Counting and Representing Intersections Among Triangles in Three Dimensions*

Esther Ezra
School of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
estere@post.tau.ac.il

Micha Sharir
School of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
and
Courant Institute of Mathematical Science
New York University
New York, NY 10012, USA
michas@post.tau.ac.il

ABSTRACT

We present an algorithm that efficiently counts all intersecting triples among a collection T of n triangles in $\mathbb{R}^3$, in time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. This solves a problem posed by Pellegrini [18]. 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 $O(n^{2+\varepsilon})$ construction time and storage, for any $\varepsilon > 0$. Our approach also applies to any collection of convex 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 (since it is believed that 3SUM-hard problems cannot be solved in subquadratic time).

1. INTRODUCTION

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 [8], detecting a *red-blue* intersection between two sets of "red" and "blue" Jordan arcs [5], and counting intersections in a set of segments [1], or in a set of circular arcs [4]. In contrast, there exist much fewer studies of intersection problems involving objects in three dimensions. In [2], Agarwal *et al.* present an algorithm for counting or reporting all intersecting pairs in a collection of convex polytopes in three dimensions. Counting requires time $O(n^{8/5+\varepsilon})$, for any $\varepsilon > 0$, 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. An earlier work of de Berg *et al.* [12] includes 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. If $\kappa \ll n^{3/2}$, this will take subquadratic time. Once the κ intersection segments of the triangles are constructed, we perform, in batched mode, segment intersection queries between these segments and the triangles. This step too can be implemented in $O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon})$ time, for any $\varepsilon > 0$. Hence, when $\kappa \ll n^{3/2}$, we obtain a subquadratic algorithm for counting (and for representing) all triple intersections.

We are not aware of any previous specific work on counting intersecting triples among n objects in three dimensions, although known solutions for the planar case can be employed to solve (suboptimally) three-dimensional instances. For example, counting the number of intersecting triples among n given triangles in $\mathbb{R}^3$ can be done as follows: For each input triangle t, intersect all other input triangles with t, obtaining at most $n-1$ segments within t, and then count, in time $O(n^{4/3+\varepsilon})$, the number of intersections between them, using the algorithm of Agarwal [1]. Repeating this procedure to each triangle t, the number of intersecting triples is one third of the total count. The overall running time is thus $O(n^{7/3+\varepsilon})$, for any $\varepsilon > 0$. Another approach might be to construct the intersection segments of all pairs of triangles, and then perform repeatedly ray shooting queries along each of them in the collection of input triangles, thereby obtaining all triple intersections. Using the best available algorithm of Agarwal and Matoušek [3], this will take $O(n^{12/5+\varepsilon})$ time, for any $\varepsilon > 0$, which is even worse than the first method. As our paper shows, both solutions are far from being optimal. (However, as already noted above, the latter approach may result in a subquadratic algorithm, when the number of intersection segments formed by the triangles is small. We will use this as a supplementary routine in our solution, to take advantage of such situations.)

*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.

SCG '04, June 9–11, 2004, Brooklyn, New-York, USA.

1.1 Our Result

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. This problem was posed by Pellegrini [18]. Our algorithm is recursive, and exploits 3-dimensional "curve-sensitive" cuttings that were recently introduced by Koltun and Sharir [16]. A cutting of this kind is a standard $(1/r)$-cutting of an arrangement of surfaces (the input triangles in our case), which is also sensitive to a set of curves (the triangles edges), in the sense that the overall number of crossings between the curves and the cells of the cutting is small. See [16] 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 [1]. The recursion is handled in a careful manner that ensures that the algorithm indeed runs in nearly-quadratic time.

