is analyzed in essentially the same manner.) Then F satisfies the following recurrence:

$$F(N_S, N_L) \leq \begin{cases} O\left(r^{1+\varepsilon} N_S (N_S + N_L) \log^2 (N_S + N_L) + (N_S + N_L)^{1+\varepsilon'}\right) & \\ + \quad \sum_{i=0}^{\log\left(\frac{M}{r^2}\right)} O(2^i r^2) F\left(\frac{{N_S}^{1+\varepsilon}}{2^i r}, \frac{N_S + N_L}{r}\right), & \text{if } N_S > \max\left\{\sqrt{N_L}, c\right\} \\ & \\ O(N_S^{3+\varepsilon} + N_S N_L^{1+\varepsilon}), & \text{if } N_S \leq \max\left\{\sqrt{N_L}, c\right\}, \end{cases}$$

for any $\varepsilon > 0$, where $c \geq 3$ is constant, and M and ε' are defined as in Section 2.

Applying arguments similar to those in Section 2, we conclude that the solution of this recurrence is

$$F(N_S, N_L) = O\left(N_S (N_S + N_L)^{1+\varepsilon}\right), \text{ for any } \varepsilon > 0.$$

A slightly simpler version of this recurrence, that results in the same bound, applies for the *size* of the compact representation; we omit the straightforward details. We have thus shown:

Theorem 3.1 *Given a collection T of n triangles in $\mathbb{R}^3$, the set of all intersecting triples among the triangles of T can be represented in compact form, as the disjoint union of complete tripartite hypergraphs, with an overall size of*

$$\min\left\{O(\kappa n^{1/3+\varepsilon}), O(n^{2+\varepsilon})\right\},$$

where κ is the overall number of pairs of intersecting triangles. The time needed to construct this representation is

$$\min\left\{O(n^{8/5+\varepsilon} + \kappa n^{1/3+\varepsilon}), O(n^{2+\varepsilon})\right\},$$

for any $\varepsilon > 0$.

3.1 Applications

Drawing random intersections. We now present two applications of the results obtained so far. In the first application, we wish to draw at random an element from the set of all intersecting triples among a set T of n triangles in $\mathbb{R}^3$. This is easy to do, in $O(\log n)$ time, using the compact representation of this set.[5] We

[5] This algorithm is a variant of another algorithm that the authors have used for drawing a random element of the set of all intersecting pairs of n given segments in the plane [21].

first count the number X of all the intersecting triples among the triangles of T in $\min\left\{O(n^{8/5+\varepsilon} + \kappa n^{1/3}\log n), O(n^{2+\varepsilon})\right\}$ time, where κ denotes, as usual, the overall number of intersecting pairs. If $X = O(n^2)$, we construct all intersecting triples explicitly in time $O(n^2 \log n)$, using the reporting algorithm mentioned in Section 1. In this case, a random intersection can then be drawn in $O(1)$ time. Otherwise, let the compact representation of all intersecting triples be given as $\bigcup_{i=1}^{s}(A_i \times B_i \times C_i)$. We first compute, as a preprocessing step, all the "prefix sums" $X_i = \sum_{i'<i} |A_{i'}| \cdot |B_{i'}| \cdot |C_{i'}|$, for $i = 1, \ldots, s$. We store these sums in a (sorted) array. The cost of this step is $O(n^{2+\varepsilon})$. Next, to draw a random intersecting triple, we draw a random number j between 1 and X, and find in $O(\log n)$ time the index i that satisfies $X_i < j \le X_{i+1}$. We then pick the $(j - X_i)$-th edge of the hypergraph $A_i \times B_i \times C_i$, according to some obvious lexicographical order, and output the corresponding intersecting triple of triangles. Thus, drawing a random intersecting triple takes $O(\log n)$ time, with $O(n^{2+\varepsilon})$ preprocessing time and storage, for any $\varepsilon > 0$.