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 3-uniform subhypergraphs $\{A_i \times B_i \times C_i\}_{i=1}^{s}$ (where s is the overall number of subhypergraphs), so that $\sum_{i=1}^{s}(|A_i| + |B_i| + |C_i|) = O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. 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 constructing this arrangement explicitly. This technique has been recently used by the authors [13] to efficiently construct the union of n simplices in three dimensions, when the union is determined by $\xi \ll n$ simplices. Another application of this problem is to select the k-th highest vertex in an arrangement of n triangles in $\mathbb{R}^3$. This can be performed using an implicit binary search on the vertices of the arrangement, where in each step we choose a random vertex v from among those that lie in some specified slab, and count the number of vertices that lie below v, in order to determine how to continue the search. This extends similar approaches for the 2-dimensional version of the problem [10, 17]. Using our machinery, the problem can be solved in nearly-quadratic time.

We also extend our technique to count or represent all intersecting triples among n planar convex simply shaped objects that lie in distinct planes.

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 [14], 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 $O(n^2 \log n + \kappa)$ time, where κ 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 line-sweeping procedure on these segments, constructing all κ_t triple intersections that involve t, in time $O(n \log n + \kappa_t)$, for a total of $O(n^2 \log n + \kappa)$ time over all the triangles of T. This phenomenon that reporting is simpler than counting also arises for intersecting segments in the plane.

In the next section we present an 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 3-uniform 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 convex objects of constant description complexity 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.

If the number of pairs of intersecting triangles is significantly smaller than $n^{3/2}$ then the problem can be solved in subquadratic time, using the algorithm of de Berg *et al.* [12], briefly reviewed in the introduction. 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 [16] on the existence of "curve-sensitive" cuttings. In our context, it implies the following. For any $r \leq n$ there exists a $(1/r)$-cutting Ξ for T of size $O(r^{3+\varepsilon})$, for any $\varepsilon > 0$, 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 [16]. 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 [11], which has the property that a line that crosses a cell C crosses only $O(\log r)$ of its simplices. 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 can be constructed in time $O(nr^2 \log^4 r)$: We construct $\mathcal{A}(R)$ and hierarchically decompose each cell Δ of it, in a total of $O(r^3 \log^3 r)$ time [11]. We next compute, for every original cell Δ of $\mathcal{A}(R)$, the subset T^{Δ} of the triangles of T that cross Δ, in overall time $O(nr^2 \log^2 r)$ [9], and then determine the crossings between the triangles in T^{Δ} and the subcells of Δ (constructed by the hierarchical decomposition). The running time of the latter step is $O(\log r)$ for each triangle of T^{Δ} [11], for a total of $O(\frac{n}{r} \log r)$

time over all triangles of T^Δ, and thus for a grand total of $O(nr^2 \log^4 r)$ time over all cells of $\mathcal{A}(R)$. However, for more general planar convex figures, that we will consider in Section 5, this simpler approach does not work, and the more general curve-sensitive cutting of [16] 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* 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,
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, we partition Δ_0 into smaller subcells Δ, and immediately dispose of any new LLL and LLS intersections within each subcell Δ, using two simple algorithms that count all intersecting triples of types LLL and LLS within Δ in time $O\left(\left(N_L^\Delta\right)^2 \log N_L^\Delta\right)$ 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 $O({N_S}^3 + N_S N_L \log N_L)$—see below. We note that the goal of the recursive step is only to count efficiently intersecting triples of types 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.)

For the algorithm to attain the desired efficiency, we need, for each parent cell Δ_0, 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 $\frac{N_S^{\Delta_0}}{r}$ short triangles in Δ_0 and by at most $\frac{N_L^{\Delta_0}}{r}$ 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 [15] implies that the resulting cutting has the desired property.

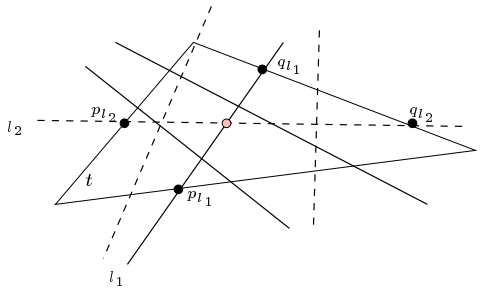

Figure 1: The long triangles that intersect the triangle t, drawn as lines crossing t. Two lines l_1 and l_2 intersect within t if and only if their intersection points p_{l_1}, q_{l_1}, p_{l_2}, q_{l_2} with ∂t interleave along ∂t.