Finding the k-th highest vertex. We have mentioned in the introduction another application of our machinery: Given a set T of n triangles in $\mathbb{R}^3$, and a parameter k, find the k-th highest vertex of $\mathcal{A}(T)$. This problem can be solved in nearly quadratic time as follows. We construct the compact representation of all triple intersections in T, and draw a random subset V of $O(n^2)$ intersections. We sort the points in V in increasing order of their z-coordinates, and run a binary search through them. At each step of the search, involving a vertex v, we count the number of triple intersections between the triangles of T that lie above v, and use this count to guide the binary search. This counting can be done by clipping each triangle to the halfspace above v, and by applying our counting algorithm to the clipped triangles. When the search terminates, we obtain a horizontal slab Σ, which is known to contain the desired k-th highest vertex, and which contains, with high probability, only $O\left(\frac{n^3}{n^2}\log n\right) = O(n \log n)$ triple intersections. We can then enumerate all of them explicitly, by applying an intersection reporting algorithm to the portions $t \cap \Sigma$, for $t \in T$, in $O(n^2 \log n)$ time, and select from among these points the desired vertex. The total cost of this algorithm is, with high probability, $O(n^{2+\varepsilon})$. Note that when the number κ of intersecting pairs of triangles is small, we can get a subquadratic algorithm that runs in time $O(n^{8/5+\varepsilon} + \kappa n^{1/3+\varepsilon})$, for any $\varepsilon > 0$.

4 Counting Intersecting Triples is 3SUM-Hard

In this section we show that the problem of counting all intersecting triples among triangles in $\mathbb{R}^3$, a problem that we denote as 3COUNTING, belongs to the 3SUM-hard family (see [22]), and thus, the best solution to this problem is likely to require $\Omega(n^2)$ time in the worst case, thus making our algorithm nearly worst-case optimal. Let us consider the following problem, which is among the basic 3SUM-hard problems [22]:

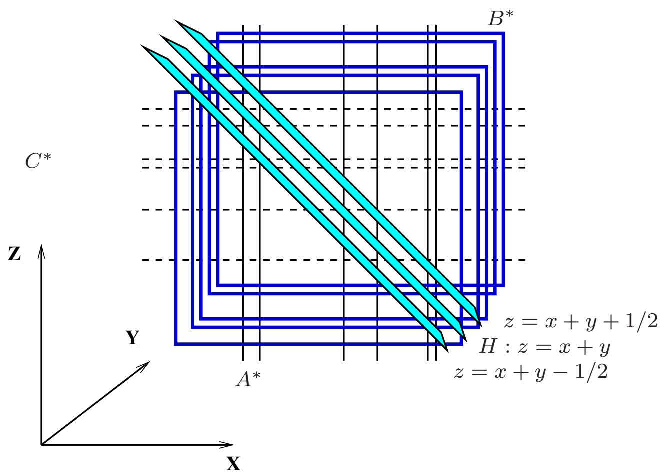

Figure 2: The construction used to reduce 3SUM' to 3COUNTING. The vertical lines are the planes representing the elements of A, the rectangles are the planes representing the elements of B, and the horizontal lines are the planes representing the elements of C. We exclude the region $x + y - 1/2 < z < x + y + 1/2$.

Problem 3SUM'**:**

Given three sets of integers A, B, and C of total size n, are there $a \in A$, $b \in B$, $c \in C$ with $a + b = c$?

We show that the 3SUM' problem is linear-time reducible to 3COUNTING. Given three sets A, B and C of integers, we transform each element $a \in A$ into the plane $h_a : x = a$, each element $b \in B$ into the plane $h_b : y = b$, and each element $c \in C$ into the plane $h_c : z = c$. We denote the three resulting sets of planes by A^*, B^* and C^*, respectively. Every triple of planes $h_a \in A^*$, $h_b \in B^*$ and $h_c \in C^*$ intersects at the point (a, b, c), and the overall number of such intersecting triples is $|A||B||C|$. We now add to the scene the plane $H : z = x + y$. See Figure 2 for an illustration.

The (obvious but) key observation is that there is a triple $a \in A$, $b \in B$, $c \in C$ such that $a + b = c$ if and only if the plane H contains the intersection point of the three planes h_a, h_b and h_c. We thus split each plane $h \in A^* \cup B^* \cup C^*$ into two halfplanes at the intersection line $h \cap H$, resulting in six sets of *open* halfplanes, such that the halfplanes in three subfamilies lie above H and the halfplanes in the other three subfamilies lie below H. We now count all triple intersections among the open halfplanes that lie above H and all triple intersections among the open halfplanes that lie below H. It now follows that the overall number of intersections on both sides of H is strictly smaller than $|A||B||C|$ if and only if there are three planes $h_a \in A^*$, $h_b \in B^*$ and $h_c \in C^*$, such that H contains their intersection point (a, b, c), which is equivalent to the existence of three numbers $a \in A$, $b \in B$, $c \in C$ such that $a + b = c$.

To make the reduction compatible with the type of input assumed by 3COUNTING,

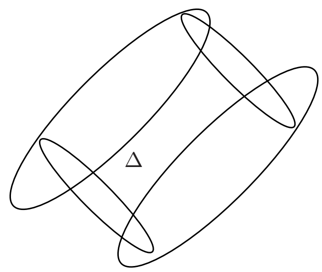

Figure 3: A view from above of four ellipses in $\mathbb{R}^3$. After the vertical walls from their boundaries are erected, nonconvex cells, such as Δ, are generated. Thus, the intersection of Δ with any ellipse that crosses Δ is not convex.

we next modify each open halfplane into a bounded closed triangle. We first intersect each plane in A^*, B^* and C^* with a box bounding all vertices of the arrangement of these planes, obtaining three corresponding sets of rectangles $A^\square$, $B^\square$ and $C^\square$. Next, we observe that the intersection points among all triples $r_a \in A^\square$, $r_b \in B^\square$ and $r_c \in C^\square$ have integer coordinates (and thus the distance between each pair of such points is at least 1). Hence, the region $\{x + y - 1/2 < z < x + y + 1/2\} \setminus H$ does not contain any of these intersection points (see Figure 2 for an illustration). Hence, it is sufficient to intersect each rectangle in $A^\square$, $B^\square$ and $C^\square$ with the two halfspaces $z \leq x + y - 1/2$ and $z \geq x + y + 1/2$, obtaining six families of bounded closed triangles, quadrilaterals, and pentagons, which can be further refined into families of closed bounded triangles, with the same properties as the families of halfplanes constructed earlier.

We note that computing the bounding box of all vertices of the arrangement of the planes in A^*, B^* and C^* is trivially done in $O(n)$ time ($n = |A| + |B| + |C|$), and that the construction of the six families of triangles takes additional $O(n)$ time. Thus we have shown that 3SUM' is linear-time reducible to 3COUNTING, implying that 3COUNTING is a 3SUM-hard problem.

5 Extensions

In this section we extend the algorithms presented in Sections 2 and 3 to count or represent all intersecting triples among n planar simply shaped objects in $\mathbb{R}^3$.

Let $\mathcal{S}$ be a collection of n planar objects in $\mathbb{R}^3$, such that each object $s \in \mathcal{S}$ is bounded by a closed planar curve $c \in \mathbb{R}^3$ of *constant description complexity*; recall that this means that each bounding curve is defined as a Boolean combination of a constant number of polynomial equalities and inequalities of constant maximum degree. We also assume that the objects in $\mathcal{S}$ are in general position, and in particular that *no two of them are coplanar*. In this case, as already discussed in Section 2, we can construct for $\mathcal{S}$ a $(1/r)$-cutting Ξ of size $O(r^{3+\varepsilon})$, which is sensitive to the set of the bounding curves of the elements in $\mathcal{S}$, in the sense that the number of crossings between these bounding

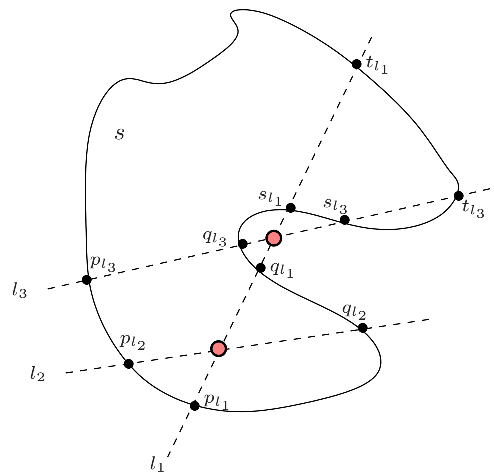

Figure 4: The lines l_1, l_2, l_3 are cross sections within s of corresponding long objects of $\mathcal{S}$. The lines l_1 and l_2 intersect within s since the intersection points p_{l_1}, p_{l_2}, q_{l_1}, q_{l_2} of ∂s with two of their contained segments interleave along ∂s. The lines l_1 and l_3 do not intersect within s since they do not contain segments that are clipped to within s and have interleaving endpoints.

curves and the cells of Ξ is $O(n^{1+\varepsilon}r)$. The time needed to construct this cutting, when r is at most $O(n^{\varepsilon})$, is $O(n^{1+\varepsilon'})$, for any $\varepsilon' > 0$ that is sufficiently larger than ε.