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 [1] to each clipped triangle $t \in L^\Delta$. That is, we intersect t with all the other triangles in L^Δ, and count all intersecting pairs within t. Since the boundary of every triangle $t' \in L^\Delta$ lies outside Δ, t' must cross t in a line segment, both of whose endpoints lie on ∂t; see Figure 1. As described in [1], 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 $O(N_L^2 \log N_L)$.

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 $t_1, \ldots, t_k$, and apply the algorithm, within each t_i, only to the new long triangles t_j, for $j > i$, and to all the old 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 counted among all N_L long triangles).

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 the *bichromatic* version of the problem, which counts all intersections between the new long triangles and all the 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 [1] 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 in $t \in S^\Delta$ with all the other triangles of S^Δ and L^Δ. We thus face the problem of counting intersecting pairs of long segments (whose endpoints lie on the boundary of t) and short segments, 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 $O({N_S}^2 + N_L \log N_L)$ time, using an approach presented in [1], 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 $O\left({N_S}^3 + N_S N_L \log N_L\right)$.

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 compute LSS intersections.

The Overall Recursive Algorithm

Each step of the algorithm involves a simplex Δ_0, which is initially the entire 3-space, or, 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. (We apply the variant that samples $O(r \log r)$ triangles from each of the sets of long triangles and of short triangles.) Then 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 $\Delta \in \Xi$, with some extra care — see below.

Since there are only $O(N_S^{1+\varepsilon} r)$ crossings between short triangles and the cells of Ξ,[1] 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 $\log\left(\frac{M}{r^2}\right) = O((1+\varepsilon)\log r)$), so that in the i-th subset Ξ_i, for $i = 0, \ldots, \log\left(\frac{M}{r^2}\right)$, we have $O(2^i r^2)$ cells of Ξ, each of which satisfies (recall that S^Δ denotes the set of short triangles in Δ)

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

where the last subset of $O(r^{3+\varepsilon})$ cells satisfies $|S^\Delta| = O\left(\frac{N_S^{1+\varepsilon}}{r^2}\right)$, for each cell Δ in this subset.

We now create recursive subproblems, taking into account the number of long triangles in each subcell Δ, as follows. For each cell $\Delta \in \Xi_i$, we partition (arbitrarily) the set of long triangles in Δ into 2^i subsets, each of size at most $\frac{N_S+N_L}{2^i r}$ (note that 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), and process each subset in a separate recursive step. Thus, Δ generates 2^i subproblems, each involving all $O\left(\frac{N_S^{1+\varepsilon}}{2^i r}\right)$ short triangles in Δ, and at most $\frac{N_S+N_L}{2^i r}$ long triangles in Δ. These subproblems are then solved recursively. Note that counting all LLL an LLS intersections within Δ, before proceeding down the recursion, is crucial. Otherwise, in addition to the issues discussed earlier, we might miss intersecting triples of types LLL and LLS that involve long triangles from two different subsets.

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

[1] We use here the more general and slightly weaker bound of [16] 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 convex planar objects.

$\frac{N_S+N_L}{r}$, 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 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:

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

where $c \ge 3$ is constant, and the term $O\left((N_S+N_L)^{1+\varepsilon'}\right)$ is the time to construct the curve-sensitive cutting, for any $\varepsilon' > 0$, whose choice depends on the choice of r.

To solve the recurrence, for a given $\varepsilon > 0$, we substitute $r = n^{c'\varepsilon'}$, for an appropriate constant $c' > 0$. It is then easy to see, using induction on n and choosing ε' appropriately, that the solution is

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

with a constant of proportionality that depends on ε. (Note that in the case $N_S < \sqrt{N_L}$, 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 $O(N_S N_L \log N_L)$.)

Initially, the algorithm begins with Δ 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, but the recursive process will generate the other types of intersections as 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*

$$\min\left\{O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon}), O(n^{2+\varepsilon})\right\},$$

for any $\varepsilon > 0$, where κ is the overall 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:

"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 $\mathbb{R}^3$, efficiently count the number of triples in $T_r \times T_b \times T_g$ with nonempty intersection."

3. COMPACT REPRESENTATION OF ALL INTERSECTING TRIPLES

Given a collection T on n triangles in $\mathbb{R}^3$, we represent the set of all intersecting triples among the triangles of T as a 3-*uniform hypergraph* [7] $H = (T, E)$ where

$$E = \{\{t_i, t_j, t_k\} \quad | \quad t_i, t_j, t_k \in T \text{ and } t_i \cap t_j \cap t_k \neq \emptyset\},$$

for all $1 \le i < j < k \le n$.

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, so that the set of all intersecting triples in T need not be listed in the above explicit manner. As noted in the introduction, one immediate 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 a few applications, already mentioned in the introduction, at the end of this section.

The compact representation for H that we seek (defined analogously to that in [6]) is a collection $\mathcal{H} = \{H_i = (V_i, E_i)\}_{i=1}^s$, of s subhypergraphs of H, such that

1. Each H_i is a complete tripartite 3-uniform hypergraph, that is, the set of its vertices V_i can be partitioned into three disjoint subsets A_i, B_i and C_i, such that any triple of triangles $\{a_i, b_i, c_i\}$, such that $a_i \in A_i$, $b_i \in B_i$, and $c_i \in C_i$, is an edge of H.

2. $E = \bigcup_{i=1}^s E_i$.

3. $E_i \bigcap E_j = \emptyset$, for $i \neq j$.

Clearly, the storage needed for such a compact representation is $\sum_{i=1}^s |V_i|$, 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 $\sum_{i=1}^s |V_i| = O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. We use the same recursive mechanism as in the preceding section, but modify the four simple algorithms (that the recursive algorithm uses as subroutines) so that they construct a compact representation for all relevant intersecting triples (instead of counting them). We first describe these simple algorithms.

(As in the preceding section, we first run as a preliminary stage the algorithm of de Berg *et al.*, which can be easily modified to produce all vertices along each intersection segment as the union of precomputed canonical subsets, so that the total size of all these subsets is $O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon})$, for any $\varepsilon > 0$. (With some care, we can ensure that no vertex is implicitly constructed more than once.) Hence, when the number κ of intersection segments is significantly smaller than $n^{3/2}$, this method will yield a compact representation of subquadratic size. As above, we abandon this alternative computation when its output size becomes more than quadratic, and resort to the main algorithm.)

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. As described in the preceding section, given a cell Δ, the algorithm that counts all intersecting triples of type LLS in Δ applies the planar algorithm on each (clipped) short triangle t in Δ. Let L_t denote the set of the 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. These intersecting pairs are represented as follows. Sort the intersection points of the lines in L_t with the boundary of 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 ($i = 1, \ldots, N_L = N_L^{\Delta}$) that intersects t in a line l^i, we write l_1^i for its first intersection point in σ_t and l_2^i for its second intersection point in σ_t. We want to represent compactly all pairs $\{t_i, t_j\}$, $i \neq j$, that satisfy $l_1^i < l_1^j < l_2^i < l_2^j$.

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, 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 easily 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_j 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, with one of the three vertex sets in each hypergraph being a singleton.

As in Section 2, 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 query 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 (see Section 2 for further details). 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 this 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 (each endpoint of a segment s in the primal plane is transformed into a line in the dual plane, and thus the two endpoints of s form a double wedge in the dual plane, which is the locus of all points dual to lines that intersect s), and by L_t^* the set of points dual to the lines in L_t.