Note that the cells of Ξ, and the clippings of objects of $\mathcal{S}$ to cells of Ξ, need not be convex, even when the original objects in $\mathcal{S}$ are convex. Indeed, as part of the construction of Ξ (as presented in [26]), we draw a random sample R of the bounding curves, and erect vertical walls up and down from each such curve. Thus, a clipped object $s \in \mathcal{S}$ to within a cell $\Delta \in \Xi$ need not be convex; see Figure 3. Nevertheless, since the given objects have constant description complexity (and hence so does each cell $\Delta \in \Xi$), it follows that each clipped object s has constant description complexity, and each element $s' \in \mathcal{S}$ intersects the (clipped) object s in $O(1)$ line-segments (recall that they are assumed to lie in different planes). A clipped object need not be connected, but it has at most $O(1)$ connected components, and we treat each of them separately. In what follows we abuse the notation of s to denote a (connected component of a) clipped object to within a cell Δ of Ξ.

These properties allow us to apply a similar algorithm to that presented in Section 2, in order to count all intersecting triples among the elements of $\mathcal{S}$. More specifically, the recursive mechanism remains the same, and the four simple algorithms can be applied with slight modifications. In the case of counting intersecting triples of type LLL (or LLS), the input to the planar algorithm, that we apply within each (clipped) object s, is the set of all clipped segments that are generated by the intersections of s with an appropriate subset of the long objects. Note that, since we intersect s only with long objects, the endpoints of each intersection segment lie on ∂s. In this case, as in the case of triangles, two clipped segments l_1, l_2 intersect within s if and only if their endpoints interleave along ∂s; see Figure 4 for an illustration. Since each long object s' intersects s in a constant number of segments, the number of input segments to the two-dimensional algorithm is $O(N_L)$, and thus the running time of the planar algorithm remains $O(N_L \log N_L)$.

In the case of counting intersecting triples of type LSS, the input to the two-dimensional algorithm, that we apply within each clipped object, is the set of all clipped short segments and the set of the containing lines of all the clipped long segments. Here however we face a new difficulty. Since the cells Δ need not be convex, extending a clipped long segment into a full line, as we did in the case of triangles, may produce new intersections with short (or long) segments, a situation that is illustrated in Figure 5. We overcome this difficulty as follows. We extend each short segment e to the (unique) longer segment e' that contains e, which is contained in s, and has endpoints lying on ∂s. Let S'_s denote the set of the resulting extended segments, and let L_s denote the set of long segments in s. We first apply an appropriate variant of the *representation* algorithm for LLS (or LLL) intersections (see Section 3), that reports the set of all bichromatic intersecting pairs $(e', \lambda) \in S'_s \times L_s$ as the disjoint union of complete bipartite graphs $S'_i \times L_i$, $i = 1, \ldots, m$. Recall that $\sum_{i=1}^m |S'_i| = O(N_S^\Delta \log^2 N_L^\Delta)$ and $\sum_{i=1}^m |L_i| = O(N_L^\Delta \log^2 N_L^\Delta)$. We now process each subgraph $S'_i \times L_i$ separately. For any pair $(e', \lambda) \in S'_i \times L_i$, we have $e' \cap \lambda \neq \emptyset$. This is easily seen to imply that if we replace λ by its containing line $\bar{\lambda}$, then the original short segment e whose extension is e' intersects λ if and only if e intersects $\bar{\lambda}$; see Figure 5. Hence we are now in the scenario of the LSS algorithm of Section 2, in which we need to count intersections between segments and lines, which can be accomplished using the same duality-based algorithm. The running time is, as in Section 2,

$$O\left(\sum_{i=1}^{m} (|S'_i|^2 + |L_i| \log |S_i|)\right) = O((N_S^\Delta)^2 \log^2 N_L^\Delta + N_L^\Delta \log^2 N_L^\Delta \log N_S^\Delta),$$

where the first term in the bound is obtained by noting that $|S'_i| = O(N_S^\Delta)$, for each $i = 1, \ldots, m$.

In the case of counting intersecting triples of type SSS, we use a brute-force algorithm as in Section 2. The running time of this algorithm is $O(N_S^3)$, because the objects in $\mathcal{S}$ have constant description complexity, and thus each triple is examined in constant time.

With all the four intersection counting routines (for LLL, LLS, LSS, and SSS intersections) properly extended, the algorithm can proceed in essentially the same manner as in Section 2. The slight increase in the running time of the LSS routine is subsumed in the final bound $O(n^{2+\varepsilon})$, as is easy to check.