We construct a $(1/r)$-cutting Π for the double wedges in S_t^*, for a sufficiently large constant parameter r. Next, we locate all the points of L_t^* in the cells of Π, and then compute for each cell $\pi \in \Pi$ all the double wedges of S_t^* that 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 a part of the representation). We now subdivide, if needed, each cell in Π into smaller subcells, each containing at most $\frac{|L_t^*|}{r^2}$ 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 now recursively continue to construct such a compact representation within each cell π of Π', where the subproblem at π involves the at most $\frac{|L_t^*|}{r^2}$ dual points in π and the at most $\frac{|S_t^*|}{r}$ double wedges whose boundaries cross π. The recursion is stopped when either N_S or N_L become smaller than r. We then report all intersecting pairs in a brute-force manner. 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:

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

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

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

for any $\delta > 0$. (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 $O((N_S^{\Delta})^{3+\delta} + N_S^{\Delta}(N_L^{\Delta})^{1+\delta})$, for any $\delta > 0$.

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 graph. 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 representation of the intersecting triples of type LSS.

The Overall Compact Representation

We use the same recursive mechanism as in Section 2. In this case, 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. Then F satisfies the following recurrence:

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

for any $\delta > 0$, 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

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

and the same bound applies for the size of the compact representation. 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 3-uniform hypergraphs, with an overall size of*

$$\min\left\{O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon}), O(n^{2+\varepsilon})\right\},$$

for any $\varepsilon > 0$, where κ is the overall number of pairs of intersecting triangles. The time needed to construct this representation has the same bound.

Drawing a Random Intersecting Triple

We now present an application that exploits the compact representation of the intersecting triples among a set T of n triangles in $\mathbb{R}^3$. In this application we wish to draw at random an element from the set of all intersecting triples. This is easy to do, in $O(\log n)$ time, using the compact representation of this set.[2] We first count all the intersecting triples among the triangles of T, in time $\min\left\{O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon}), O(n^{2+\varepsilon})\right\}$, for any $\varepsilon > 0$, where κ denotes the overall number of intersecting pairs. If $\kappa \ll n^{3/2}$, we construct all intersecting triples explicitly in subquadratic time, using the procedure of de Berg *et al.* [12]. Otherwise, if the number of intersecting triples is $O(n^2)$, we construct all intersecting triples explicitly in time $O(n^2 \log n)$, using the reporting algorithm described in Section 1. In this case a random intersection can 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" $\kappa_i = \sum_{i'<i} |A_{i'}| \cdot |B_{i'}| \cdot |C_{i'}|$, for $i = 1, \ldots, s$. We store these sums in a (sorted) array. The cost of this step is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. Next, to draw a random intersecting triple, we draw a random number j between 1 and κ, and find in $O(\log n)$ time the index i that satisfies $\kappa_i < j \leq \kappa_{i+1}$. We then pick the $(j - \kappa_i)$-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 an intersecting triple takes $O(\log n)$ time, with $O(n^{2+\varepsilon})$ preprocessing and storage, for any $\varepsilon > 0$.

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 $\mathcal{A}(T)$. We have noted that our results can be used to solve this problem in nearly quadratic time, using an appropriate form of randomized binary search on the vertices. For lack of space we omit the full details of this application in this version.

4. COUNTING INTERSECTING TRIPLES IS A 3SUM-HARD PROBLEM

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 [14]), and thus, the best solution to this problem is likely to require $\Theta(n^2)$ time in the worst case. We show that the following problem

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 $a + b = c$?"

is linear-time reducible to 3COUNTING.

Given three sets A, B and C of integers, we proceed as follows. We transform each element $a \in A$ (resp., $b \in B$, $c \in C$) to the plane $h_a : x = a$ (resp., $h_b : y = b$, $h_c : z = c$). 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

[2] This algorithm is a variant of another algorithm presented by the authors, for drawing a random element of the set of all intersecting pairs of n given segments in the plane [13].