Arguing as in Section 3, we can use the same mechanism, with appropriate modifications, to derive an algorithm for constructing a compact representation of these intersections, with the same nearly quadratic bound on the storage and the running time.

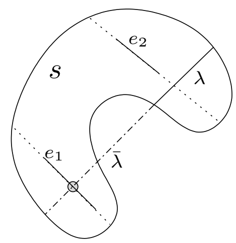

Figure 5: Extending a long segment λ into a full line may create new intersections with short segments (as the intersection with e_1). However, if the extended short segment (such as the extension of e_2) and λ meet, extending λ to a line is safe.

Note that the preliminary alternative algorithm described in Section 2 is not applicable to general planar objects, since it operates only on polygonal objects. We can replace this step by a more complicated procedure based on the technique of [4] for range searching with semi-algebraic sets, to obtain, in time $O(n^{2-\gamma}+\kappa)$, all the κ intersecting objects in S, where $\gamma > 0$ is some constant that depends on the complexity of the objects in S. This yields a total of $O(\kappa)$ intersection segments on the individual objects in S, and we can apply the algorithm of [1, 27] to count the number of (or represent) their intersections, exactly as in Sections 2 and 3. If κ is relatively smaller than $n^{5/3}$, we end up with alternative subquadratic counting algorithms that run in time $O(n^{2-\gamma} + \kappa n^{1/3} \log n)$ (or $O(n^{2-\gamma} + \kappa n^{1/3+\varepsilon})$ when constructing the compact representation). We thus conclude:

Theorem 5.1 *The number of intersecting triples in a set of n planar objects of constant description complexity that lie in different planes in $\mathbb{R}^3$ can be counted in time*

$$\min\left\{O(n^{2-\gamma} + \kappa n^{1/3} \log n), O(n^{2+\varepsilon})\right\},$$

where γ is a positive constant that depends on the complexity of the objects in S. Moreover, these intersecting triples can be represented in a compact form, as the disjoint union of complete tripartite hypergraphs, whose total size is

$$\min\left\{O(\kappa n^{1/3+\varepsilon}), O(n^{2+\varepsilon})\right\}.$$

The time needed to construct this representation is

$$\min\left\{O(n^{2-\gamma} + \kappa n^{1/3+\varepsilon}), O(n^{2+\varepsilon})\right\},$$

for any $\varepsilon > 0$.

Applications. It follows, as in Section 3, that, with $O(n^{2+\varepsilon})$ preprocessing time and storage, we can draw a random intersecting triple of objects of $\mathcal{S}$ in $O(\log n)$ time. Similarly, we can find in $O(n^{2+\varepsilon})$ time the k-highest vertex in an arrangement of n such objects. As in the case of triangles, subquadratic solutions can be obtained when the number κ of intersecting pairs of objects is significantly smaller than $n^{5/3}$.

6 Concluding Remarks and Open Problems

The results obtained in this paper raise several open problems. A challenging open problem is to obtain comparably efficient algorithms for the case where the input objects are not necessarily planar, but are curved surface patches (or closed surfaces) in $\mathbb{R}^3$ of constant description complexity. The three subtasks of counting LLL, LLS, and LSS intersections become considerably harder, because they call for counting the number of intersections between curves and arcs on some curved surface, and the best known algorithms for these tasks are much less efficient than those for lines and line-segments, which we have used above.

Consider, for example, the problem of counting all intersecting triples among n spheres in $\mathbb{R}^3$. In this case, in the LLL subroutine, we need to solve the problem of counting all intersecting pairs among circles and/or long circular arcs (that is, arcs within a patch of a sphere, that completely cross this patch). However, we are not aware of any algorithm for this task that is faster than (an appropriate variant of) the standard algorithm that counts intersections between circular arcs in the plane, and runs in time $O(n^{3/2+\varepsilon})$ [5]. Thus, in this case, our algorithm is not better than a simple-minded algorithm that intersects each sphere with all the others spheres, and uses the two-dimensional algorithm of [5] on each sphere. The running time of this algorithm is thus $O(n^{5/2+\varepsilon})$.

Finally, another challenging problem is to count d-wise intersections among $(d-1)$-simplices in $\mathbb{R}^d$, for $d \geq 4$. Note that the reduction presented in Section 4 can be extended to d-space and thus the d-dimensional problem is dSUM-hard [20], which implies that the best solution to this problem is likely to require $\Omega(n^{\lfloor (d+1)/2 \rfloor})$ time in the worst case. However, we are not aware of any algorithm whose running time is close to this bound. A simple minded algorithm can be performed, for example, by induction on the dimension d, as follows. Given n $(d-1)$-simplices in $\mathbb{R}^d$, we intersect the facets of each simplex s with all the other $n-1$ input simplices, obtaining $O(n)$ subproblems in one dimension lower (with a constant of proportionality that depends on d). We now continue to solve each such subproblem recursively. We stop the recursion when we reach three-dimensional problems, and then solve each of them in nearly quadratic time. It thus follows that the overall running time of this algorithm is $O(n^{d-1+\varepsilon})$, for each $d \geq 3$, where the constant of proportionality depends on d and ε. An open problem is to prove that this bound is nearly optimal, or, alternatively, design an improved algorithm for this problem.

We also note that in higher dimensions there is a wider range of problems, in which one might wish to count (or represent) the number of all k-wise intersections among n $(d-1)$-simplices in d-space, where k can vary from 2 to d. Each of these variants is a challenging open problem.

7 Acknowledgments

The authors would like to thank Dan Halperin for useful discussions regarding the preliminary alternative stage of the algorithms, and to Matthew Katz for a useful discussion concerning the motivation of this problem. The authors also wish to thank two anonymous referees for useful comments.

References

[1] P. K. Agarwal. Partitioning arrangements of lines, part II: applications. *Discrete Comput. Geom.*, 5(6):533–573, 1990.

[2] P. K. Agarwal. *Intersection and Decomposition Algorithms for Planar Arrangements*. Cambridge University Press, New York, USA, 1991.

[3] P. K. Agarwal, M. de Berg, S. Har-Peled, M. H. Overmars, M. Sharir, and J. Vahrenhold. Reporting intersecting pairs of polytopes in two and three dimensions. *Comput. Geom. Theory Appl.*, 23(2):195–207, 2002.

[4] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[5] P. K. Agarwal, M. Pellegrini, and M. Sharir. Counting circular arc intersections. *SIAM J. Comput.*, 22(4):778–793, 1993.

[6] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM J. Comput.*, 19(2):297–321, 1990.

[7] P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete Comput. Geom.*, 23:273–291, 2000.

[8] N. Alon and J. H. Spencer. *The Probabilistic Method.* 2nd Edition, Wiley-Interscience, New York, USA, 2000.

[9] J. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, 28:643–647, 1979.

[10] H. Brönnimann and B. Chazelle. Optimal slope selection via cuttings. *Comput. Geom. Theory Appls.*, 10(1):23–29, 1998.

[11] B. Chazelle. Reporting and counting segment intersections. *J. Comput. System Science*, 32:156–182, 1986.

[12] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1-54, 1992.

[13] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[14] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[15] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4(5):387–421, 1989.

[16] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi. An optimal time algorithm for slope selection. *SIAM J. Comput.*, 18:792–810, 1989.

[17] M. B. Dillencourt, D. M. Mount, and N. S. Netanyahu. A randomized algorithm for slope selection. *Internat. J. Comput. Geom. Appl.*, 2:1–27, 1992.

[18] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.

[19] M. de Berg, L. J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. *Discrete Comput. Geom.*, 15:35–61, 1996.

[20] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28:1198–1214, 1999.

[21] E. Ezra and M. Sharir. Output-sensitive construction of the union of triangles. In *Proc. 15th Annu. ACM-SIAM Sympos. Discr. Alg. (SODA'04)*, pages 413–422, 2004. A revised version to appear in *SIAM J. Comput.*.

[22] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[23] L. J. Guibas, M. Overmars, and M. Sharir. Counting and reporting intersections in arrangements of line segments. Technical Report 434, Dept. Computer Science, New York University, March 1989.

[24] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[25] M. J. Katz and M. Sharir. Optimal slope selection via expanders. *Inform. Process. Lett.*, 47(3):115–122, 1993.

[26] V. Koltun and M. Sharir. Curve-sensitive cuttings. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 136–143, 2003. A revised version to appear in *SIAM J. Comput.*.

[27] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[28] J. Matoušek. Randomized optimal algorithm for slope selection. *Inform. Process. Lett.*, 39(4):183–187, 1991.

[29] J. Matoušek. *Lectures on Discrete Geometry*. Springer Verlag, Heidelberg, 2002.

[30] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[31] M. Pellegrini. On counting pairs of intersecting segments and off-line triangle range searching. *Algorithmica*, 17:380–398, 1997.

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