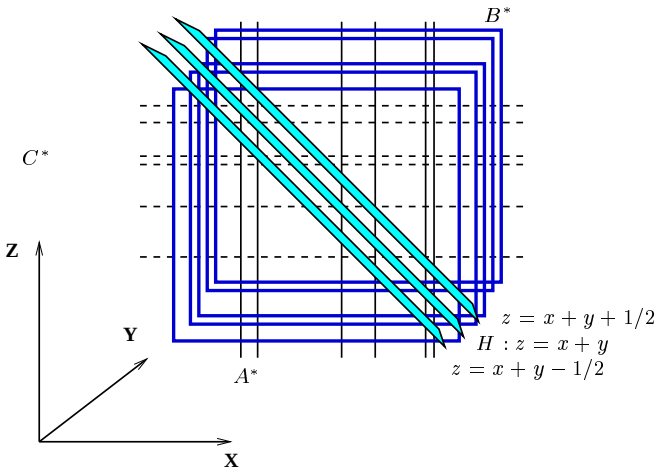

Figure 2: The construction used to reduce 3COUNTING to 3SUM'. The vertical lines are the planes representing the elements of A, the rectangles are the planes representing the elements of B, and the horizontal lines are the planes representing the elements of C. We exclude the region $x+y-1/2 < z < x+y+1/2$.

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. (Since 3COUNTING receives as input *closed* triangles, we actually replace each plane in $A^* \cup B^* \cup C^*$ by its intersections with the two closed halfspaces $z \leq x + y - 1/2$ and $z \geq x + y + 1/2$; it is easy to see that all the triple intersections in the remaining slab $x + y - 1/2 < z < x + y + 1/2$ lie on H itself.) We now count all triple intersections among the resulting closed halfplanes that lie above H, and all triple intersections among the closed 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 $a + b = c$.

We note that the construction of the six families of triangles takes $O(n)$ time. Thus we have shown that 3SUM' $\ll_n$ 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 convex objects in $\mathbb{R}^3$.

Let $\mathcal{S}$ be a collection of n planar convex objects in $\mathbb{R}^3$, such that each object $s \in \mathcal{S}$ is bounded by a closed (and con-

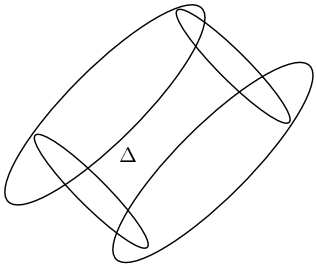

Figure 3: A view from above on four ellipses in $\mathbb{R}^3$. After the vertical walls from their boundaries are erected, a nonconvex cell Δ is generated. Thus, the intersection of Δ with any ellipse that crosses Δ is not convex.

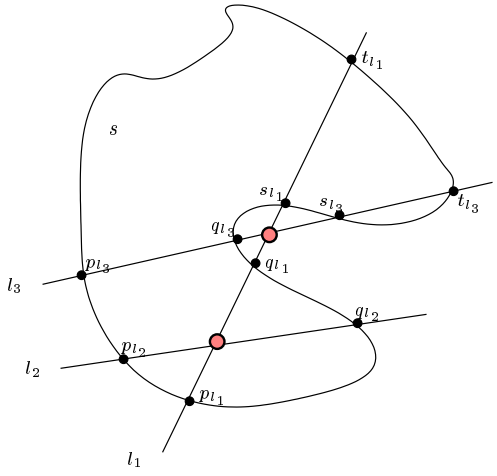

Figure 4: The lines l_1, l_2, l_3 are cross sections within s of corresponding long objects of $\mathcal{S}$. The lines l_1 and l_2 intersect within s since the intersection points p_{l_1}, q_{l_1}, p_{l_2}, q_{l_2} of their contained segments interleave along ∂s. The lines l_1 and l_3 do not intersect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

vex) planar curve $c \in \mathbb{R}^3$ of *constant description complexity*. That is, 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, we can construct for $\mathcal{S}$ a $(1/r)$-cutting Ξ of size $O(r^{3+\varepsilon})$, which is sensitive to all the bounding curves of the elements in $\mathcal{S}$, in the sense that the number of crossings between these bounding curves and the cells of Ξ is $O(n^{1+\varepsilon}r)$, for any $\varepsilon > 0$; see Section 2 and [16]. 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 need not be convex, since, as part of the construction of Ξ, we draw a random sample R of the bounding curves, and erect vertical walls up and down from each such curve; see [16] for further details. 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. Note that 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 (see Section 2 for further details), 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, two clipped segments l_1, l_2 intersect within s if and only if (obeying the same rules as in Section 2, so as to ensure that no intersection is counted twice) 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 (described in Section 2), 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. Note that if s' is long, then it follows from the convexity of s and s' that replacing all the long segments that constitute $s \cap s'$ by the line l that contains them does not generate new (clipped) long segments within s. Since the overall number of short segments is $O(N_S)$ (due to the properties that we have discussed above), and the overall number of lines is N_L, the running time of the two-dimensional algorithm is, as in Section 2, $O(N_S^2 + N_L \log N_L)$.

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.

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.

Note that the preliminary algorithm of [12] is not applicable to general convex planar objects, since it operates only on polygonal objects. We therefore do not run this stage at all. It would be interesting to investigate to what extent this technique can be extended to more general planar regions. We thus conclude:

THEOREM 5.1. *The number of intersecting triples in a set of n planar convex objects of constant description complexity in $\mathbb{R}^3$ can be counted in time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. Moreover, these intersecting triples can be represented in a compact form, as the disjoint union of complete tripartite 3-uniform hypergraphs, using $O(n^{2+\varepsilon})$ time and storage.*

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 nearly quadratic time the k-highest vertex in an arrangement of such objects.

6. CONCLUDING REMARKS AND OPEN PROBLEMS

In this paper we have presented an algorithm that counts all intersecting triples among n triangles in $\mathbb{R}^3$ in nearly-quadratic time. This algorithm can be modified so that it constructs a compact representation of these intersections with an overall size of $\min\left\{O(n^{4/5+\varepsilon}\kappa^{4/5+\varepsilon}), O(n^{2+\varepsilon})\right\}$, for any $\varepsilon > 0$, where κ denotes the overall number of intersecting pairs of triangles. We also proved that the problem of counting all intersecting triples is 3SUM-hard, and thus the algorithm presented in this paper is likely to be nearly worst-case optimal.

We have extended these results to planar convex objects in $\mathbb{R}^3$, and showed that the problem of counting all intersecting triples in this case can be solved in nearly-quadratic time as in the case of triangles. However, this problem becomes more challenging when the input objects are not necessarily planar, but are curved surface patches (or closed shapes) in $\mathbb{R}^3$ of constant description complexity. The three subtasks of counting LLL, LLS or 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 balls 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 long circular arcs (that is, arcs within a patch of a ball, that completely cross this patch). However, we are not aware of any algorithm for this task that is faster than the standard algorithm that counts intersections between circular arcs, and runs in time $O(n^{3/2+\varepsilon})$, for any $\varepsilon > 0$ [4]. Thus, in this case, our algorithm is not better than a simple-minded algorithm that intersects each ball with all the others balls, and uses the

two-dimensional algorithm of [4] on each ball. The running time of this algorithm is thus $O(n^{5/2+\varepsilon})$, for any $\varepsilon > 0$.

Finally, another challenging problem is to count d-wise intersections among $(d-1)$-simplices in $\mathbb{R}^d$, for $d \geq 4$. This can be performed, for example, by induction on the dimension d, as follows. Given n 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 any $\varepsilon > 0$, 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 we wish to count the number of 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 (based on the technique of [12]) of the algorithms.

8. REFERENCES

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

[2] 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.

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

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

[5] 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.

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

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

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

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

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

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

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

[13] E. Ezra and M. Sharir. Output-sensitive construction of the union of triangles. In *Proc. 15th Annu. ACM-SIAM Sympos. Discr. Alg. (SODA'04)*. SIAM, 2004. A revised version in preparation.

[14] 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.

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

[16] V. Koltun and M. Sharir. Curve-sensitive cuttings. In *Proceedings of the 19th Conference on Computational Geometry*, pages 136–143. ACM Press, 2003.

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

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